
Amazon Simple Storage Service

Developer Guide

API Version 2006-03-01



Amazon Simple Storage Service: Developer Guide

Copyright © 2010 Amazon Web Services LLC or its affiliates. All rights reserved.

Table of Contents

Welcome to Amazon S3	1
What's New	4
Introduction to Amazon S3	6
Making Requests	21
AWS Language Support	21
Request Endpoints	21
Authentication	22
Using the REST API	25
Common REST API Elements	26
Authenticating REST Requests	27
REST Access Control Policy	38
Virtual Hosting of Buckets	40
Request Redirection and the REST API	44
Browser-Based Uploads Using POST	46
HTML Forms	47
Upload Examples	54
POST with Adobe Flash	61
Using the SOAP API	61
Common SOAP API Elements	62
Authenticating SOAP Requests	62
Setting Access Policy with SOAP	63
Working with Amazon S3 Buckets	65
Bucket Restrictions and Limitations	66
Bucket Configuration Options	67
Buckets and Regions	67
Requester Pays Buckets	69
Setting the requestPayment Bucket Configuration	70
Retrieving requestPayment Configuration	70
Downloading Objects in Requester Pays Buckets	71
DevPay and Requester Pays	72
Charge Details	72
Buckets and Access Control	72
Billing and Reporting of Buckets	72
Working with Amazon S3 Objects	74
Object Key and Metadata	75
Using the AWS SDK	76
Using the AWS SDK for Java	76
Using the AWS SDK for .NET	77
Using the AWS SDK for PHP	79
Uploading Objects	79
Uploading Objects Using Multipart Upload	80
Multipart Upload Overview	80
Quick Facts	82
API Support for Multipart Upload	82
Multipart Upload API and Permissions	82
Using the AWS SDK for Java for Multipart Upload	84
Using the High-Level Java API for Multipart Upload	84
Upload a File	84
Abort Multipart Uploads	85
Track Multipart Upload Progress	86
Using the Low-Level Java API for Multipart Upload	89
Upload a File	89
List Multipart Uploads	92
Abort a Multipart Upload	93
Using the AWS SDK for .NET for Multipart Upload	94

Using the High-Level .NET API for Multipart Upload	94
Upload a File	94
Upload a Directory	96
Abort Multipart Uploads	99
Track Multipart Upload Progress	100
Using the Low-Level .NET API for Multipart Upload	103
Upload a File	103
List Multipart Uploads	107
Track Multipart Upload Progress	108
Abort a Multipart Upload	108
Using the AWS SDK for PHP for Multipart Upload	109
Using the High-Level PHP API for Multipart Upload	109
Upload a File	109
Using the Low-Level PHP API for Multipart Upload	110
Upload a File	111
List Multipart Uploads	114
Abort a Multipart Upload	114
Using the REST API for Multipart Upload	115
Uploading Objects in a Single Operation	115
Upload an Object Using the AWS SDK for Java	115
Upload an Object Using the AWS SDK for .NET	116
Upload an Object Using the REST API	120
Getting Objects	120
Retrieve an Object Using the AWS SDK for Java	120
Retrieve an Object Using the AWS SDK for .NET	123
Retrieve an Object Using REST API	126
Copying Objects	127
Copy an Object Using the AWS SDK for Java	127
Copy an Object Using the AWS SDK for .NET	128
Copy an Object Using the REST API	131
Listing Object Keys	132
Iterating Through Multi-Page Results	133
Listing Keys Hierarchically Using Prefix and Delimiter	133
Listing Keys Using the AWS SDK for Java	134
Listing Keys Using the AWS SDK for .NET	136
Access Control	141
Using Bucket Policies	141
Writing Bucket Policies	142
Setting Bucket Policies on a Bucket	142
Returning the Bucket Policies on a Bucket	143
Deleting Bucket Policies on a Bucket	143
Example Cases for Amazon S3 Bucket Policies	143
How to Use Resources, Principals, Operations, and Conditions in Bucket Policies	147
Using ACLs	152
Using ACLs and Bucket Policies Together	154
Using Query String Authentication	155
Data Protection	157
Using Reduced Redundancy Storage	157
Setting the Storage Class of an Object You Upload	158
Changing the Storage Class of an Object in Amazon S3	158
Return Code For Lost Data	159
Using Versioning	159
Enabling a Bucket's Versioning State	160
MFA Delete	160
Enabling Versioning	161
Configuring a Bucket with MFA Delete	162
Suspending Versioning	162
Determining the Versioning State	163

Adding Objects to Versioning-Enabled Buckets	164
Listing the Objects in a Versioning-Enabled Bucket	165
Retrieving a Subset of Objects in Buckets	165
Retrieving All Versions of a Key	165
Retrieving Additional Object Versions after Exceeding Max-Keys	166
Retrieving Object Versions	166
Retrieving the Metadata of an Object Version	168
Deleting Object Versions	168
Using MFA Delete	170
Working With Delete Markers	170
Removing Delete Markers	171
Restoring Previous Versions	173
Versioned Object Permissions and ACLs	174
Working with Versioning-Suspended Buckets	175
Adding Objects to Versioning-Suspended Buckets	176
Retrieving Objects from Versioning-Suspended Buckets	177
Deleting Objects from Versioning-Suspended Buckets	178
Setting Up Notification of Bucket Events	180
Request Routing	184
Request Redirection and the REST API	184
DNS Considerations	187
Performance Optimization	188
TCP Window Scaling	188
TCP Selective Acknowledgement	188
Using BitTorrent with Amazon S3	189
How You are Charged for BitTorrent Delivery	189
Using BitTorrent to Retrieve Objects Stored in Amazon S3	190
Publishing Content Using Amazon S3 and BitTorrent	191
Using Amazon DevPay with Amazon S3	192
Amazon S3 Customer Data Isolation	192
Amazon DevPay Token Mechanism	193
Amazon S3 and Amazon DevPay Authentication	193
Amazon S3 Bucket Limitation	194
Amazon S3 and Amazon DevPay Process	194
Additional Information	195
Handling Errors	196
The REST Error Response	196
Error Response	197
Error Code	197
Error Message	197
Further Details	197
The SOAP Error Response	197
Amazon S3 Error Best Practices	198
Server Access Logging	200
Server Access Logging Configuration API	201
Delivery of Server Access Logs	203
Server Access Log Format	204
Setting Up Server Access Logging	208
Appendix: The Access Policy Language	213
Overview	213
Key Concepts	214
Architectural Overview	216
Using the Access Policy Language	218
Evaluation Logic	219
How to Write a Policy	222
Basic Policy Structure	223
Element Descriptions	223
Supported Data Types	232

Special Information for Amazon S3 Policies	233
Document Conventions	236
Glossary	234
Index	239

Welcome to Amazon S3

Topics

- [Who Should Read this Guide \(p. 1\)](#)
- [How to Give Us Feedback \(p. 2\)](#)
- [How This Guide Is Organized \(p. 2\)](#)
- [Amazon S3 Resources \(p. 2\)](#)

Amazon S3 is a web service that enables you to store data in the cloud. You can then download the data or use the data with other AWS services, such as Amazon Elastic Cloud Computer (EC2).

This section describes who should read this guide, how the guide is organized, and other resources related to Amazon S3.

We hope you find the service to be easy-to-use, reliable, and inexpensive. If you want to provide feedback to the Amazon S3 development team, please post a message to the [Amazon S3 Discussion Forum](#) or the **Feedback** link at the top of every page in the HTML version of this guide.

Who Should Read this Guide

This guide has the following audiences:

- Developers creating libraries to implement the Amazon S3 API

This audience can use the programming guide to understand the concepts and functionality of Amazon S3 and the API reference to learn how the HTTP packets should look for particular operations.

- Developers using libraries created to implement the Amazon S3 API

This audience should not bother reading the API reference but, instead, focus on the concepts and functionality of Amazon S3 so that you can better understand the third-party libraries written for Amazon S3. These developers typically build applications that store and retrieve data across the Internet.

Required Knowledge and Skills

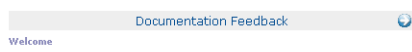
Use of this guide assumes you are familiar with the following:

- XML (go to [W3 Schools XML Tutorial](#))
- Basic understanding of web services (go to [W3 Schools Web Services Tutorial](#))
- A programming language for consuming a web service and any related tools

You should also have read the *Amazon S3 Getting Started Guide*. For more information, go to [Amazon S3 Getting Started Guide](#).

How to Give Us Feedback

The online version of this guide provides a link at the top of each page that enables you to enter feedback about this guide. We strive to make our guides as complete, error free, and easy to read as possible. You can help by giving us feedback. Thank you in advance!



How This Guide Is Organized

This guide is organized into several major sections described in the following table.

Information	Relevant Sections
General information about Amazon S3	Introduction to Amazon S3 (p. 6)
Conceptual information about Amazon S3	Introduction to Amazon S3 (p. 6)
Information about making requests	Making Requests (p. 21)
Information about how to use the Amazon S3 REST operations	Common REST API Elements (p. 26)
Information about how to use the Amazon S3 SOAP operations	Common SOAP API Elements (p. 62)
Information about using DevPay with Amazon S3	Amazon DevPay Developer Guide
Information about handling errors	Handling Errors (p. 196)
Information about BitTorrent	Using BitTorrent with Amazon S3 (p. 189)
Typographic and symbol conventions	Document Conventions (p. 236)

Each section is written to stand on its own, so you should be able to look up the information you need and go back to work. However, you can also read through the major sections sequentially to get in-depth knowledge about the Amazon S3.

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 S3 Getting Started Guide	The Getting Started Guide provides a quick tutorial of the service based on a simple use case. Examples and instructions for Java, Perl, PHP, C#, Python, and Ruby are included.
Amazon S3 API Reference	The API Reference describes Amazon S3 operations in detail.
Amazon S3 Technical FAQ	The FAQ covers the top 20 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.
Discussion Forums	A community-based forum for developers to discuss technical questions related to Amazon Web Services.
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 Calculator	Use the AWS calculator to estimate your monthly charges for using AWS services.
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.
Conditions of Use	Detailed information about the copyright and trademark usage at Amazon.com and other topics.

What's New

This What's New is associated with the 2006-03-01 release of Amazon S3. This guide was last updated on December 09, 2010.

The following table describes the important changes since the last release of the *Amazon S3 Developer Guide*.

Change	Description	Date
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 (p. 80) .	9 December 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 (p. 80) .	10 November 2010
Canonical ID support in bucket policies	You can now specify canonical IDs in bucket policies. For more information, see Specifying Principals in Bucket Policies (p. 147) and for a sample, see Granting Permission, Using Canonical ID, to a CloudFront Origin Identify (p. 146)	17 September 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 Using AWS Identity and Access Management.	2 September 2010
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 (SNS) topic when Amazon S3 detects a key event on a bucket. For more information, see Setting Up Notification of Bucket Events (p. 180) .	14 July 2010

Change	Description	Date
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 Bucket Policies (p. 11) .	6 July 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. 40) .	9 June 2010
New endpoint for EU (Ireland)	Amazon S3 now provides an endpoint for EU (Ireland): <code>http://s3-eu-west-1.amazonaws.com</code> .	9 June 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 S3 Console User Guide</i> .	9 June 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. 10) .	12 May 2010
New Region supported	Amazon S3 now supports the Asia Pacific (Singapore) Region. For more information, see Buckets and Regions (p. 67) .	28 April 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. 15) and Using Versioning (p. 159) .	8 February 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. 67) .	2 December 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 AWS Library Support (p. 21) , or go to Working With Amazon S3 in the <i>Amazon Simple Storage Service Getting Started Guide</i> .	11 November 2009

Introduction to Amazon S3

Topics

- [Overview of Amazon S3 \(p. 6\)](#)
- [Advantages to Amazon S3 \(p. 6\)](#)
- [Amazon S3 Concepts \(p. 7\)](#)
- [Features \(p. 10\)](#)
- [Amazon S3 Application Programming Interfaces \(API\) \(p. 19\)](#)
- [Paying for Amazon S3 \(p. 19\)](#)
- [Related Amazon Web Services \(p. 20\)](#)

This introduction to Amazon S3 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.

Overview of Amazon S3

Amazon S3 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 can be used 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 to developers.

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

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 to ensure that data is kept secure from unauthorized access.
- **Standard interfaces**—Use standards-based REST and SOAP interfaces designed to work with any Internet-development toolkit.

Amazon S3 Concepts

Topics

- [Buckets \(p. 7\)](#)
- [Objects \(p. 7\)](#)
- [Keys \(p. 7\)](#)
- [Regions \(p. 8\)](#)
- [Amazon S3 Data Consistency Model \(p. 8\)](#)

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 like 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 `johnsmith` bucket, then it is addressable using the URL `http://johnsmith.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 \(p. 67\)](#). 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 \(p. 159\)](#).

For more information about buckets, see [Working with Amazon S3 Buckets \(p. 65\)](#).

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. The developer 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 \(p. 7\)](#) and [Versioning \(p. 159\)](#).

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
This is the default Region. The US Standard Region automatically routes requests to facilities in Northern Virginia or the Pacific Northwest using network maps. To use this region, select US - Standard as the region when creating a bucket in the console. The US Standard Region provides eventual consistency for all requests.
- **US(Northern California)**—Uses Amazon S3 servers in Northern California
To use this Region, choose US - N. California as the Region when creating the bucket in the AWS Management Console.
In Amazon S3, the US Northern California Region 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)**—Uses Amazon S3 servers in Ireland
To use this Region, choose EU - Ireland as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the EU (Ireland) Region provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- **APAC (Singapore)**—Uses Amazon S3 servers in Singapore.
To use this Region, choose APAC - Singapore as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the APAC (Singapore) Region 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. After a "success" is returned, your data is safely stored. However, information about the changes might not immediately replicate across Amazon S3 and 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. The EU (Ireland) and Northern California 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 puts are simultaneously made to the same key, the put 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.

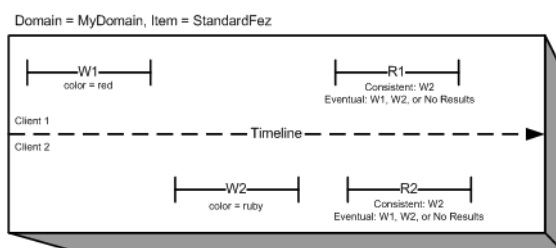
The following table describes the characteristics of eventually consistent read and consistent read.

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

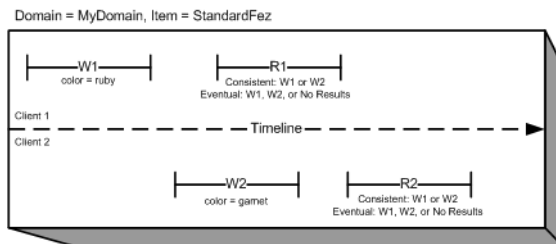
This section provides examples of eventually consistent and consistent read requests when multiple clients are writing to the same items.

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.

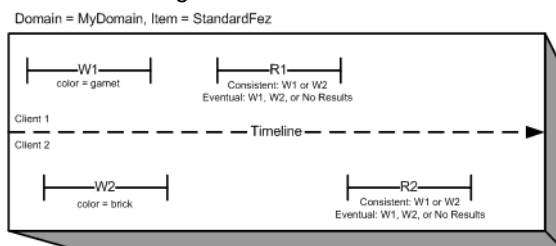


In the next example, W2 does not complete before the start of R1. Therefore, R1 might return `color = ruby` or `color = garnet` for either a consistent read or an eventually consistent read. Also, depending on the amount of time that has elapsed, an eventually consistent read might return no results.

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.



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.



Features

Topics

- [Reduced Redundancy Storage \(p. 10\)](#)
- [Bucket Policies \(p. 11\)](#)
- [AWS Identity and Access Management \(p. 11\)](#)
- [Access Control Lists \(p. 12\)](#)
- [Versioning \(p. 15\)](#)
- [Operations \(p. 18\)](#)

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's 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.

We charge 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 \(p. 157\)](#).

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 ACLs (described below) that can only add (grant) permissions 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 \(p. 141\)](#).

AWS Identity and Access Management

Amazon S3 integrates with AWS Identity and Access Management (IAM), a service that lets your organization do the following:

- Create users and groups under your organization's AWS Account
- Easily share your AWS Account resources between the users in the account
- Assign unique security credentials to each user

- Granularly control users access to services and resources
- Get a single AWS bill for all users in the AWS Account

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 general information about IAM, go to:

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

For specific information about how you can control User access to Amazon S3, go to [Integrating with Other AWS Products](#) in *Using AWS Identity and Access Management*.

Access Control Lists

Topics

- [Grantees \(p. 12\)](#)
- [Permissions \(p. 15\)](#)
- [ACLs and Bucket Policy Differences \(p. 15\)](#)

Each bucket and object in Amazon S3 has an ACL that defines its access control policy. When a request is made, Amazon S3 authenticates the request using its standard authentication procedure and then checks the ACL to verify sender was granted access to the bucket or object. If the sender is approved, the request proceeds. Otherwise, Amazon S3 returns an error.



Note

Bucket and object ACLs are completely independent; an object does not inherit the ACL from its bucket. For example, if you create a bucket and grant write access to another user, you will not be able to access the user's objects unless the user explicitly grants access. This also applies if you grant anonymous write access to a bucket. Only the user "anonymous" will be able to access objects the user created unless permission is explicitly granted to the bucket owner.



Important

We highly recommend that you do not grant the anonymous group write access to your buckets as you will have no control over the objects others can store and their associated charges. For more information, see [Grantees \(p. 12\)](#) and [Permissions \(p. 15\)](#)

Grantees

Following are five types of grantees that can access a bucket or object within Amazon S3.

- Owner
- User by E-mail
- User by Canonical Representation
- AWS User Group
- Anonymous Group

When granting access to a specific AWS user, the user must have an Amazon account and must be signed up for AWS and Amazon S3. This enables a user to access any allowed buckets or objects using his or her AWS Access Key ID and Secret Access Key. When you grant access to all AWS users, any AWS user is able to access allowed buckets or objects using an AWS Access Key ID and Secret Access Key. When you grant anonymous access, any user is able to access allowed buckets or objects by omitting the AWS Access Key ID and Signature from a request.

Any user that is granted access to an object or bucket can construct an HTTP URL that can be used to access that object or bucket through the query string authentication mechanism. This HTTP URL can be distributed to any user with a web client or embedded in a web page.



Note

All HTTP queries have an expiration parameter that allows you to set how long the query will be valid. For example, you can configure a web page graphic to expire after a very long period of time or a software download to only last for 24 hours.

Owner

Every bucket and object in Amazon S3 has an owner, the user that created the bucket or object. The owner of a bucket or object cannot be changed. However, if the object is overwritten by another user (deleted and rewritten), the new object will have a new owner.



Note

Even the owner is subject to the ACL. For example, if an owner does not have READ access to an object, the owner cannot read that object. However, the owner of an object always has write access to the access control policy (WRITE_ACP) and can change the ACL to read the object.

User by E-mail

You can grant access to buckets and objects within your Amazon S3 account to anyone with an Amazon Web Services account. Any users that you grant access will be able to access buckets and objects using their AWS Access Key IDs and Secret Access Keys.

Following is an example that shows the XML format for granting access to a user through an Amazon customer e-mail address.

```
<Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="AmazonCustomerByEmail">
  <EmailAddress>chriscustomer@email.com</EmailAddress>
</Grantee>
```

E-mail grants are internally converted to the CanonicalUser representation when you create the ACL. If the grantee changes his or her e-mail address, it will not affect the existing Amazon S3 permissions.

Adding a grantee by e-mail address only works if exactly one Amazon account corresponds to the specified e-mail address. If multiple Amazon accounts are associated with the e-mail address, an AmbiguousGrantByEmail error message is returned. This is rare but usually occurs if a user created an Amazon account in the past, forgot the password, and created another Amazon account using the same e-mail address. If this occurs, the user should contact Amazon.com customer service to have the accounts merged or you should grant user access specifying the CanonicalUser representation.

User by Canonical Representation

You can grant access to buckets and objects within your Amazon S3 account to anyone with an Amazon Web Services account. Any users that you grant access will be able to access buckets and objects using their AWS Access Key IDs and Secret Access Keys.



Note

To locate the CanonicalUser ID for a user, the user must perform the ListAllMyBuckets operation in his or her Amazon S3 account and copy the ID from the Owner XML object.

Following is an example that shows the XML format for granting access to a user through an Amazon customer CanonicalUser ID.

```
<Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="CanonicalUser">
  <ID>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666e9</ID>

  <DisplayName>chriscustomer</DisplayName>
</Grantee>
```

The ID string specifies the CanonicalUser ID and must exactly match the ID of the user that you are adding. The DisplayName element is read-only. If you specify a DisplayName, it will be ignored and replaced with the name stored by Amazon. We recommend that you match your DisplayName to your Forum name.

AWS User Group

You can grant access to buckets or objects to anyone with an Amazon AWS account. Although this inherently insecure as any AWS user who is aware of the bucket or object will be able to access it, you might find this authentication method useful.

All AWS users can be specified as a grantee using the following example XML representation.

```
<Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">

  <URI>http://acs.amazonaws.com/groups/global/AuthenticatedUsers<URI>
</Grantee>
```

AllUsers Group

You can grant anonymous access to any Amazon S3 object or bucket. Any user will be able to access the object by omitting the AWS Key ID and Signature from a request.

AllUsers can be specified as a grantee using the following example XML representation:

```
<Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">

  <<URI>http://acs.amazonaws.com/groups/global/AllUsers<URI>
</Grantee>
```

Permissions

The permission in a grant describes the type of access to be granted to the respective grantee. Following are permissions that are supported by Amazon S3.

Elements

- **READ**—when applied to a bucket, grants permission to list the bucket.
When applied to an object, this grants permission to read the object data and/or metadata.
- **WRITE**—when applied to a bucket, grants permission to create, overwrite, and delete any object in the bucket.
This permission is not supported for objects.
- **READ_ACP**—grants permission to read the ACL for the applicable bucket or object.
The owner of a bucket or object always has this permission implicitly.
- **WRITE_ACP**—gives permission to overwrite the ACP for the applicable bucket or object.
The owner of a bucket or object always has this permission implicitly.
Granting this permission is equivalent to granting FULL_CONTROL because the grant recipient can make any changes to the ACP.
- **FULL_CONTROL**—provides READ, WRITE, READ_ACP, and WRITE_ACP permissions.
It does not convey additional rights and is provided only for convenience.

ACLs and Bucket Policy Differences

The following table describes how ACLs and bucket policies are different from one another.

Bucket Policy	ACL
Policies provide you the ability to <i>grant</i> or <i>deny</i> an account access to one or many Amazon S3 resources.	ACLs only provide the ability to <i>grant</i> a user access to a single resource.
Policies can restrict access based on customized conditions. For example, using policies, companies can restrict employee access to data for requests made outside of the company's firewall.	ACLs only grant access, with no conditions, to the grantee for the specific resource. With ACLs, a company can grant an employee access to a resource, but cannot require that the access occurs within the company's firewall.
Policies are able to restrict access based off of request attributes, such as IP address and HTTP referrer	No equivalent.

In general, customers should use bucket policies when they need to apply consistent rules across objects. ACLs work better for customers with small numbers of objects or inconsistent permissions across their objects where maintaining policies becomes overly burdensome.

Versioning

An object consists of two components: a key and a version ID. The combination of key and version ID uniquely identifies an object in a bucket.



Objects in the same bucket that have the same key (but different version IDs) are called versions of one another.

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 enable versioning to recover from unintended overwrites and deletions or to archive objects so that you can retrieve previous versions of them.



Note

The SOAP API does not support Versioning.

Before you enable versioning, Amazon S3 behaves exactly as it had before versioning was introduced: only one key can exist in a bucket, a `GET` retrieves an object, a `PUT` stores it, a `DELETE` removes the object, and so forth. When you create a bucket, this is the default behavior. Objects stored in unversioned buckets have a version ID of `null`.

When you enable versioning on a bucket, the objects in it are unchanged: the version IDs (`null`), contents, and permissions remain the same.

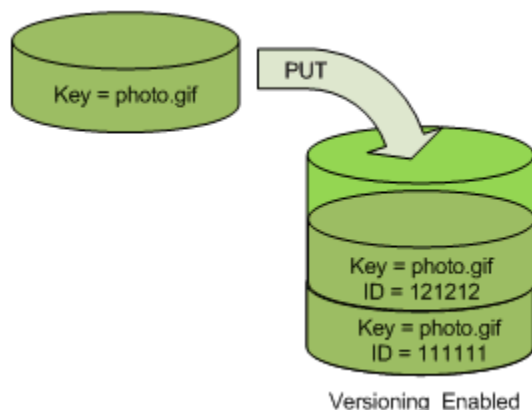
Enabling and suspending versioning is done at the bucket level. When you enable versioning for a bucket, all objects added to it will have a unique version ID. Unique version IDs are randomly generated, Unicode, UTF-8 encoded, URL-ready, opaque strings that are at most 1024 bytes long. An example version ID is `3/L4kqtJlcpXroDTDmJ+rmSpXd3dlbrHY+MTRCxf3vjVBH40Nr8X8gdRQBpUMLUo`. Only Amazon S3 generates version IDs. They cannot be edited.



Note

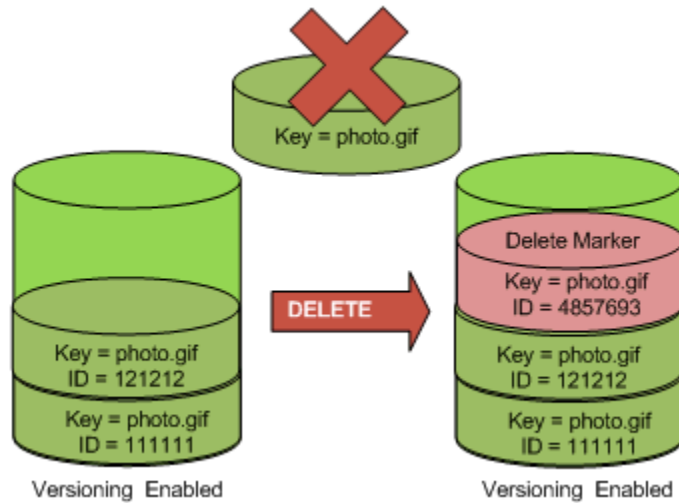
For simplicity, we will use much shorter IDs in all our examples.

When you `PUT` an object in a version-enabled bucket, the old 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.

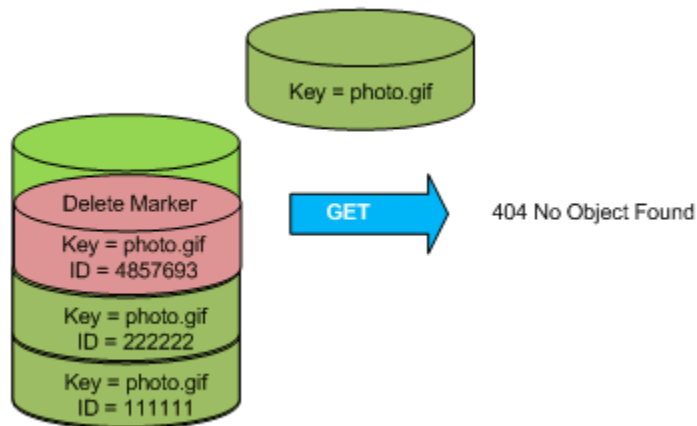


This functionality prevents you from accidentally overwriting or deleting objects and affords you the opportunity to retrieve a previous version of an object.

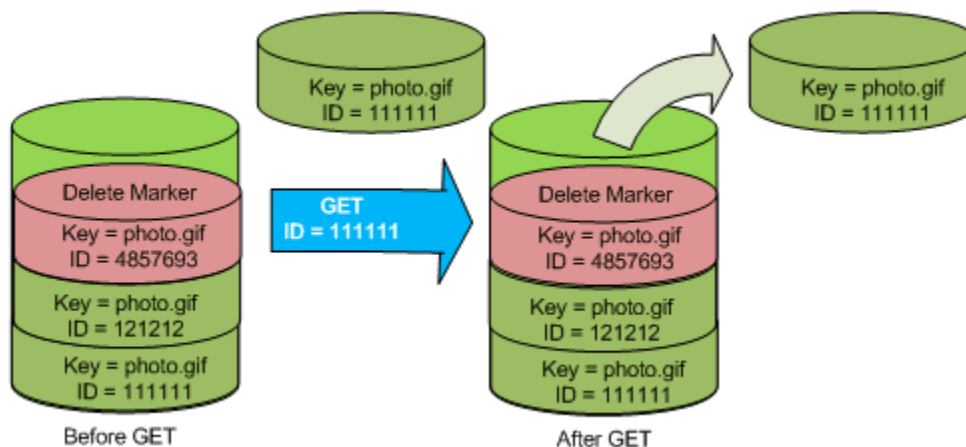
When you `DELETE` an object, all versions remain in the bucket and Amazon S3 inserts a delete marker, as shown in the following figure.



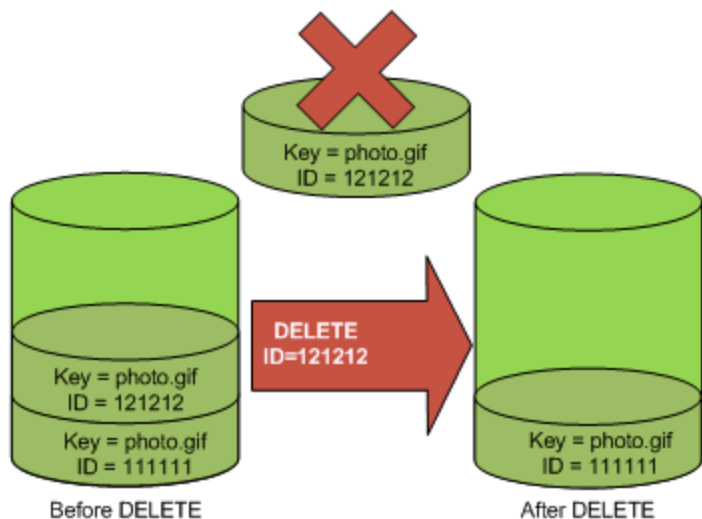
The delete marker becomes the latest version of the object. By default, `GET` requests retrieve the most recently stored version. Performing a simple `GET` `Object` request when the latest version is a delete marker returns a 404 Not Found error, as shown in the following figure.



You can, however, `GET` an older 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 latest 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 \(p. 160\)](#).

For more information, see [Using Versioning \(p. 159\)](#).

Operations

Amazon S3 offers APIs in REST and SOAP. 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 choose to 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.

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

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 [AWS Resource Center](#).

Related Amazon Web Services

Once we load your data into AWS you can use it with all AWS services. The following services are the ones you might use most frequently:

- **Amazon ElasticCompute Cloud**—This web service provides virtual compute resources in the cloud. For more information, go to [Amazon ElasticCompute Cloud](#).
- **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 Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3). For more information, go to [Amazon Elastic MapReduce](#).
- **Amazon Import/Export**—This service enables you to mail a storage device, such as a RAID drive, to Amazon so that we can upload your (terabytes) of data onto Amazon S3. For more information, go to [AWS Import/Export Developer Guide](#).

Making Requests

Topics

- [AWS Language Support \(p. 21\)](#)
- [Request Endpoints \(p. 21\)](#)
- [Authentication \(p. 22\)](#)
- [Using the REST API \(p. 25\)](#)
- [Using the SOAP API \(p. 61\)](#)

This section describes how to make requests using REST and SOAP.

AWS Language Support

AWS provides libraries, sample code, tutorials, and other resources for software developers who prefer to build applications using language-specific APIs instead of Amazon S3's SOAP and REST APIs. These libraries provide basic functions (not included in Amazon S3's SOAP and REST 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, go to:

- [Java](#)
- [PHP](#)
- [Ruby](#)
- [Windows and .NET](#)

For libraries and sample code in all languages, go to [Sample Code & Libraries](#).

Request Endpoints

An endpoint is a URL that is the entry point for a web service. Every web service request contains an endpoint. Amazon S3 REST requests use the following SSL secured or unsecured endpoints:

- <http://s3.amazonaws.com>
- <https://s3.amazonaws.com>

The Amazon S3 SOAP endpoints are the same, except that you append `/soap`, for example, `https://s3.amazonaws.com/soap`.

Region-Specific Endpoints

Amazon S3 also supports Region-specific endpoints. You can use a Region-specific endpoint to work on an object or bucket in a specific Region. The advantage of using a Region-specific endpoint is reducing the temporary latency you might experience immediately after creating a bucket in a Region. This temporary latency typically lasts less than one hour.

Amazon S3 supports the following Region-specific endpoints..

Region	Endpoint	Location Constraint
US-West (Northern California)	<code>s3-us-west-1.amazonaws.com</code>	<code>us-west-1</code>
Asia Pacific (Singapore)	<code>s3-ap-southeast-1.amazonaws.com</code>	<code>ap-southeast-1</code>
EU (Ireland)	<code>http://s3-eu-west-1.amazonaws.com</code>	<code>EU</code>

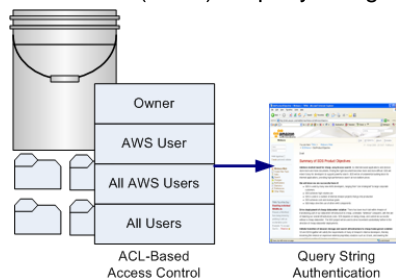
If you use a Region-specific endpoint to create a bucket, you *must* set the `LocationConstraint` bucket parameter to the same Region. For more information, see [Buckets and Regions \(p. 67\)](#).

Authentication

Authentication is the process of verifying the identity of a user or service trying to access an Amazon Web Services (AWS) product. Access Control defines who can access objects and buckets within Amazon S3 and the type of access (e.g., READ, WRITE, and so on). Authentication combined with access control prevents unauthorized users from accessing your data, modifying your data, deleting your data, or using your AWS account for services that cost you money.

Every interaction with Amazon S3 is authenticated or anonymous. When you sign up for an AWS account, you are provided with an AWS Access Key ID and a Secret Access Key. When you perform a request with Amazon S3, you assemble the request, perform a hash on the request using your Secret Access Key, attach the Signature (hash) to the request, and forward it to Amazon S3. Amazon S3 verifies the Signature is a valid hash of the request and, if authenticated, processes the request.

To allow selected users to access objects or buckets in your Amazon S3 account, you can use access control lists (ACLs) or query string authentication.



All requests the AWS must include a signature value. The signature value authenticates the request sender. This is important because some actions you take in Amazon S3 incur charges. Signing your request insures that you will only be charged for the actions you take. The signature value is, in part, generated from identifiers AWS assigns you when you create an AWS account.

When you create an AWS account, AWS assigns your AWS access key identifiers, a pair of related credentials:

- Access Key ID (a 20-character, alphanumeric string). For example: 022QF06E7MXBSH9DHM02
- Secret Access Key (a 40-character string). For example:
kWcrIUX5JEDGM/LtmEENI/aVmYvHNif5zB+d9+ct



Important

Your Secret Access Key is a secret and should be known only by you and AWS. It is important to keep it confidential to protect your account. Never include it in your requests to AWS and never e-mail it to anyone. Do not share it outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your Secret Access Key.

The Access Key ID uniquely identifies an AWS account. You include it in AWS service requests to identify yourself as the sender of the request.

To prove that you are the owner of the account making the request, you must include a signature. For all requests, you calculate the signature with your Secret Access Key. AWS uses the Access Key ID in the request to look up your Secret Access Key and then calculates a signature with the key. If the calculated signature matches the signature you sent, the request is considered authentic. Otherwise, the request fails authentication and is not processed.

Viewing Your Credentials

Your Access Key ID and Secret Access Key are displayed when you create your AWS account. They are not e-mailed to you. If you need to see them again, you can view them at any time from your AWS account.

To view your AWS access identifiers

1. Go to the Amazon Web Services web site at <http://aws.amazon.com>.
2. Point to **Your Web Services Account to display a list of options**.
3. Click **View Access Key Identifiers and log in to your AWS account**.

Your Access Key ID and Secret Access Key are displayed on the resulting AWS Access Identifiers page.

Using HMAC-SHA1 Signatures

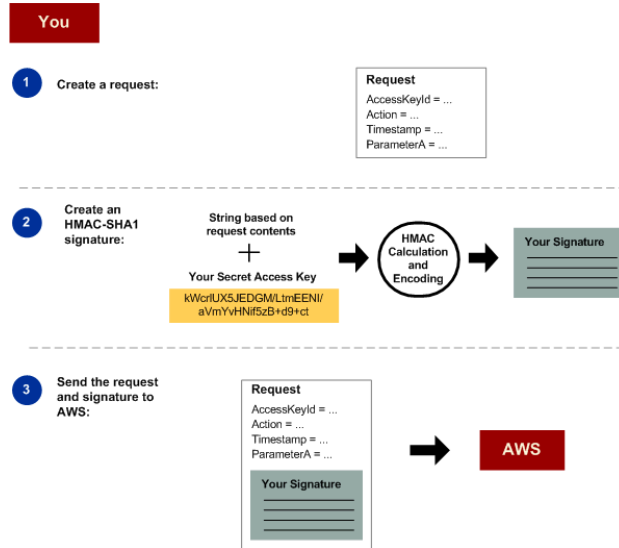
When accessing Amazon S3 using REST and SOAP, you must provide the following items so the request can be authenticated:

Request Elements

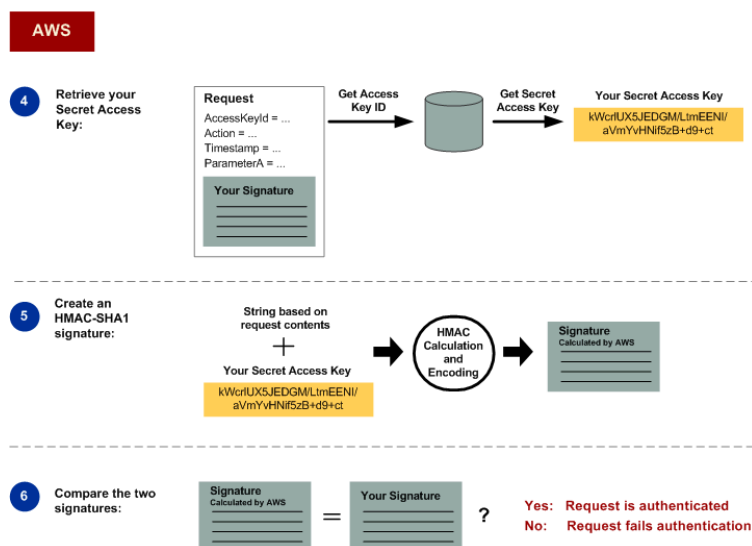
- **AWS Access Key Id**—Your AWS account is identified by your Access Key ID, which AWS uses to look up your Secret Access Key.
- **Signature**—Each request must contain a valid request signature, or the request is rejected. A request signature is calculated using your Secret Access Key, which is a shared 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.
The format of the value of this parameter is API-specific.

- **Date**—Each request must contain the time stamp of the request. Depending on the API 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 API to determine what the API requires.

Following are the general steps for authenticating requests to AWS. It is assumed you have already created an AWS account and received an Access Key ID and Secret Access Key.



1	Construct a request to AWS.
2	Calculate a keyed-hash message authentication code (HMAC) signature using your Secret Access Key.
3	Include the signature and your Access Key ID in the request, and then send the request to AWS. AWS performs the next three steps.



4	AWS uses the Access Key ID to look up your Secret Access Key.
5	AWS generates a signature from the request data and the Secret Access Key using the same algorithm you used to calculate the signature you sent in the request.
6	If the signature generated by AWS matches the one you sent in the request, the request is considered authentic. If the comparison fails, the request is discarded, and AWS returns an error response.

Detailed Authentication Information

For detailed information about REST and SOAP authentication, see [Authenticating REST Requests \(p. 27\)](#) and [Authenticating SOAP Requests \(p. 62\)](#).

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. Two characters, plus (+) and forward slash (/), cannot be used directly and must be encoded if used in a URI. For example, if the authentication code includes a plus (+) sign, encode it as %2B; in the request. Encode a forward slash as %2F;.

For examples of Base64 encoding, refer to the Amazon S3 code samples.

Using the REST API

Topics

- [Common REST API Elements \(p. 26\)](#)
- [Authenticating REST Requests \(p. 27\)](#)
- [REST Access Control Policy \(p. 38\)](#)
- [Virtual Hosting of Buckets \(p. 40\)](#)
- [Request Redirection and the REST API \(p. 44\)](#)
- [Browser-Based Uploads Using POST \(p. 46\)](#)

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. For more information, see [Working with Amazon S3 Buckets \(p. 65\)](#)

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 OPN5J17HBGZHT7JJ3X82: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 OPN5J17HBGZHT7JJ3X82: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 non US Standard bucket with path-style syntax using:

- `http://s3.amazonaws.com`
- A different Region endpoint than where the bucket resides, for example, `http://s3-euwest-1.amazonaws.com` and the bucket was created with the location constraint of Northern-California

Common REST API Elements

Amazon S3 REST operations are HTTP requests, as defined by RFC 2616 (<http://www.ietf.org/rfc/rfc2616.txt>). This section describes how Amazon S3 uses HTTP and the parts of HTTP requests and responses that Amazon S3 REST operations have in common. Detailed descriptions of individual operations are provided later in this guide.

A typical REST operation consists of a sending a single HTTP request to Amazon S3, followed by waiting for an HTTP response. Like any HTTP request, a request to Amazon S3 contains a request method, a URI, request headers, and sometimes a query string and request body. The response contains a status code, response headers, and sometimes a response body.

Following is an example that shows how to get an object named "Nelson" from the "quotes" bucket.

Sample Request

```
GET /Nelson HTTP/1.1
Host: quotes.s3.amazonaws.com
Date: Wed, 25 Nov 2009 12:00:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
```


Sample Response

```
HTTP/1.1 200 OK
x-amz-id-2: qBmKRcEWBBhH6XAqsKU/eg24V3jf/kWKN9dJip1L/FpbYr9FDy7wWfUrfdQOEMcY
x-amz-request-id: F2A8CCCA26B4B26D
Date: Wed, 25 Nov 2009 12:00:00 GMT
Last-Modified: Sun, 1 Jan 2009 12:00:00 GMT
ETag: "828ef3fdfa96f00ad9f27c383fc9ac7f"
Content-Type: text/plain
Content-Length: 5
Connection: close
Server: AmazonS3
```


ha-ha

Common Request Headers

Amazon S3 REST requests include headers which contain basic information about the request. Following is a table that describes common headers for Amazon S3 REST requests.

Header Name	Description	Required
Content-Length	Length of the message (without the headers) according to RFC 2616. Condition: Required for PUTs and operations that load XML, such as logging and ACLs.	Conditional
Content-Type	The content type of the resource. Example: <code>text/plain</code>	No
Date	The current date and time according to the requester. Example: <code>Wed, 25 Nov 2009 12:00:00 GMT</code>	Yes
Host	Normally, the value of Host is <code>s3.amazonaws.com</code> . A Host header with a value other than <code>s3.amazonaws.com</code> selects the bucket for the request as described in Virtual Hosting of Buckets (p. 40) . Condition: Required for HTTP 1.1 (most toolkits add this header automatically); optional for HTTP/1.0 requests.	Conditional
Authorization	The information required for request authentication. For more information, see The Authentication Header (p. 28) for details about the format.	Yes
x-amz-security-token	The security tokens for operations that use Amazon DevPay. Each request that uses Amazon DevPay requires two <code>x-amz-security-token</code> headers: one for the product token and one for the user token. Condition: Required for requests that use Amazon DevPay.  Note When Amazon S3 receives an authenticated request, it compares the computed signature with the provided signature. Improperly formatted multi-value headers used to calculate a signature can cause authentication issues. To ensure the signature is calculated properly, follow the instructions in the Constructing the CanonicalizedResource Element (p. 29) section.	Conditional

Authenticating REST Requests

Topics

- [The Authentication Header \(p. 28\)](#)

- [Request Canonicalization for Signing \(p. 29\)](#)
- [Constructing the CanonicalizedResource Element \(p. 29\)](#)
- [Constructing the CanonicalizedAmzHeaders Element \(p. 30\)](#)
- [Positional versus Named HTTP Header StringToSign Elements \(p. 31\)](#)
- [Time Stamp Requirement \(p. 31\)](#)
- [Authentication Examples \(p. 31\)](#)
- [REST Request Signing Problems \(p. 36\)](#)
- [Query String Request Authentication Alternative \(p. 36\)](#)

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 \(p. 46\)](#).

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, 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, then 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 0PN5J17HBGZHT7JJ3X82:frJIUN8DYpKDtOLCwo//y1lqDzg=
```

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 AWSSecretAccessKey:Signature
```

Developers are issued an AWS Access Key ID and AWS SecretAccess Key when they register. For request authentication, the *AWSAccessKeyId* element identifies the secret key that was used to compute the signature, and (indirectly) the developer making the request.

The *Signature* element is the RFC 2104HMAC-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, then the requester will have demonstrated possession to 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 pseudo-grammar that illustrates the construction of the *Authorization* request header (\n means the Unicode code point U+000A commonly called newline).

```
Authorization = "AWS" + " " + AWSAccessKeyId + ":" + Signature;

Signature = Base64( HMAC-SHA1( UTF-8-Encoding-Of( YourSecretAccessKeyID,
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> +
  [ sub-resource, if present. For example "?acl", "?location", "?logging", or
  "?torrent" ];

CanonicalizedAmzHeaders = <described below>
```

HMAC-SHA1 is an algorithm defined by RFC 2104 (go to [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 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 the empty string (""). |
|---|-----------------------------------|

2	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 on virtual hosted-style requests, see Virtual Hosting of Buckets (p. 40) .
3	Append the path part of the un-decoded HTTP Request-URI, up-to but not including the query string.
4	<p>If the request addresses a sub-resource, like ?versioning, ?location, ?acl, or ?torrent, or ?versionid append the sub-resource, its value if it has one, and the question mark. Note that in case of multiple sub-resources, sub-resources must be lexicographically sorted by sub-resource name and separated by '&'. e.g. ?acl&versionId=<i>value</i>.</p> <p>The list of sub-resources that must be included when constructing the CanonicalizedResource Element are: acl, location, logging, notification, partNumber, policy, requestPayment, torrent, uploadId, uploads, versionId, versioning, and versions.</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 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 \(p. 40\)](#).

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 lower-case. 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 white-space 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 white-space (including new-line) by a single space.
5	Trim any white-space 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 new-line (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, (Content-Type or Content-MD5, for example, are optional for PUT requests, and meaningless for GET requests), substitute the empty string ("") in 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 time-stamp 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 status 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.

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 time-stamp for the request 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	0PN5J17HBGZHT7JJ3X82
AWSSecretAccessKey	uV3F3YluFJax1cknvbcGwgjvx4QpvB+leU8dUj2o

In the example *StringToSigns*, formatting is not significant and \n means the Unicode code point U+000A commonly called newline.

Example Object GET

This example gets an object from the johnsmith bucket.

Request	StringToSign
GET /photos/puppy.jpg HTTP/1.1 Host: johnsmith.s3.amazonaws.com Date: Tue, 27 Mar 2007 19:36:42 +0000 <i>Authorization: AWS OPN5J17HBG ZHT7JJ3X82: xXjDGYUmKxnrwqr5KXNPGldn5LbA=</i>	GET\n\n\nTue, 27 Mar 2007 19:36:42 +0000\n/johnsmith/photos/puppy.jpg

Note that the CanonicalizedResource includes the bucket name, but the HTTP Request-URI does not (it is specified by the Host header)

Example Object PUT

This example puts an object into the johnsmith bucket.

Request	StringToSign
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 OPN5J17HBG ZHT7JJ3X82: hcicpDDvL9SsO6AkvxqmIWkmOuQ=</i>	PUT\n\n\nimage/jpeg\nTue, 27 Mar 2007 21:15:45 +0000\n/johnsmith/photos/puppy.jpg

Note the Content-Type header in the request and in the StringToSign. Also note that the Content-MD5 is left blank in the StringToSign since 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 Authorization: AWS OPN5J17HBGZHT7JJ3X82:js Rt/rhG+Vtp88HrYL706QhE4w4=</pre>	<pre>GET\n \n \n Tue, 27 Mar 2007 19:42:41 +0000\n \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 sub-resource 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 Authorization: AWS OPN5J17HBG ZHT7JJ3X82:thdUi9VAkzhkniLj96JIrOPGi0g=</pre>	<pre>GET\n \n \n Tue, 27 Mar 2007 19:44:46 +0000\n \n /johnsmith/?acl</pre>

Notice how the sub-resource 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
<code>DELETE /johnsmith/photos/puppy.jpg HTTP/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 0PN5J17HBG ZHT7JJ3X82:k3nL7gH3+PadhTEVn5Ip83xlyzk=</i></code>	<code>DELETE\n\n\n\nx-amz-date:Tue, 27 Mar 2007 21:20:26 +0000\n/johnsmith/photos/puppy.jpg</code>

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 field for the actual 'Date' header is left blank in the StringToSign.

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 Authorization: AWS OPN5J17HBGZHT7JJ3X82: COFlOtU8Ylb9KDTpZqYkZPX91iI= </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:\n joe@johnsmith.net,jane@johns\n mith.net\n /static.johnsmith.net/db-\n backup.dat.gz </pre>

Notice how the 'x-amz-' headers are sorted, white-space trimmed, converted to lowercase, and multiple headers with the same name have been joined using a comma 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 Authorization: AWS OPN5J17HBG ZHT7JJ3X82:Db+gepJSUbZKwpx1FR0DLtEYoZA= </pre>	<pre> GET\n \n \n Wed, 28 Mar 2007 01:29:59\n +0000\n / </pre>

Example Unicode Keys

Request	StringToSign
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 <i>Authorization: AWS 0PN5J17HBGZHT7JJ3X82: dxhSBHoI6eVSPcXJqEghlUzZMnY=</i>	GET\n \n \n Wed, 28 Mar 2007 01:49:49 +0000\n /diction ary/fran%C3%A7ais/pr%c3%a9f%c3%a8re



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 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.

Creating a Signature

Following is an example query string authenticated Amazon S3 REST request.

```
GET /photos/puppy.jpg
?AWSAccessKeyId=0PN5J17HBGZHT7JJ3X82&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>AWSSecretAccessKeyId</i>	0PN5J17HBGZHT7JJ3X82	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>	vjbyPxybdZaNmGa%2ByT272YEAiv4%3D	The URL encoding of the Base64 encoding of the HMAC-SHA1 of StringToSign.

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 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. Also note that in *StringToSign*, the HTTP *Date* positional element has been replaced with *Expires*. The *CanonicalizedAmzHeaders* and *CanonicalizedResource* are the same.

Example Query String Request Authentication

Request	StringToSign
GET /photos/puppy.jpg?AWSAccessKey Id=0PN5J17HBGZHT7JJ3X82& Signature=rucSbH0yNEcP9oM2XN louVI3BH4%3D& Expires=1175139620 HTTP/1.1 Host: johnsmith.s3.amazonaws.com	GET\n \n \n 1175139620\n /johnsmith/photos/puppy.jpg

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.

REST Access Control Policy

Topics

- [REST Access Policy Existing Bucket \(p. 38\)](#)
- [Canned Access Policies \(p. 39\)](#)

There are two ways to set the access control policy with REST. You can set the access control policy (ACP) for an existing bucket or object by requesting a PUT to `/bucket?acl` or `/bucket/key?acl`. Or, at the time you are writing a bucket or object you can include an x-amz-acl header with your PUT request that stores a canned ACP with the written resource.

REST Access Policy Existing Bucket

You can set the ACL on an existing bucket or object with an HTTP PUT to `/bucket?acl`, or `/bucket/key?acl`, where the body of the operation is the new ACL. To edit an existing ACL, fetch `/bucket?acl` or `/bucket/key?acl` to get the existing ACL, edit it locally, and then PUT the modified version back to `?acl`.

Following is an example that demonstrates how to set an existing object ACL so that only the owner has full access to the object.

First, we get the owner's canonical user grant information.

```
GET /Neo?acl HTTP/1.1
Host: quotes.s3.amazonaws.com
Date: Wed, 25 Nov 2009 12:00:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=

HTTP/1.1 200 OK

<AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>314133b66967d86f031c7249d1d9a80249109428335cd0ef1cdc487b4566cb1b</ID>
    <DisplayName>s3-nickname</DisplayName>
  </Owner>
```

```
<AccessControlList>
  <Grant>
    <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
      <ID>314133b66967d86f031c7249d1d9a80249109428335cd0ef1cdc487b4566cb1b</ID>

      <DisplayName>s3-nickname</DisplayName>
    </Grantee>
    <Permission>FULL_CONTROL</Permission>
  </Grant>
</AccessControlList>
</AccessControlPolicy>
```

Then, we set an existing object ACL so that only the owner has full access to the object.

```
PUT /Neo?acl HTTP/1.1
Host: quotes.s3.amazonaws.com
Content-Length: 214
Date: Wed, 25 Nov 2009 12:00:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=

<AccessControlPolicy>
  <Owner>
    <ID>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666e9</ID>
    <DisplayName>chriscustomer</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xsi:type="CanonicalUser">
        <ID>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666e9</ID>

        <DisplayName>chriscustomer</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

Canned Access Policies

Because of restrictions in what can be sent via http headers, Amazon S3 supports the concept of canned access policies for REST. A canned access policy can be included with the x-amz-acl header as part of a PUT operation to provide shorthand representation of a full access policy. When Amazon S3 sees the x-amz-acl header as part of a PUT operation, it will assign the respective access policy to the resource created as a result of the PUT. If no x-amz-acl header is included with a PUT request, then the bucket or object is written with the private access control policy (even if, in the case of an object, the object already exists with some other pre-existing access control policy).

Following are canned ACLs that are supported for REST.

- **private**—Owner gets FULL_CONTROL.
No one else has access rights (default).
- **public-read**—Owner gets FULL_CONTROL and the anonymous principal is granted READ access.
If this policy is used on an object, it can be read from a browser with no authentication.

- **public-read-write**—Owner gets `FULL_CONTROL`, the anonymous principal is granted `READ` and `WRITE` access.
This can be a useful policy to apply to a bucket, but is generally not recommended.
- **authenticated-read**—Owner gets `FULL_CONTROL`, and any principal authenticated as a registered Amazon S3 user is granted `READ` access.
- **bucket-owner-read**—Object Owner gets `FULL_CONTROL`, Bucket Owner gets `READ`
This ACL applies only to objects and is equivalent to private when used with PUT Bucket. You use this ACL to let someone other than the bucket owner write content (get full control) in the bucket but still grant the bucket owner read access to the objects.
- **bucket-owner-full-control**—Object Owner gets `FULL_CONTROL`, Bucket Owner gets `FULL_CONTROL`
This ACL applies only to objects and is equivalent to private when used with PUT Bucket. You use this ACL to let someone other than the bucket owner write content (get full control) in the bucket but still grant the bucket owner full rights over the objects.

Following is an example that shows how to write data to an object and makes the object readable by anonymous principals.

Sample Request

```
PUT /Neo HTTP/1.1
x-amz-acl: public-read
Content-Length: 4
Host: quotes.s3.amazonaws.com
Date: Wed, 25 Nov 2009 12:00:00 GMT
Content-Type: text/plain
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=

woah
```

Sample Response

```
HTTP/1.1 200 OK
x-amz-id-2: LriYPLdmOdAiIfgSm/F1YsViT1LW94/xUQxMsF7xiEbl0wiIOIx1+zbwZ163pt7
x-amz-request-id: 0A49CE4060975EAC
Date: Wed, 25 Nov 2009 12:00:00 GMT
ETag: "aba878a8"
Content-Length: 0
Connection: close
Server: AmazonS3
```

Virtual Hosting of Buckets

Topics

- [HTTP Host Header Bucket Specification \(p. 41\)](#)
- [Examples \(p. 41\)](#)
- [Customizing Amazon S3 URLs with CNAMEs \(p. 43\)](#)
- [Limitations \(p. 44\)](#)
- [Backward Compatibility \(p. 44\)](#)

Virtual Hosting, in general, 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 using the first slash delimited

component of the Request-URI path. Alternatively, using Amazon S3 Virtual Hosting, you can address a bucket in a REST API call 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 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 non US Standard bucket with path-style syntax using:

- `http://s3.amazonaws.com`
- A different Region endpoint than where the bucket resides, for example, `http://s3-euwest-1.amazonaws.com` and the bucket was created with the location constraint of Northern-California

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 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 example in the following table. Note that omitting the `Host` header is only legal for HTTP 1.0 requests.
- Otherwise, if the value of the `Host` header ends in `'s3.amazonaws.com'`, then 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 sub-domains of `s3.amazonaws.com`, and is illustrated by the third and fourth example in the following table.
- Otherwise, the bucket for the request will be the lower-cased 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 outside the scope of this document, but the result is illustrated by the final example in the following table.

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.

Following is the example URL.

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

Following is the example request.

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

Following is the example request with HTTP 1.0 omitting the host header.

```
GET /johnsmith/homepage.html  
HTTP/1.0  
Host: s3.amazonaws.com
```

For information about DNS compatible names, see [Limitations \(p. 44\)](#). For more information about keys, see [Keys \(p. 7\)](#).

Example Virtual Hosted Style Method

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

Following is the example URL.

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

Following is the example request.

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

Following is the example request with the incorrect case. Notice that sentence case is irrelevant. However, uppercase buckets are not accessible using this method.

```
GET /homepage.html HTTP/1.1  
Host: JohnSmith.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`.

Following is the example URL.

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

Following is the example request.

```
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 web site or service. For example, if you host your web site's images on Amazon S3, you might prefer `http://images.johnsmith.net/` as opposed to `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 may be referenced as follows:

`http://[BucketName].s3.amazonaws.com/[Filename]`, for example,
`http://images.johnsmith.net.s3.amazonaws.com/mydog.jpg`. Using CNAME you can map `images.johnsmith.net` to an Amazon S3 host name so 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) is `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 may result in extra HTTP redirects.



Note

Amazon S3 only sees the original host name and is unaware of the CNAME mapping used to resolve the request.

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 \(p. 21\)](#).

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

Your 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 exact procedure for configuring DNS depends on your DNS server or DNS provider and is beyond scope of this document.

Limitations

Because DNS names are case insensitive, only lower-case buckets are addressable using the virtual hosting method. For more information, see [Bucket Restrictions and Limitations \(p. 66\)](#).

Specifying the bucket for the request 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.

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 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 \(p. 44\)](#)
- [Redirects and 100-Continue \(p. 45\)](#)
- [Redirect Example \(p. 45\)](#)

This section describes how to handle HTTP redirects using REST. For general information about Amazon S3 redirects, see [Request Redirection and the REST API \(p. 184\)](#).

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 relying on a library to fulfill the redirect requirement, test the following cases:

Launch Process

1	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.
2	Verify non-GET redirects, such as PUT and DELETE, work correctly.
3	Verify large PUT requests follow redirects correctly.
4	Verify PUT requests follow redirects correctly if the 100-continue response takes a long time to arrive.

HTTP user-agents that strictly conform to RFC2616 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 only issue redirects to hosts within the amazonaws.com domain and the effect of the redirected request will be the same as 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
```

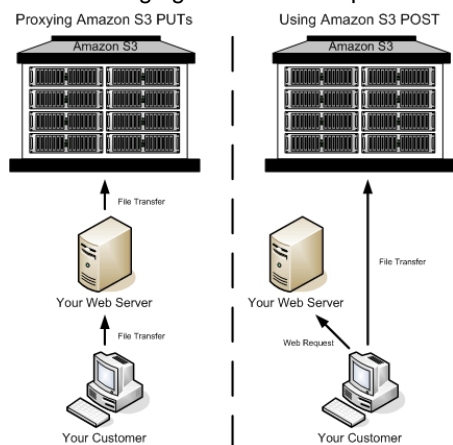
Browser-Based Uploads Using POST

Topics

- [HTML Forms \(p. 47\)](#)
- [Upload Examples \(p. 54\)](#)
- [POST with Adobe Flash \(p. 61\)](#)

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.

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

Topics

- [HTML Form Encoding \(p. 48\)](#)
- [HTML Form Declaration \(p. 48\)](#)
- [HTML Form Fields \(p. 49\)](#)
- [Policy Construction \(p. 51\)](#)
- [Constructing a Signature \(p. 54\)](#)
- [Redirection \(p. 54\)](#)

When communicating with Amazon S3, you normally use the REST or SOAP APIs to perform put, get, delete, and other operations. With POST, users upload data directly to Amazon S3 through their browsers, which do not understand SOAP APIs or how to make a REST `PUT` request.

To allow users to upload content to Amazon S3 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 make sure that it meets conditions that you specify.



Note

The form data and boundaries (excluding the contents of the file) cannot exceed 20K.

This section describes 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. For information about query string authentication, see [Using Query String Authentication \(p. 155\)](#).

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>
```

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 (go to [RFC 1867](#)) for both file uploads and text area uploads.

Example

This is a form declaration for the bucket "johnsmith".

```
<form action="http://johnsmith.s3.amazonaws.com/" method="post"
enctype="multipart/form-data">
```

HTML Form Fields



Following is a table that describes a list of fields that can be used within a 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 (e.g., "C:\Program Files\directory1\file.txt" will be interpreted as "file.txt"). If no file or filename is provided, the variable is replaced with an empty string.

Element Name	Description	Required
<i>AWSSecretAccessKeyId</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 is required if a policy document is included with the request.	Conditional
<i>acl</i>	Specifies an Amazon S3 access control list. If an invalid access control list is specified, an error is generated. For more information on ACLs, see Amazon S3 ACLs (p. 12) . 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, Content-Type, Content-Disposition, Content-Encoding, 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 <code>\${filename}</code> variable. For example, if the user Betty uploads the file the file lolcatz.jpg and you specify <code>/user/betty/\${filename}</code> , the file is stored as <code>/user/betty/lolcatz.jpg</code> . For more information, see Using Keys .	Yes
<i>policy</i>	Security Policy describing what is permitted in the request. Requests without a security policy are considered anonymous and only work on publicly writable buckets.	Yes

Element Name	Description	Required
<i>success_action_redirect, redirect</i>	<p>The URL to which the client is redirected upon successful upload.</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 acts as if the field is not present.</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. 54).</p> <p> Note</p> <p>The <i>redirect</i> field name is deprecated and support for the <i>redirect</i> 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>Accepts the values 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 on 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</p> <p>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
<i>signature</i>	<p>The HMAC signature constructed using the secret key of the provided <code>AWSAccessKeyId</code>.</p> <p>For more information, see Using Auth Access.</p>	Conditional
<i>x-amz-security-token</i>	<p>Amazon DevPay security token.</p> <p>Each request that uses Amazon DevPay requires two <i>x-amz-security-token</i> form fields: one for the product token and one for the user token.</p> <p>For more information, see Using DevPay.</p>	No

Element Name	Description	Required
Other field names prefixed with x-amz-meta-	User-specified metadata. Amazon S3 does not validate or use this data. For more information, see PUT Object .	No
file	File or text content. The file or content must be the last field in the form. You cannot upload more than one file at a time.	Yes

Policy Construction

Topics

- [Expiration \(p. 51\)](#)
- [Conditions \(p. 52\)](#)
- [Condition Matching \(p. 53\)](#)
- [Character Escaping \(p. 53\)](#)

The policy is a UTF-8 and Base64 encoded JSON document that specifies conditions which 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.

Following is an example of a policy document.

```
{ "expiration": "2007-12-01T12:00:00.000Z",  
  "conditions": [  
    { "acl": "public-read" },  
    { "bucket": "johnsmith" },  
    [ "starts-with", "$key", "user/eric/" ],  
  ]  
}
```

The policy document contains the expiration and conditions.

Expiration

The expiration specifies the expiration date of the policy in ISO8601 GMT date format. For example, "2007-12-01T12:00:00.000Z" specifies that the policy is not valid after 12:00 GMT on 2007-12-01.

Conditions

The conditions in the policy document are used to 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 prior to validating the policy. Therefore, all condition matching should be 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 \(p. 53\)](#).

Policy document conditions are described in the following table.

Element Name	Description
<code>acl</code>	Specifies conditions the ACL must meet. Supports exact matching and <i>starts-with</i> .
<code>content-length-range</code>	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> .
<code>key</code>	The name of the uploaded key. Supports exact matching and <i>starts-with</i> .
<code>success_action_redirect</code> , <code>redirect</code>	The URL to which the client is redirected upon successful upload. Supports exact matching and <i>starts-with</i> .
<code>success_action_status</code>	The status code returned to the client upon successful upload if <code>success_action_redirect</code> is not specified. Supports exact matching.
<code>x-amz-security-token</code>	Amazon DevPay security token. Each request that uses Amazon DevPay requires two <code>x-amz-security-token</code> 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>eW9ldHVizQ==</code> and the product token is <code>b0hnNVNKNWVJIQTA=</code> , you set the policy entry to: <code>{ "x-amz-security-token" : "eW9ldHVizQ==,b0hnNVNKNWVJIQTA=" }</code> . For more information about Amazon DevPay, see Using DevPay .
Other field names prefixed with <code>x-amz-meta-</code>	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

Following is a table that 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	Exact matches verify that fields match specific values. This example indicates that the ACL must be set to public-read:
	<pre>{ "acl": "public-read" }</pre>
	This example is an alternate way to indicate that the ACL must be set to public-read:
	<pre>["eq", "\$acl", "public-read"]</pre>
Starts With	If the value must start with a certain value, use starts-with. This example indicates that the key must start with user/betty/:
	<pre>["starts-with", "\$key", "user/betty/"]</pre>
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:
	<pre>["starts-with", "\$success_action_redirect", ""]</pre>
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:
	<pre>["content-length-range", 1048579, 10485760]</pre>


Character Escaping

Characters that must be escaped within a policy document are described in the following table.

Escape Sequence	Description
<code>\\</code>	Backslash
<code>\\$</code>	Dollar symbol

Escape Sequence	Description
\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 using UTF-8.
2 Encode those UTF-8 bytes using Base64.
3 Sign the policy with your Secret Access Key using HMAC SHA-1.
4 Encode the SHA-1 signature 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, 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 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

Topics

- [File Upload \(p. 55\)](#)
- [Text Area Upload \(p. 58\)](#)

File Upload

This example shows the complete process for constructing a policy and form to upload a file attachment.

Policy and Form Construction

Following is a policy that 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 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/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

Following is a Base64 encoded version of this policy.

```
eyJhZXBhXGhGdGlvbiI6IClYMDA3LDEyLTExVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXRpb25zI  
jogWwogICAgYyJidWNrZXQiOiAiam9obnNtaXR0In0sCiAgICBbInN0YXJ0cy13aXR0IiwgIiRrZXkiL  
CAidXNlci9lcmllLyJdLAogICAgYyJhY2wiOiAicHViYGljLXJlYWQifSwKICAgIH  
sic3YjY2Vzcl9hY3Rpb25fcmVkaXJlY3QiOiAiaHR0cDovL2pvaG5zbWl0aC5zMy5hb  
WF6b25hd3MuY29tL3NlY2Nlc3NmdWxfZXBsb2FkLmh0bWwifSwKICAgIFsic3RhcnczLXdpdGgiL  
CAiJENvbnRlbnQtVHlwZSI6ICJpbWFnZS8iXSwKICAgIHsieC1hbXotbWV0YS1ldWlkIjogIjE0MzY1MT  
IzNjUxMjc0In0sCiAgICBbInN0YXJ0cy13aXR0IiwgIiR4LWFteiltZXRhLXRhZyIsICl0IiwgQo  
gIF0KFQo=
```

The secret key ID is uV3F3YluFJaxlcknvbcGwgjvx4QpvB+leU8dUj2o, so
0RavWzkygo6QX9caELEqKi9kDbU= is the signature for the preceding Policy document.

Following is a form that supports a POST to the johnsmith.net bucket using 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="mul
tipart/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://johns
mith.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="15B4D3461F177624206A" />

      <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"

15B4D3461F177624206A
--9431149156168
Content-Disposition: form-data; name="Policy"

eyJAiZXhwaXJhdGlvbIi6ICIyMDA3LTEyLTAxVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXRpb25zI
jogWwogICAgeyJidWNrZXQiOiAiam9obnNtaXR0In0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiRrZXkiL
CAidXNlci9lcmljLyJdLAogICAgeyJhY2wiOiAicHVibGljLXJlYWQifSwKICAgIH
sic3VjY2Vzc19hY3Rpb25fcmlkX3Q1OjAiaHR0cDovL2pvaG5zbWl0aC5zMy5hb
WF6b25hd3MuY29tL3N1Y2Nlc3NmdWxmdXBsb2FkLmh0bWwifSwKICAgIFsic3RhcncRzLXdpdGgiL
CAiJENvbnRlbnQtVHlwZSIsICJpbWFnZS8iXSwKICAgIHsieClhbXotbWV0YS1ldWlkIjogIjE0MzY1MT
IzNjUxMjc0In0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiR4LWFteiltZXRhLXRhZyIsICJiXQo
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: 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/successful_upload.html?bucket=john
smith&key=user/eric/MyPicture.jpg&etag=&quot;39d459df
bc0faabbb5e179358dfb94c3&quot;;
Server: AmazonS3
```

Topics

- Policy and Form Construction (p. 58)
- Sample Request (p. 59)
- Sample Response (p. 60)

This example shows the complete process for constructing a policy and form to upload a text area. This is useful for submitting user-created content such as blog postings.

Policy and Form Construction

Following is a policy that 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.

eyJhZmVudGlvbiI6ICJyMDA3LTEyLTAxVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXRpb25zIjogWwogICAgeyJidWNrZXQoIiAiam9obnNtaXR0In0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiRrZXkiLCAidXNlci9lcm1jLyJdLAogICAgeyJhY2wiOiAichVibGljLXJlYWQifSwKICAgIHsic3VjY2Vzcl9hY3Rpb25fcmlVkaXJlY3QiOiAiaHR0cDovL2pvaG5zbWl0aC5zMy5hbWV6b25hd3MuY29tL25ld19wb3N0Lmh0bWwifSwKICAgIFsiZXEiLCAiJENvbnRlbnQtVHlwZSI6ICJ0ZXh0L2h0bWwiXSwKIICAgIHsic1h3otbWV0YS1ldWlkIjogIjE0MTIzNjUxMjc0In0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiR4LWVteiltlZXRhZyIsICIiXQogIF0KfQo=

The secret key ID is uV3F3YluFJax1cknvbcGwgjvx4QpvB+leU8dUj2o, so qA7FWXKq6VvU681I9KdveT1cWgE= is the signature for the preceding Policy document.

Following is a form that supports a POST to the johnsmith.net bucket using 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="mul
tipart/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://johns
mith.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="15B4D3461F177624206A" />

      <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"

15B4D3461F177624206A
--178521717625888
Content-Disposition: form-data; name="Policy"
eyJhZiZlZmVhZG9ldGVhbiI6IClYMDA3LTExYUxVDEYwOjAwOjAwLjAwMFoiLAogICJjb25kaXRpb25zI
jogWwogICAgeyJidWNrZXQzQm9obnNtaXR0In0sCiAgICBbInN0YXJ0cy13aXR0IiwgIiRrZXkiL
CAidXNlci9lcmllLyJdLAogICAgeyJhY2wiOiAicHVibGljLXJlYWQifSwKICAgIH
sic3VjY2Vzc19hY3Rpb25fcmVkaXJlY3QiOiAiaHR0cDovL2pvaG5zbWl0aC5zMy5hb
WF6b25hd3MuY29tL25ld19wb3N0Lmh0bWwifSwKICAgIFsiZXEiLCAiJENvbnRlbnQtVHlwZSIs
ICJ0ZXh0L2h0bWwifSwKICAgIHSieClhbXotbWV0YS1ldWlkIjogIjE0MTIzN
jUxMjc0In0sCiAgICBbInN0YXJ0cy13aXR0IiwgIiR4LWFteiltZXRhLXRhZyIsICIiXQogIF0KfQo=
--178521717625888
Content-Disposition: form-data; name="Signature"

qA7FWXKq6VvU68lI9KdveTlcWgE=
--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 public-readable `crossdomain.xml` file to the bucket that will accept `POST` uploads. 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 web site.

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 building Adobe Flash applications that upload to Amazon S3 using the `FileReference` API, 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. When set, Amazon S3 returns an XML document with a 201 status code. For information on the content of the XML document, see [POST Object](#). For information on form fields, go to [Form Fields](#).

Using the SOAP API

Topics

- [Common SOAP API Elements](#) (p. 62)
- [Authenticating SOAP Requests](#) (p. 62)

- [Setting Access Policy with SOAP \(p. 63\)](#)

This section contains information specific to the Amazon S3 SOAP API.

Common SOAP API Elements

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:

- *AWSSecretAccessKey*: 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 \(p. 21\)](#).

Authenticating SOAP Requests

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:

- *AWSSecretAccessKey*: Your AWS Access Key ID
- *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>1D9FVRAYCP1VJEXAMPLE=</AWSAccessKeyId>
  <Timestamp>2009-01-01T12:00:00.000Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
</CreateBucket>
```



Note

Authenticated SOAP requests must be sent to Amazon S3 over SSL. Only anonymous requests are allowed over non-SSL connections.



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.

For more information, see the sample [.NET SOAP libraries](#) for an example of how to do this.

Setting Access Policy with SOAP

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 [Access Control \(p. 141\)](#). 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>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666e9</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>1D9FVRAYCP1VJEXAMPLE=</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.

Working with Amazon S3 Buckets

Topics

- [Bucket Restrictions and Limitations \(p. 66\)](#)
- [Bucket Configuration Options \(p. 67\)](#)
- [Requester Pays Buckets \(p. 69\)](#)
- [Buckets and Access Control \(p. 72\)](#)
- [Billing and Reporting of Buckets \(p. 72\)](#)
- [Bucket Configuration Errors \(p. 72\)](#)

Every object stored in Amazon S3 is contained in a bucket. Buckets partition the namespace of objects stored in Amazon S3 at the top level. Within a bucket, you can use any names for your objects, but bucket names must be unique across all of Amazon S3.

Buckets are similar to Internet domain names. Just as Amazon is the only owner of the domain name Amazon.com, only one person or organization can own a bucket within Amazon S3. Once you create a uniquely named bucket in Amazon S3, you can organize and name the objects within the bucket in any way you like and the bucket will remain yours for as long as you like and as long as you have the Amazon S3 account.

The similarities between buckets and domain names is not a coincidence—there is a direct mapping between Amazon S3 buckets and subdomains of `s3.amazonaws.com`. Objects stored in Amazon S3 are addressable using the REST API under the domain `bucketname.s3.amazonaws.com`. For example, if the object `homepage.html` is stored in the Amazon S3 bucket `mybucket` its address would be `http://mybucket.s3.amazonaws.com/homepage.html`. For more information, see [Virtual Hosting of Buckets \(p. 40\)](#).

To determine whether a bucket name exists using REST, use `HEAD`, specify the name of the bucket, and set `max-keys` to 0. To determine whether a bucket name exists using SOAP, use `ListBucket` and set `MaxKeys` to 0. A `NoSuchBucket` response indicates that the bucket is available, a `AccessDenied` response indicates that someone else owns the bucket, and a `Success` response indicates that you own the bucket or have permission to access it.

Bucket Restrictions and Limitations

A bucket is owned by the AWS account (identified by AWS Access Key ID) 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, it can be deleted and its name can be reused.



Note

If you are using Amazon DevPay, each of your customers can have up to 100 buckets for each Amazon DevPay product they use. For more information, see [Using Amazon DevPay with Amazon S3 \(p. 192\)](#).

To comply with Amazon S3 requirements, bucket names:

- Can contain lowercase letters, numbers, periods (.), underscores (_), and dashes (-)
- Must start with a number or letter
- Must be between 3 and 255 characters long
- Must not be formatted as an IP address (e.g., 192.168.5.4)

To conform with DNS requirements, we recommend following these additional guidelines when creating buckets:

- Bucket names should not contain underscores (_)
- Bucket names should be between 3 and 63 characters long
- Bucket names should not end with a dash
- Bucket names cannot contain two, adjacent periods
- Bucket names cannot contain dashes next to periods (e.g., "my-.bucket.com" and "my.-bucket" are invalid)



Note

Buckets with names containing uppercase characters are not accessible using the virtual hosted-style request (e.g., `http://yourbucket.s3.amazonaws.com/yourobject`)

If you create a bucket using `<CreateBucketConfiguration>`, you must follow the DNS guidelines.

If you create a bucket using `<CreateBucketConfiguration>`, applications that access your bucket must be able to handle 307 redirects. For more information, see [Request Redirection and the REST API \(p. 184\)](#).

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.

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

Buckets cannot be nested, meaning buckets cannot be created within buckets.

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 make bucket create or delete calls 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. Additionally, make sure your application has logic to choose a different bucket name if a bucket name is already taken.

Bucket Configuration Options

When creating buckets, you can take advantage of additional Amazon S3 features by attaching the `<CreateBucketConfiguration>` XML body to a `PUT Bucket` request. Currently, you can select a location constraint. For more information, see [Buckets and Regions \(p. 67\)](#).

Buckets created with `<CreateBucketConfiguration>` are subject to additional restrictions:

- You cannot make a request to a bucket created with `<CreateBucketConfiguration>` using a path-style request; you must use the virtual hosted-style request. For more information, see [Virtual Hosting of Buckets \(p. 40\)](#).
- You must follow additional bucket naming restrictions. For more information, see [Bucket Restrictions and Limitations \(p. 66\)](#).

Buckets and 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. For example, if you reside in Europe, you will probably find it advantageous to create buckets in the EU (Ireland) Region.



Important

The SOAP API does not support geographical constraints.

Objects stored in a 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.

Amazon S3 supports the following Regions:

- **US Standard**—Uses Amazon S3 servers in the United States
This is the default Region. The US Standard Region automatically routes requests to facilities in Northern Virginia or the Pacific Northwest using network maps. To use this region, select US - Standard as the region when creating a bucket in the console. The US Standard Region provides eventual consistency for all requests.
- **US(Northern California)**—Uses Amazon S3 servers in Northern California
To use this Region, choose US - N. California as the Region when creating the bucket in the AWS Management Console.
In Amazon S3, the US Northern California Region 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)**—Uses Amazon S3 servers in Ireland
To use this Region, choose EU - Ireland as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the EU (Ireland) Region provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.
- **APAC (Singapore)**—Uses Amazon S3 servers in Singapore.

To use this Region, choose APAC - Singapore as the Region when creating the bucket in the AWS Management Console. In Amazon S3, the APAC (Singapore) Region provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and DELETES.

Regions are set at the bucket level. You specify a Region using the *LocationConstraint* bucket parameter. If you do not specify a Region, Amazon S3 hosts your buckets on servers in the US Standard Region.




Important

If you use a Region-specific endpoint to create a bucket, you *must* set the *LocationConstraint* bucket parameter to the same Region. For more information about Region-specific endpoints, see [Request Endpoints \(p. 21\)](#).

How to Specify a Bucket's Region

Use the following process to specify a bucket's Region.

Specifying a Bucket's Region

1	In a bucket creation request, set the <i>LocationConstraint</i> parameter to a specific Region, for example, <i>CreateBucketConfiguration.LocationConstraint=us-west-1</i>
2	<p>Optionally, when creating a bucket in the Northern California Region, you can also use a Region-specific endpoint in the request to create the bucket. For example, to match the Region specified in step 1, you would use the endpoint, https://s3-us-west-1.amazonaws.com (or http://s3-us-west-1.amazonaws.com).</p> <p>Using the Region-specific endpoint avoids the latency caused by the redirection of requests from US Standard to the Northern California Region. For more information, see Redirection (p. 68). (The EU (Ireland) Region does not have a Region-specific endpoint.)</p> <div>Important<p>Even if you use a Region-specific endpoint in a request to create a Northern California Region bucket, you must set the value of <i>LocationConstraint</i> to the same Region.</p></div>

Bucket Access

To access Amazon S3 buckets and objects that were created using *CreateBucketConfiguration*, you can use the virtual hosted-style request in all Regions. For example:

```
http://yourbucket.s3.amazonaws.com/yourobject
```

To use the path-style request, the bucket must be in the US Classic Region, or the bucket must be in the same Region as the endpoint in the request. For example:

```
http://s3.amazonaws.com/yourbucket/yourobject
```

Redirection

Amazon supports two types of redirects: temporary and permanent.

Temporary redirects automatically redirect users that do not have DNS information for the requested bucket. This occurs because DNS changes take time to propagate through the Internet. For example, if a user creates a bucket with a location constraint and immediately stores an object in the bucket, information about the bucket might not distribute throughout the Internet. Because the bucket is a sub domain of `s3.amazonaws.com`, Amazon S3 redirects it to the correct Amazon S3 location.

You can remove this (short lived) redirection latency by using a Region-specific endpoint in the bucket creation request. The `LocationConstraint` bucket parameter specifies the Region where the bucket will reside. Using the Region-specific endpoint is optional. The only Region you can do this in is US-West. For more information, see [How to Specify a Bucket's Region](#) (p. 68).

Transferring a Bucket's Contents to Another Region

Use the following process to transfer your bucket from one Region to another.

To transfer a bucket to another Region

1	Create a new Amazon S3 bucket in the Region you wish to transfer your data to.
2	Use the <code>copy</code> operation to transfer each of your objects from the source Region to the target Region. Bandwidth charges apply for this transfer. For more information, go to COPY Object .

Requester Pays Buckets

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 a non Requester Pays bucket.

Setting the requestPayment Bucket Configuration

The bucket owner and 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. If you don't, the bucket, by default, is a non-Requester Pays bucket.

You use the value, *BucketOwner*, to revert Requester Pays buckets to regular buckets. Typically, you would use *BucketOwner* when uploading data to the Amazon S3 bucket, then 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
```

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

You can freely 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, for the configuration value to take 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 expiry. The bucket owner is charged each time the requester uses pre-signed URLs that use the bucket owner's credentials.

Retrieving requestPayment Configuration

You can determine the *Payer* value 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 demonstrates 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.

Make sure to include `x-amz-request-payer` and its value in your signature calculation. For more information, see [Constructing the CanonicalizedAmzHeaders Element \(p. 30\)](#).

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 `Access Denied` errors for requests trying to get objects from Requester Pays buckets. For more information, go to [Error Code List](#).

DevPay and Requester Pays

You can use Amazon DevPay to sell the content 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 straight forward: the requester pays for data transfer and the request; the bucket owner pays for the data storage. However, the bucket owner is charged for the request if:

- 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 [Access Control \(p. 141\)](#).

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 Amazon Web Services developer portal organize your Amazon S3 usage reports by bucket. For more information, go to [Amazon S3 Pricing page](#).

Bucket Configuration Errors

The following list shows the errors Amazon S3 can return in response to bucket configuration requests.

- [MalformedXML](#)

- [MissingRequestBodyError](#)

Working with Amazon S3 Objects

Topics

- [Object Key and Metadata \(p. 75\)](#)
- [Using the AWS SDK \(p. 76\)](#)
- [Uploading Objects \(p. 79\)](#)
- [Getting Objects \(p. 120\)](#)
- [Copying Objects \(p. 127\)](#)
- [Listing Object Keys \(p. 132\)](#)

Amazon S3 is designed to store objects. All objects are stored in buckets and consist of the following:

- **Value**—The content that you are storing.
The object value can be any sequence of bytes but must be smaller than 5 TB. You can upload up to 5 GB objects in a single operation and larger objects in parts, where you upload the large object using the Multipart Upload API. For more information, see [Uploading Objects \(p. 79\)](#). You can store any number of objects in a bucket. Almost all objects have values; the single exception is a Delete Marker. For more information, see [Delete Marker](#).
- **Key**—The handle that you assign to an object that allows you retrieve it later.
For more information, see [Object Key and Metadata \(p. 75\)](#)
- **Metadata**—A set of key-value pairs with which you can store information regarding the object.
For more information, see [Object Key and Metadata \(p. 75\)](#).
- **Access Control Policy**—The access control policy that controls access to the object.
For more information, see [Access Control Policy](#).
- **Version ID**—Is a string generated by Amazon S3 when you add an object to a bucket.
Within a bucket, a key and version ID uniquely identify an object. Amazon S3 gives objects unique version IDs when they're added to a versioning-enabled bucket, and `null` when added to a versioning-suspended bucket. Objects in unversioned buckets also have a version ID of `null`. For more information, see [Versions \(p. 15\)](#).

Object Key and Metadata

Each Amazon S3 object has data, a key, and metadata. When you create an object you specify the key name. This key name uniquely identifies the object in the bucket. The name for a key is a sequence of Unicode characters whose UTF-8 encoding is at most 1024 bytes long.

In addition to the key, each Amazon S3 object has metadata. It is a set of name, value pairs. 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 to track and maintain the object.

User-Defined Metadata

When uploading an object, you can 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 as 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 and when you retrieve the object using the SOAP API, the prefix is removed, regardless of which API you used to upload the object.

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.

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

There is no limit on number and length of metadata associated with an object. However, the PUT request header is limited to 8K in size. This limits the total size of the metadata allowed on the object.

Using the AWS SDK

Topics

- [Using the AWS SDK for Java \(p. 76\)](#)
- [Using the AWS SDK for .NET \(p. 77\)](#)
- [Using the AWS SDK for PHP \(p. 79\)](#)

You can use AWS SDK for developing applications with Amazon S3. The AWS SDK for Java, PHP, and .NET wraps the underlying REST API, simplifying your programming tasks. This section provides an overview of the AWS SDK. This section also describes how you can test the AWS SDK code samples provided in this guide.

Instead of using the AWS SDK, if your application requires it, you can send REST requests directly from your application. The REST API is documented in the *API Reference Guide* (see [API Reference](#)).

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 \(p. 80\)](#)). 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 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 includes the `TransferManagerConfiguration` class to configure advanced settings such as providing object metadata, specifying the storage class (STANDARD, REDUCED_REDUNDANCY) for storing objects, and setting object permissions using ACL. When uploading objects in parts, it provides options for you to configure the number of concurrent threads and the part size.
- **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 installing or updating to the latest version, go to <http://aws.amazon.com/eclipse>. The following tasks guide you through the creation and testing of the Java code samples provided in this section.

General Process of Creating Java Code Examples

1	Create a new AWS Java project in Eclipse. The project is pre-configured with the AWS SDK for Java and it also includes the <code>AwsCredentials.properties</code> file for your AWS security credentials.
2	Copy the code from the section you are reading to your project.
3	Update the code by providing any required data. For example, if uploading a file, provide the file path and the bucket name.
4	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 \(p. 78\)](#)
- [Testing the .NET Code Examples \(p. 78\)](#)

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 \(p. 80\)](#)). 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](#) (p. 80)) 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. 115)).
- **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 <http://aws.amazon.com/sdkfornet>. 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 new Visual Studio project using the <i>AWS Empty Project</i> template.
---	---

2	In the AWS Access Credentials dialog box, provide your AWS credentials.
3	Note that the <i>AWS Empty Project</i> template is preconfigured with an <code>App.config</code> file for your AWS credentials and the following required references. <code>AWSSDK</code> <code>System.Configuration</code> Verify your credentials in the <code>App.config</code> file.
4	Replace the code in the project file, <code>Program.cs</code> , with the code in the section you are reading.
5	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

Topics

The AWS SDK for PHP provides the API for Amazon S3 bucket and object operations. 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 on buckets and objects. This API also provides greater control when uploading objects in parts (see [Uploading Objects Using Multipart Upload \(p. 80\)](#)), such as letting you pause and resume multipart uploads, vary part sizes during the upload, or 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

The high-level API simplifies the multipart upload (see [Uploading Objects Using Multipart Upload \(p. 80\)](#)) flow. In just a few lines of code you can upload files to Amazon S3. This is recommended for simple file uploads.

Uploading Objects

Topics

- [Uploading Objects Using Multipart Upload \(p. 80\)](#)
- [Uploading Objects in a Single Operation \(p. 115\)](#)

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 \(p. 115\)](#).
- **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](#) (p. 80).

We encourage Amazon S3 customers to use Multipart Upload for objects larger than 100 MB.

Uploading Objects Using Multipart Upload

Topics

- [Multipart Upload Overview](#) (p. 80)
- [Using the AWS SDK for Java for Multipart Upload](#) (p. 84)
- [Using the AWS SDK for .NET for Multipart Upload](#) (p. 94)
- [Using the AWS SDK for PHP for Multipart Upload](#) (p. 109)
- [Using the REST API for Multipart Upload](#) (p. 115)

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. After 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.

Multipart Upload Overview

Topics

- [Concurrent Multipart Upload Operations](#) (p. 81)
- [Multipart Upload and Pricing](#) (p. 82)
- [Quick Facts](#) (p. 82)
- [API Support for Multipart Upload](#) (p. 82)
- [Multipart Upload API and Permissions](#) (p. 82)

Multipart upload is a three-step process: You initiate the upload, you upload all the object parts, and then you complete the multipart upload. Upon receiving the complete request, Amazon S3 constructs the object from the uploaded parts and stores it in your bucket.

You can get a list of parts that you have uploaded for a specific multipart upload or a list of all your in-progress multipart uploads. Each of these operations is explained in more detail in this section.

Multipart Upload Initiation

When you send an initiate multipart upload request, Amazon S3 returns a response with an upload ID that is a unique identifier for your multipart upload. You must include this upload ID whenever you upload

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 multipart upload initiation request.

Parts Upload

When uploading a part, in addition to the upload ID, you must specify a part number. You can choose any part number from 1 to 10,000. A part number uniquely identifies a part and its position within the object you are uploading. If you upload a new part using the same part number as a previously uploaded part, that 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 complete multipart upload request.

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 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 1,000 parts. If more than 1,000 parts exist in the multipart upload, you must send a series of list part requests to retrieve all the parts. Note that this parts list does not include parts that haven't completed uploading.



Note

Use the returned listing only 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 in-progress multipart uploads. An in-progress multipart upload is an upload that you have initiated, but have not yet completed or aborted. Each request returns at most 1,000 multipart uploads. If more than 1,000 multipart uploads are 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, the last complete multipart upload operation overrides any previous update. Note that 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

After 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 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	1,000
Maximum number of multipart uploads returned in a list multipart uploads request	1,000



Note

If you create an object using the multipart upload APIs, currently you cannot COPY the object between regions.

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 it, you can use the REST API directly.

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, the bucket policy, or the user policy.

Action	Required Permissions
Initiate Multipart Upload	<p>You must be allowed to perform the <code>s3:PutObject</code> action on an object to initiate multipart upload.</p> <p>The bucket owner can allow other principals to perform the <code>s3:PutObject</code> action.</p>
Upload Part	<p>You must be allowed to perform the <code>s3:PutObject</code> action on an object to upload a part.</p> <p>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.</p>
Complete Multipart Upload	<p>You must be allowed to perform the <code>s3:PutObject</code> action on an object to complete a multipart upload.</p> <p>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.</p>
Abort Multipart Upload	<p>You must be allowed to perform the <code>s3:AbortMultipartUpload</code> action to abort a multipart upload.</p> <p>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.</p> <p>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.</p>
List Parts	<p>You must be allowed to perform the <code>s3:ListMultipartUploadParts</code> action to list parts in a multipart upload.</p> <p>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.</p> <p>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.</p>
List Multipart Uploads	<p>You must be allowed to perform the <code>s3:ListBucketMultipartUploads</code> action on a bucket to list in-progress multipart uploads to that bucket.</p> <p>In addition to the default, the bucket owner can allow other principals to perform the <code>s3:ListBucketMultipartUploads</code> action on the bucket.</p>

For additional information on actions and their use in policies, see [Action \(p. 226\)](#). For information on the relationship between policy actions and ACL permissions, see [Relationship Between Actions and Permissions \(p. 154\)](#). For information on IAM Users, go to [Working with Users and Groups](#).

Using the AWS SDK for Java for Multipart Upload

Topics

- [Using the High-Level Java API for Multipart Upload \(p. 84\)](#)
- [Using the Low-Level Java API for Multipart Upload \(p. 89\)](#)

As described in [Using the AWS SDK](#) (see [Using the AWS SDK \(p. 76\)](#)), the SDK for Java provides support for the multipart upload API (see [Uploading Objects Using Multipart Upload \(p. 80\)](#)). This section provides examples of using the both high-level and the low-level Java SDK API for uploading objects in parts.

Using the High-Level Java API for Multipart Upload

Topics

- [Upload a File \(p. 84\)](#)
- [Abort Multipart Uploads \(p. 85\)](#)
- [Track Multipart Upload Progress \(p. 86\)](#)

The AWS SDK for Java exposes a high-level API that simplifies multipart upload (see [Uploading Objects Using Multipart Upload \(p. 80\)](#)). You can upload data from a file or a stream. You can optionally 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.

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 by providing your AWS credentials.
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 sample demonstrates the preceding tasks.

```
AWSCredentials myCredentials = new BasicAWSCredentials(myAccessKeyID,
mySecretKey);

TransferManager tm = new TransferManager(myCredentials);
// Asynchronous call.
Upload upload = tm.upload(existingBucketName, keyName, new File(filePath));
```

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 \(p. 77\)](#)

```
import java.io.File;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.Upload;

public class HighLevel_Java_UploadFile {

    public static void main(String[] args) throws Exception {
        String existingBucketName = "**** Provide existing bucket name ****";
        String keyName           = "**** Provide object key ****";
        String filePath           = "**** Provide file to upload ****";

        TransferManager tm = new TransferManager(new PropertiesCredentials(
            HighLevel_Java_UploadFile.class.getResourceAsStream(
                "AwsCredentials.properties")));

        // TransferManager processes all transfers asynchronously,
        // so this call will return immediately.
        Upload upload = tm.upload(
            existingBucketName, keyName, new File(filePath));

        try {
            // Or you can block and wait for the upload to finish
            upload.waitForCompletion();
        } 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 \(p. 82\)](#)), 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 by providing your AWS credentials.
2	Execute the <code>TransferManager.abortMultipartUploads</code> method by passing the bucket name and a <code>Date</code> value.

The following Java code sample demonstrates the preceding tasks.

```
AWSCredentials myCredentials = new BasicAWSCredentials(myAccessKeyID,
mySecretKey);

TransferManager tm = new TransferManager(myCredentials);
tm.abortMultipartUploads(existingBucketName, someDate);
```

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 \(p. 77\)](#)

```
import java.util.Date;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.transfer.TransferManager;

public class HighLevel_Java_AbortMultipartUploads {

    public static void main(String[] args) throws Exception {
        String existingBucketName = "*** Provide existing bucket name ***";

        TransferManager tm = new TransferManager(new PropertiesCredentials(
            HighLevel_Java_UploadFile.class.getResourceAsStream(
                "AwsCredentials.properties")));

        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 \(p. 93\)](#).

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.

```
AWSCredentials myCredentials = new BasicAWSCredentials(myAccessKeyID,
mySecretKey);

TransferManager tm = new TransferManager(myCredentials);

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 \(p. 77\)](#)

```
import java.io.File;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.model.ProgressEvent;
import com.amazonaws.services.s3.model.ProgressListener;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.Upload;

public class HighLevel_Java_UploadFile {

    public static void main(String[] args) throws Exception {
        String existingBucketName = "*** Provide bucket name ***";
        String keyName           = "*** Provide object key ***";
        String filePath           = "*** Provide file to upload ***";

        TransferManager tm = new TransferManager(new PropertiesCredentials(
            HighLevel_Java_UploadFile.class.getResourceAsStream(
                "AwsCredentials.properties")));

        // 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.setProgressListener(new ProgressListener() {
            public void progressChanged(ProgressEvent event) {
                System.out.println("Transferred bytes: " +
                    event.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 \(p. 89\)](#)
- [List Multipart Uploads \(p. 92\)](#)
- [Abort a Multipart Upload \(p. 93\)](#)

The AWS SDK for Java exposes a low-level API that closely resembles to the Amazon S3 REST API for multipart upload (see [Uploading Objects Using Multipart Upload \(p. 80\)](#)).

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, by providing your AWS credentials.
2	Initiate multipart upload by executing the <code>AmazonS3Client.initiateMultipartUpload</code> method. You will need to provide information required, 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 PropertiesCredentials(
    LowLevel_Java_UploadFile.class.getResourceAsStream(
        "AwsCredentials.properties")));

// 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 \(p. 77\)](#).

```
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import com.amazonaws.auth.PropertiesCredentials;
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.ListMultipartUploadsRequest;
import com.amazonaws.services.s3.model.ListPartsRequest;
import com.amazonaws.services.s3.model.MultipartUploadListing;
import com.amazonaws.services.s3.model.PartETag;
import com.amazonaws.services.s3.model.PartListing;
import com.amazonaws.services.s3.model.UploadPartRequest;

public class LowLevel_Java_UploadFile {

    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 PropertiesCredentials(
            LowLevel_Java_UploadFile.class.getResourceAsStream(
                "AwsCredentials.properties")));

        // 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 the key name. The following Java code sample demonstrates how you can abort a multipart upload in progress.

```
AWSCredentials myCredentials = new
    BasicAWSCredentials(myAccessKeyID, mySecretKey);

InitiateMultipartUploadRequest initRequest =
    new InitiateMultipartUploadRequest(existingBucketName, keyName);
InitiateMultipartUploadResult initResponse =
    s3Client.initiateMultipartUpload(initRequest);

AmazonS3 s3Client = new AmazonS3Client(myCredentials);
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 or aborted. For more information, see [Abort Multipart Uploads \(p. 85\)](#).

Using the AWS SDK for .NET for Multipart Upload

Topics

- [Using the High-Level .NET API for Multipart Upload \(p. 94\)](#)
- [Using the Low-Level .NET API for Multipart Upload \(p. 103\)](#)

As described in [Using the AWS SDK](#) (see [Using the AWS SDK \(p. 76\)](#)), the SDK for .NET provides support for the multipart upload API (see [Uploading Objects Using Multipart Upload \(p. 80\)](#)). This section provides examples of using the the high-level and the low-level Java SDK API for uploading objects in parts.

Using the High-Level .NET API for Multipart Upload

Topics

- [Upload a File \(p. 94\)](#)
- [Upload a Directory \(p. 96\)](#)
- [Abort Multipart Uploads \(p. 99\)](#)
- [Track Multipart Upload Progress \(p. 100\)](#)

The AWS SDK for .NET exposes a high-level API that simplifies multipart upload (see [Uploading Objects Using Multipart Upload \(p. 80\)](#)). 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(accessKeyID, secretAccessKey);  
utility.Upload(filePath, existingBucketName);
```

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 \(p. 78\)](#)

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

namespace s3.amazon.com.docsamples.highlevel_uploadfile
{
    class Program
    {
        static string accessKeyID      = "";
        static string secretAccessKey = "";

        static string existingBucketName = "*** Provide bucket name ***";
        static string keyName            = "*** Provide your object key ***";
        static string filePath           = "*** Provide file name ***";

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

            try
            {
                TransferUtility fileTransferUtility = new
                    TransferUtility(accessKeyID, secretAccessKey);

                // 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()  
        .WithBucketName(existingBucketName)  
        .WithFilePath(filePath)  
        .WithStorageClass(S3StorageClass.ReducedRedundancy)  
        .WithMetadata("param1", "Value1")  
        .WithMetadata("param2", "Value2")  
        .WithPartSize(6291456) // This is 6 MB.  
        .WithKey(keyName)  
        .WithCannedACL(S3CannedACL.PublicRead);  
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(accessKeyID, secretAccessKey);  
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 \(p. 78\)](#)

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

namespace s3.amazon.com.docsamples.highlevel_uploadDirectory
{
    class Program
    {
        static string accessKeyID      = "";
        static string secretAccessKey   = "";

        static string existingBucketName = "*** Provide bucket name ***";
        static string directoryPath      = "*** Provide directory name ***";

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

            try
            {
                TransferUtility directoryTransferUtility =
                    new TransferUtility(accessKeyID, secretAccessKey);

                // 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 trUtilDirUpReq =
                    new TransferUtilityUploadDirectoryRequest()
                    .WithBucketName(existingBucketName)
                    .WithDirectory(directoryPath)
                    .WithSearchOption(SearchOption.AllDirectories)
                    .WithSearchPattern("*.txt");
            }
        }
    }
}
```



```
        directoryTransferUtility.UploadDirectory(trUtilDirUpReq);  
        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 \(p. 82\)](#)), 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.AbortMultipartUploads</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(accessKeyID, secretAccessKey);  
utility.AbortMultipartUploads(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 \(p. 78\)](#)

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

namespace s3.amazon.com.docsamples.highlevel_abortmultipartupload
{
    class Program
    {
        static string accessKeyID      = "";
        static string secretAccessKey   = "";

        static string existingBucketName = "****Provide bucket name****";

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

            try
            {
                TransferUtility transferUtility =
                    new TransferUtility(accessKeyID, secretAccessKey);
                // 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 \(p. 107\)](#).

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(accessKeyID, secretAccessKey);

// Use TransferUtilityUploadRequest to configure options.
// In this example we subscribe to an event.
TransferUtilityUploadRequest uploadRequest =
    new TransferUtilityUploadRequest()
        .WithBucketName(existingBucketName)
        .WithFilePath(filePath);

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 \(p. 78\)](#)

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

namespace s3.amazon.com.docsamples.highlevel_trackprogress
{
    class Program
    {
        static string accessKeyID      = "";
        static string secretAccessKey  = "";

        static string existingBucketName = "*** Provide bucket name ***";
        static string keyName           = "*** Provide key name ***";
        static string filePath          = "*** Provide file to upload ***";

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

            try
            {
                TransferUtility fileTransferUtility =
                    new TransferUtility(accessKeyID, secretAccessKey);

                // Use TransferUtilityUploadRequest to configure options.
                // In this example we subscribe to an event.
                TransferUtilityUploadRequest uploadRequest =
                    new TransferUtilityUploadRequest()
                        .WithBucketName(existingBucketName)
                        .WithFilePath(filePath);

                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 \(p. 103\)](#)
- [List Multipart Uploads \(p. 107\)](#)
- [Track Multipart Upload Progress \(p. 108\)](#)
- [Abort a Multipart Upload \(p. 108\)](#)

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 \(p. 115\)](#)). 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 \(p. 94\)](#)), 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 UploadingProcess

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.

```
AmazonS3 s3Client = new AmazonS3Client(AccessKeyID, SecretAccessKey);

// List to store upload part responses.
List<UploadPartResponse> uploadResponses = new List<UploadPartResponse>();

// 1. Initialize.
InitiateMultipartUploadRequest initRequest =
    new InitiateMultipartUploadRequest()
        .WithBucketName(existingBucketName)
        .WithKey(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()
            .WithBucketName(existingBucketName)
            .WithKey(keyName)
            .WithUploadId(initResponse.UploadId)
            .WithPartNumber(i)
            .WithPartSize(partSize)
            .WithFilePosition(filePosition)
            .WithFilePath(filePath);

        // Upload part and add response to our list.
        uploadResponses.Add(s3Client.UploadPart(uploadRequest));

        filePosition += partSize;
    }

    // Step 3: complete.
    CompleteMultipartUploadRequest compRequest =
        new CompleteMultipartUploadRequest()
            .WithBucketName(existingBucketName)
            .WithKey(keyName)
            .WithUploadId(initResponse.UploadId)
            .WithPartETags(uploadResponses);

    CompleteMultipartUploadResponse completeUploadResponse =
        s3Client.CompleteMultipartUpload(compRequest);
}
catch (Exception exception)
{
    Console.WriteLine("Exception occurred: {0}", exception.Message);
    s3Client.AbortMultipartUpload(new AbortMultipartUploadRequest())
}
```

```
.WithBucketName(existingBucketName)  
.WithKey(keyName)  
.WithUploadId(initResponse.UploadId);  
}
```

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 \(p. 78\)](#)

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

namespace s3.amazon.com.docsamples.LowLevel_UploadFromFile
{
    class Program
    {
        // Your AWS Credentials.
        static string AccessKeyID      = "";
        static string SecretAccessKey  = "";

        static string existingBucketName = "*** Provide bucket name";
        static string keyName = "*** Provide object key ***";
        static string filePath = "*** Provide file to upload ***";

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

            AmazonS3 s3Client =
                new AmazonS3Client(AccessKeyID, SecretAccessKey);

            // List to store upload part responses.
            List<UploadPartResponse> uploadResponses =
                new List<UploadPartResponse>();

            // 1. Initialize.
            InitiateMultipartUploadRequest initiateRequest =
                new InitiateMultipartUploadRequest()
                    .WithBucketName(existingBucketName)
                    .WithKey(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++)
                {

                    // Create request to upload a part.
```



```
UploadPartRequest uploadRequest = new UploadPartRequest()  
    .WithBucketName(existingBucketName)  
    .WithKey(keyName)  
    .WithUploadId(initResponse.UploadId)  
    .WithPartNumber(i)  
    .WithPartSize(partSize)  
    .WithFilePosition(filePosition)  
    .WithFilePath(filePath);  
  
    // Upload part and add response to our list.  
    uploadResponses.Add(s3Client.UploadPart(uploadRequest));  
  
    filePosition += partSize;  
}  
  
// Step 3: complete.  
CompleteMultipartUploadRequest completeRequest =  
    new CompleteMultipartUploadRequest()  
        .WithBucketName(existingBucketName)  
        .WithKey(keyName)  
        .WithUploadId(initResponse.UploadId)  
        .WithPartETags(uploadResponses);  
  
CompleteMultipartUploadResponse completeUploadResponse =  
    s3Client.CompleteMultipartUpload(completeRequest);  
  
}  
catch (Exception exception)  
{  
    Console.WriteLine("Exception occurred: {0}", exception.Message);  
  
    s3Client.AbortMultipartUpload(new AbortMultipartUploadRequest()  
  
        .WithBucketName(existingBucketName)  
        .WithKey(keyName)  
        .WithUploadId(initResponse.UploadId));  
}  
}  
}
```

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 <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>ListMultipartUploadsResponse</code> class, providing you the information about the in-progress multipart uploads.

The following C# code sample demonstrates the preceding tasks.

```
ListMultipartUploadsRequest allMultipartUploadsRequest = new ListMultipartUploadsRequest()  
    .WithBucketName(existingBucketName);  
ListMultipartUploadsResponse mpUploadsResponse = s3Client.ListMultipartUploads(allMultipartUploadsRequest);
```

Track Multipart Upload Progress

The low-level multipart upload API provides an event, `UploadPartRequest.UploadPartProgressEvent`, 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 `UploadPartProgressEvent` event and write a handler.

```
UploadPartRequest uploadRequest = new UploadPartRequest();  
// Provide request data (for example, bucket name, key name and part number).  
// ...  
// Subscribe to the event.  
uploadRequest.UploadPartProgressEvent += new  
EventHandler<UploadPartProgressArgs>(uploadRequest.UploadPartProgressEvent);  
  
// Sample event handler.  
static void uploadRequest_UploadPartProgressEvent(object sender,  
                                                    UploadPartProgressArgs 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()  
    .WithBucketName(existingBucketName)  
    .WithKey(keyName)  
    .WithUploadId(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. 99).

Using the AWS SDK for PHP for Multipart Upload

Topics

- [Using the High-Level PHP API for Multipart Upload \(p. 109\)](#)
- [Using the Low-Level PHP API for Multipart Upload \(p. 110\)](#)

The AWS SDK for PHP provides support for the multipart upload API (see [Uploading Objects Using Multipart Upload \(p. 80\)](#)).

High-level API

The high-level API simplifies 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 low-level APIs correspond to the multipart upload REST operations (see [Using the REST API for Multipart Upload \(p. 115\)](#)). 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 High-Level PHP API for Multipart Upload

The AWS SDK for PHP exposes a high-level method that simplifies the multipart upload flow (see [Uploading Objects Using Multipart Upload \(p. 80\)](#)). You can upload data from a file or an open file stream. You can optionally set advanced options, such as the part size you want to use for the multipart upload, optional file metadata, the storage class (STANDARD or REDUCED_REDUNDANCY), or ACL.

Upload a File

The following tasks guide you through using the high-level PHP classes to upload a file.

High-Level API File Uploading Process

1	Create an instance of the <code>AmazonS3</code> class by providing your AWS credentials.
2	Execute the <code>AmazonS3::create_mpu_object()</code> method.

The following PHP code sample demonstrates the preceding tasks.

```
// Instantiate the class
$s3 = new AmazonS3();

$response = $s3->create_mpu_object($bucket, $keyname, array(
    'fileUpload' => $filepath
));

print_r($response);
```

Example

The following PHP example uploads a file to an Amazon S3 bucket. The example illustrates the use of the `AmazonS3::create_mpu_object()` method to upload a file. Each successive call to upload replaces the previous upload.

```
<?php
require_once '/path/to/sdk.class.php';

$bucket = '*** Provide your existing bucket name ***';
$keyname = '*** Provide object key ***';
$filepath = '*** Provide file to upload ***';

// Define a mebibyte
define('MB', 1048576);

// Instantiate the class
$s3 = new AmazonS3();

// 1. Upload file. Specify object key name explicitly.
$response = $s3->create_mpu_object($bucket, $keyname, array(
    'fileUpload' => $filepath
));

// Success?
header('Content-Type: text/plain; charset=utf-8');
print_r($response);
echo "Upload 1 completed!" . PHP_EOL . PHP_EOL;

// 2. Specify optional configuration.
$response = $s3->create_mpu_object($bucket, $keyname, array(
    'fileUpload' => $filepath,

    // Optional configuration
    'partSize' => 40*MB, // Defaults to 50MB
    'acl' => AmazonS3::ACL_PUBLIC,
    'storage' => AmazonS3::STORAGE_REDUCED,

    // Object metadata.
    'meta' => array(
        'param1' => 'value1',
        'param2' => 'value2',
    )
));

// Success?
print_r($response);
echo "Upload 2 completed!";
```

Using the Low-Level PHP API for Multipart Upload

Topics

- [Upload a File \(p. 111\)](#)
- [List Multipart Uploads \(p. 114\)](#)
- [Abort a Multipart Upload \(p. 114\)](#)

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 \(p. 115\)](#)). 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 PHP API for Multipart Upload \(p. 109\)](#)) whenever you don't have these requirements.

Upload a File

The following tasks guide you through using the low-level PHP classes to upload a file.

Low-Level API File UploadingProcess

1	Create an instance of the <code>AmazonS3</code> class, by providing your AWS credentials.
2	Initiate multipart upload by executing the <code>AmazonS3::initiate_multipart_upload()</code> method. You need to provide the required information, i.e., bucket name and object key.
3	Retrieve the <code>UploadID</code> from the response body. You will need to provide this upload ID for each subsequent multipart upload operation.
4	Call the <code>AmazonS3::get_multipart_counts()</code> to generate parts information. You need to provide the input file and the desired part size. The method returns an associative array of all the parts information, such as the part length and its starting position in the file.
5	Upload parts by executing the <code>AmazonS3::upload_part()</code> method. Save the response of each of the <code>upload_part()</code> methods in an array. Each response includes the ETag value you will later need to complete the multipart upload.
6	Execute the <code>AmazonS3::complete_multipart_upload()</code> method to complete the multipart upload.

The following PHP code sample demonstrates the preceding tasks.

```
// Instantiate the class.
$s3 = new AmazonS3();

// Define a megabyte
define('MB', 1048576);

// 1. Initiate a new multipart upload.
$response = $s3->initiate_multipart_upload($bucket, $keyname);

// Get the Upload ID.
$upload_id = (string) $response->body->UploadId;

// 2. Upload parts.
// Get part list for a given input file and given part size.
// Returns an associative array.
$parts = $s3->get_multipart_counts(filesize($filepath), 5*MB);

$responses = new CFArray(array());

foreach ($parts as $i => $part)
{
    // Upload part and save response in an array.
    $responses[] = $s3->upload_part($bucket, $keyname, $upload_id, array(
```

```
        'fileUpload' => $filepath,  
        'partNumber' => ($i + 1),  
        'seekTo' => (integer) $part['seekTo'],  
        'length' => (integer) $part['length'],  
    ));  
}  
  
// 3. Complete multipart upload. We need all part numbers and ETag values.  
$parts = array();  
foreach ($responses as $i => $response)  
{  
    $parts[] = array(  
        'PartNumber' => ($i + 1),  
        'ETag' => (string) $response->header['etag']  
    );  
}  
  
$response = $s3->complete_multipart_upload(  
    $bucket, $keyname, $upload_id, $parts);
```

Example

The following PHP code example uploads a file to an Amazon S3 bucket.

```
<?php
require_once '/path/to/sdk.class.php';

$bucket = '*** Provide your existing bucket name ***';
$keyname = '*** Provide object key name ***';
$filepath = '*** Provide file name to upload ***';

// Define a megabyte.
define('MB', 1048576);

// Instantiate the class
$s3 = new AmazonS3();

// 1. Initiate a new multipart upload. (Array parameter is optional)
$response = $s3->initiate_multipart_upload($bucket, $keyname, array(
    'acl' => AmazonS3::ACL_PUBLIC,
    'storage' => AmazonS3::STORAGE_REDUCED,
    'meta' => array(
        'param1' => 'value 1',
        'param2' => 'value 2',
        'param3' => 'value 3'
    )
));

if (!$response->isOK())
{
    throw new S3_Exception('Bad!');
}

// Get the Upload ID.
$upload_id = (string) $response->body->UploadId;

// 2. Upload parts.
// Get part list for a given input file and given part size.
// Returns an associative array.
$parts = $s3->get_multipart_counts(filesize($filepath), 5*MB);

$responses = new CFArray(array());

foreach ($parts as $i => $part)
{
    // Upload part and save response in an array.
    $responses[] = $s3->upload_part($bucket, $keyname, $upload_id, array(
        'fileUpload' => $filepath,
        'partNumber' => ($i + 1),
        'seekTo' => (integer) $part['seekTo'],
        'length' => (integer) $part['length'],
    ));
}

// Verify that no part failed to upload, otherwise abort.
if (!$responses->areOK())
{
    // Abort an in-progress multipart upload
```

```
$response = $s3->abort_multipart_upload($bucket, $keyname, $upload_id);

throw new S3_Exception('Failed!');
}

// 3. Complete the multipart upload. We need all part numbers and ETag values.
$parts = array();
foreach ($responses as $i => $response)
{
    $parts[] = array(
        'PartNumber' => ($i + 1),
        'ETag' => (string) $response->header['etag']
    );
}

$response = $s3->complete_multipart_upload(
    $bucket, $keyname, $upload_id, $parts);

// Display the results
header('Content-Type: text/plain; charset=utf-8');
print_r($response);

if ($response->isOk())
{
    echo 'Object uploaded!';
}
```

List Multipart Uploads

The following tasks guide you through using the low-level PHP 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>AmazonS3</code> class.
2	Execute the <code>AmazonS3::list_multipart_uploads()</code> method by providing a bucket name. The method returns all of the in-progress multipart uploads on that bucket.

The following PHP code sample demonstrates the preceding tasks.

```
$s3 = new AmazonS3();
$response = $s3->list_multipart_uploads($bucket);
print_r($response);
```

Abort a Multipart Upload

You can abort a multipart upload that is in progress by calling the `AmazonS3::abort_multipart_upload()` method. This method deletes any parts that were uploaded to S3 and frees up the resources. You must provide the upload ID, bucket name, and the object key to this method. The following PHP code sample demonstrates how you can abort a multipart upload in progress.


```
$s3 = new AmazonS3();  
$response = $s3->abort_multipart_upload($bucket, $keyname, $upload_id);  
print_r($response);
```

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. 82).

Uploading Objects in a Single Operation

Topics

- [Upload an Object Using the AWS SDK for Java](#) (p. 115)
- [Upload an Object Using the AWS SDK for .NET](#) (p. 116)
- [Upload an Object Using the REST API](#) (p. 120)

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.

Java API File Uploading Process

1	Create an instance of the <code>AmazonS3Client</code> class by providing your AWS credentials.
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.

```
AWSCredentials myCredentials = new BasicAWSCredentials(  
    myAccessKeyID, mySecretKey);  
AmazonS3 s3client = new AmazonS3Client(myCredentials);  
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 \(p. 77\)](#)

```
import java.io.File;
import java.io.IOException;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.PutObjectRequest;

public class S3Sample {
    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 PropertiesCredentials(
            S3Sample.class.getResourceAsStream(
                "AwsCredentials.properties")));
        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.

.NET API File Uploading Process

1	Create an instance of the <code>AmazonS3</code> class by providing your AWS credentials.
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 AmazonS3 client;
client = Amazon.AWSClientFactory.CreateAmazonS3Client(
    accessKeyID, secretAccessKeyID);

PutObjectRequest request = new PutObjectRequest();
request.WithFilePath(filePath)
    .WithBucketName(bucketName)
    .WithKey(keyName);
S3Response responseWithMetadata =
    client.PutObject(titledRequest);
```

Example

The following C# code example uploads an object. The object data is provided as a text string in the code. The example illustrates the use of the `AmazonS3.PutObject` to upload an object. The example uploads the object twice. In the first object upload, the `PutObjectRequest` specifies only the bucket name, key name, and sample object data. In the second object upload the `PutObjectRequest` provides additional information including the optional object metadata and a content type header. 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 \(p. 78\)](#)

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using System.Net;

using Amazon.S3;
using Amazon.S3.Model;
using Amazon.S3.Util;

namespace s3.amazon.com.docsamples.createobject
{
    class S3Sample
    {
        static string bucketName = "**** Provide bucket name ****";
        static string keyName = "**** Provide key name ****";

        static AmazonS3 client;

        public static void Main(string[] args)
        {
            if (checkRequiredFields())
            {
                NameValueCollection appConfig =
                    ConfigurationManager.AppSettings;

                string accessKeyID = appConfig["AWSAccessKey"];
                string secretAccessKeyID = appConfig["AWSSecretKey"];

                using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID))
                {
                    Console.WriteLine("Uploading an object");
                    WritingAnObject();
                }

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

            static void WritingAnObject()
            {
                try
                {
                    // 1. Simple object put.
                    PutObjectRequest request = new PutObjectRequest();
                    request.WithContentBody("Object data for simple put.")
                }
            }
        }
    }
}
```

```
        .WithBucketName(bucketName)
        .WithKey(keyName);

    S3Response response = client.PutObject(request);
    response.Dispose();

    // 2. Put a more complex object with metadata and http headers.

    PutObjectRequest request2 = new PutObjectRequest();
    request2.WithMetaData("title", "the title")
        .WithContentBody("Object data for complex put.")
        //.WithFilePath(filePath)
        .WithBucketName(bucketName)
        .WithKey(keyName);
    // Add a header to the request.
    request2.AddHeaders(AmazonS3Util.CreateHeaderEntry
        ("ContentType", "text/xml"));

    S3Response responseWithMetadata = client.PutObject(request2);
}
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);
    }
}

static bool checkRequiredFields()
{
    NameValueCollection appConfig = ConfigurationManager.AppSettings;

    if (string.IsNullOrEmpty(appConfig["AWSAccessKey"]))
    {
        Console.WriteLine(
            "AWSAccessKey was not set in the App.config file.");
        return false;
    }
    if (string.IsNullOrEmpty(appConfig["AWSSecretKey"]))
    {
        Console.WriteLine(
            "AWSSecretKey was not set in the App.config file.");
        return false;
    }
}
```

```
    }  
    if (string.IsNullOrEmpty(bucketName))  
    {  
        Console.WriteLine("The variable bucketName is not set.");  
        return false;  
    }  
    if (string.IsNullOrEmpty(keyName))  
    {  
        Console.WriteLine("The variable keyName is not set.");  
        return false;  
    }  
  
    return true;  
}  
}
```

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](#).

Getting Objects

Topics

- [Retrieve an Object Using the AWS SDK for Java \(p. 120\)](#)
- [Retrieve an Object Using the AWS SDK for .NET \(p. 123\)](#)
- [Retrieve an Object Using REST API \(p. 126\)](#)

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.

Retrieve 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.

Java API Object Retrieving Process

- | | |
|---|--|
| 1 | Create an instance of the <code>AmazonS3Client</code> class by providing your AWS credentials. |
|---|--|

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.

```
AWSCredentials myCredentials = new BasicAWSCredentials(
    myAccessKeyID, mySecretKey);
AmazonS3 s3Client = new AmazonS3Client(myCredentials);
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.

```
AWSCredentials myCredentials = new BasicAWSCredentials(
    myAccessKeyID, mySecretKey);
AmazonS3 s3Client = new AmazonS3Client(myCredentials);

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();
```

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 \(p. 77\)](#)

```
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.PropertiesCredentials;
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 S3Sample {
    private static String bucketName = "**** Provide bucket name ****";
    private static String key = "**** Provide Key ****";

    public static void main(String[] args) throws IOException {
        AmazonS3 s3Client = new AmazonS3Client(new PropertiesCredentials(
            S3Sample.class.getResourceAsStream(
                "AwsCredentials.properties")));
        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.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 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();  
}  
}
```

Retrieve 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.

.NET API File Uploading Process

1	Create an instance of the <code>AmazonS3</code> class by providing your AWS credentials.
2	Execute one of the <code>AmazonS3.GetObject</code> methods. You need to provide information such as bucketname, 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.

```
static AmazonS3 client;  
client = Amazon.AWSClientFactory.CreateAmazonS3Client(  
    accessKeyID, secretAccessKeyID);  
  
GetObjectRequest request = new GetObjectRequest()  
    .WithBucketName(bucketName).WithKey(keyName);  
using (GetObjectResponse response = client.GetObject(request))  
{  
    string title = response.Metadata["x-amz-meta-title"];  
    Console.WriteLine("The object's title is {0}", title);  
    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()  
    .WithBucketName(bucketName)  
    .WithKey(keyName)  
    .WithByteRange(0, 10);
```

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. For instructions on how to create and test a working sample, see [Testing the .NET Code Examples \(p. 78\)](#)

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using System.IO;

using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.retrieveobject
{
    class S3Sample
    {
        static string bucketName = "*** Provide bucket name ***";
        static string keyName     = "*** Provide object key ***";
        static AmazonS3 client;

        public static void Main(string[] args)
        {
            if (checkRequiredFields())
            {
                NameValueCollection appConfig =
                    ConfigurationManager.AppSettings;

                string accessKeyID = appConfig["AWSAccessKey"];
                string secretAccessKeyID = appConfig["AWSSecretKey"];
                try
                {
                    Console.WriteLine("Retrieving (getting) an object");
                    string data = ReadingAnObject(
                        accessKeyID, secretAccessKeyID);
                }
                catch (AmazonS3Exception s3Exception)
                {
                    Console.WriteLine(s3Exception.Message,
                        s3Exception.InnerException);
                }
            }

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

        static string ReadingAnObject(
            string accessKeyID, string secretAccessKeyID)
        {
            string responseBody = "";
            using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                accessKeyID, secretAccessKeyID))
            {
                GetObjectRequest request = new GetObjectRequest()
                    .WithBucketName(bucketName).WithKey(keyName);
            }
        }
    }
}
```

```
        using (GetObjectResponse response = client.GetObject(request))
        {
            string title = response.Metadata["x-amz-meta-title"];
            Console.WriteLine("The object's title is {0}", title);

            using (Stream responseStream = response.ResponseStream)
            {
                using (StreamReader reader =
                    new StreamReader(responseStream))
                {
                    responseBody = reader.ReadToEnd();
                }
            }
        }
    }
    return responseBody;
}

static bool checkRequiredFields()
{
    NameValueCollection appConfig = ConfigurationManager.AppSettings;

    if (string.IsNullOrEmpty(appConfig["AWSAccessKey"]))
    {
        Console.WriteLine(
            "AWSAccessKey was not set in the App.config file.");
        return false;
    }
    if (string.IsNullOrEmpty(appConfig["AWSSecretKey"]))
    {
        Console.WriteLine(
            "AWSSecretKey was not set in the App.config file.");
        return false;
    }
    if (string.IsNullOrEmpty(bucketName))
    {
        Console.WriteLine("The variable bucketName is not set.");
        return false;
    }
    if (string.IsNullOrEmpty(keyName))
    {
        Console.WriteLine("The variable keyName is not set.");
        return false;
    }

    return true;
}
}
```

Retrieve an Object Using 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 Bucket \(List Objects\)](#).

Copying Objects

Topics

- [Copy an Object Using the AWS SDK for Java \(p. 127\)](#)
- [Copy an Object Using the AWS SDK for .NET \(p. 128\)](#)
- [Copy an Object Using the REST API \(p. 131\)](#)

The `copy` operation enables you to copy objects within Amazon S3. 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., Northern California and EU)
- Copy objects that are less than 5 GB in size. You can upload objects of up to 5 TB in a bucket. However, Amazon S3 supports only the `copy` operation only for objects that are less than 5 GB in size.



Note

Copying objects across locations incurs bandwidth charges.

For more information, go to [PUT Object \(Copy\)](#).

Copy an Object Using the AWS SDK for Java

The following tasks guide you through using the Java classes to copy an object in S3.

Java API Object Copying Process

1	Create an instance of the <code>AmazonS3Client</code> class by providing your AWS credentials.
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.

```
AWSCredentials myCredentials = new BasicAWSCredentials(  
    myAccessKeyID, mySecretKey);  
AmazonS3 s3client = new AmazonS3Client(myCredentials);  
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 \(p. 77\)](#)

```
import java.io.IOException;
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.CopyObjectRequest;

public class S3Sample {
    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 PropertiesCredentials(
            S3Sample.class.getResourceAsStream(
                "AwsCredentials.properties")));

        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.

.NET API File Uploading Process

1	Create an instance of the <code>AmazonS3</code> class by providing your AWS credentials.
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 AmazonS3 client;
client = Amazon.AWSClientFactory.CreateAmazonS3Client(
    accessKeyID, secretAccessKeyID);

CopyObjectRequest request = new CopyObjectRequest();
request.SourceBucket = bucketName;
request.SourceKey = keyName;
request.DestinationBucket = bucketName;
request.DestinationKey = destKeyName;
S3Response response = client.CopyObject(request);
```

Example

The following C# code example makes a copy of an object in the same source bucket. The example illustrates the use of the `AmazonS3.CopyObject` method. For instructions on how to create and test a working sample, see [Testing the .NET Code Examples \(p. 78\)](#)

```
using System;
using System.Configuration;
using System.Collections.Specialized;

using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.copyobject
{
    class S3Sample
    {
        static string bucketName = "**** Provide bucket name ****";
        static string keyName = "**** Provide key name ****";
        static string destKeyName = "**** Provide destination key name ****";
        static AmazonS3 client;

        public static void Main(string[] args)
        {
            if (checkRequiredFields())
            {
                NameValueCollection appConfig =
                    ConfigurationManager.AppSettings;

                string accessKeyID = appConfig["AWSAccessKey"];
                string secretAccessKeyID = appConfig["AWSSecretKey"];

                using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID))
                {
                    Console.WriteLine("Copying an object");
                    CopyingObject();
                }
            }

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

        static bool checkRequiredFields()
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;

            if (string.IsNullOrEmpty(appConfig["AWSAccessKey"]))
            {
                Console.WriteLine(
                    "AWSAccessKey was not set in the App.config file.");
                return false;
            }
            if (string.IsNullOrEmpty(appConfig["AWSSecretKey"]))
            {
                Console.WriteLine(
                    "AWSSecretKey was not set in the App.config file.");
            }
        }
    }
}
```



```
        return false;
    }
    if (string.IsNullOrEmpty(bucketName))
    {
        Console.WriteLine("The variable bucketName is not set.");
        return false;
    }
    if (string.IsNullOrEmpty(keyName))
    {
        Console.WriteLine("The variable keyName is not set.");
        return false;
    }

    return true;
}

static void CopyingObject()
{
    try
    {
        // simple object put
        CopyObjectRequest request = new CopyObjectRequest();
        request.SourceBucket = bucketName;
        request.SourceKey = keyName;
        request.DestinationBucket = bucketName;
        request.DestinationKey = destKeyName;
        S3Response response = client.CopyObject(request);
        response.Dispose();
    }
    catch (AmazonS3Exception s3Exception)
    {
        Console.WriteLine(s3Exception.Message,
                          s3Exception.InnerException);
    }
}
}
```

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 15B4D3461F177624206A:ENoSbxYByFA0UGLZUqJN5EUUnLDg=
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: Vyaxt7qEbzv34BnSu5hctyyNSlHTYZFMWK4FtzO+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>
```

Listing Object Keys

Topics

- [Iterating Through Multi-Page Results \(p. 133\)](#)
- [Listing Keys Hierarchically Using Prefix and Delimiter \(p. 133\)](#)
- [Listing Keys Using the AWS SDK for Java \(p. 134\)](#)
- [Listing Keys Using the AWS SDK for .NET \(p. 136\)](#)
- [Listing Keys Using the REST API \(p. 140\)](#)

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.

You can iterate through large collections of keys by making multiple, paginated, list requests. For example, an initial list request against the dictionary bucket might retrieve only information about the keys "quack" through "quartermaster". But a subsequent request would retrieve "quarters" through "quince", and so on.

For instructions on how to correctly handle large list result sets, see [Iterating Through Multi-Page Results](#) (p. 133).

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 Prefix and Delimiter](#) (p. 133).

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 uses 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.

Listing Keys Hierarchically Using 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/California/San Francisco
- North America/USA/Washington/Seattle

and so on.

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.

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.

Java API Object Keys Listing Process

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.

```
AWSCredentials myCredentials = new BasicAWSCredentials(
    myAccessKeyID, mySecretKey);
AmazonS3 s3client = new AmazonS3Client(myCredentials);

ObjectListing objectListing = s3client.listObjects(new
    ListObjectsRequest()
        .withBucketName(bucketName)
        .withPrefix("m")
    );

do {
    for (S3ObjectSummary objectSummary :
        objectListing.getObjectSummaries()) {
        System.out.println(" - " + objectSummary.getKey() + "  " +
            "(size = " + objectSummary.getSize() +
            ")");
    }
    objectListing = s3client.listNextBatchOfObjects(objectListing);
} 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 \(p. 77\)](#)

```
import java.io.IOException;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.PropertiesCredentials;
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 S3Sample {
    private static String bucketName = "*** Provide bucket name ***";

    public static void main(String[] args) throws IOException {
        AmazonS3 s3client = new AmazonS3Client(new PropertiesCredentials(
            S3Sample.class.getResourceAsStream(
                "AwsCredentials.properties")));
        try {
            System.out.println("Listing objects");

            ObjectListing objectListing = s3client.listObjects(new
                ListObjectsRequest()
                    .withBucketName(bucketName)
                    .withPrefix("m")
                );

            do {
                for (S3ObjectSummary objectSummary :
                    objectListing.getObjectSummaries()) {
                    System.out.println(" - " + objectSummary.getKey() + " " +
                        "(size = " + objectSummary.getSize() +
                        ")");
                }
                objectListing = s3client.listNextBatchOfObjects(objectListing);
            } 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.

.NET API Listing Keys Process

1	Create an instance of the <code>AmazonS3</code> class by providing your AWS credentials.
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 AmazonS3 client;  
client = Amazon.AWSClientFactory.CreateAmazonS3Client(  
    accessKeyID, secretAccessKeyID);  
  
ListObjectsRequest request = new ListObjectsRequest();  
request = new ListObjectsRequest();  
request.BucketName = bucketName;  
request.WithPrefix("m");  
request.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 \(p. 78\)](#)

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

namespace s3.amazon.com.docsamples.listingkeys
{
    class S3Sample
    {
        static string bucketName = "*** Provide bucket name ***";
        static AmazonS3 client;

        public static void Main(string[] args)
        {
            if (checkRequiredFields())
            {
                NameValueCollection appConfig =
                    ConfigurationManager.AppSettings;

                string accessKeyID = appConfig["AWSAccessKey"];
                string secretAccessKeyID = appConfig["AWSSecretKey"];

                using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                    accessKeyID, secretAccessKeyID))
                {
                    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();
                request = new ListObjectsRequest();
                request.BucketName = bucketName;
                request.WithPrefix("m");
                request.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);
    }
}
}

static bool checkRequiredFields()
{
    NameValueCollection appConfig = ConfigurationManager.AppSettings;

    if (string.IsNullOrEmpty(appConfig["AWSAccessKey"]))
    {
        Console.WriteLine(
            "AWSAccessKey was not set in the App.config file.");
        return false;
    }
    if (string.IsNullOrEmpty(appConfig["AWSSecretKey"]))
    {
        Console.WriteLine(
            "AWSSecretKey was not set in the App.config file.");
        return false;
    }
    if (string.IsNullOrEmpty(bucketName))
```

```
        {  
            Console.WriteLine("The variable bucketName is not set.");  
            return false;  
        }  
  
        return true;  
    }  
}
```

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 an object or portion of the object. For more information, go to [Get Object](#).

Access Control

Topics

- [Using Bucket Policies \(p. 141\)](#)
- [Using ACLs \(p. 152\)](#)
- [Using ACLs and Bucket Policies Together \(p. 154\)](#)
- [Using Query String Authentication \(p. 155\)](#)

Amazon S3 enables you to manage access to objects and buckets using access control lists (ACLs) and bucket policies. You can use them independently or together. This section describes both.

An ACL is a list of grants. A grant consists of one grantee and one permission to access Amazon S3 resources (buckets and objects). ACLs only grant permissions; they do not deny them. ACLs can contain the following grantee types:

- Specific AWS accounts
- All AWS accounts
- Any anonymous request

Bucket policies provide access control management at the bucket level for both a bucket and the objects in it. Bucket policies are a collection of JSON statements written in the *access policy language*. The policies provide a fine granularity of access control for Amazon S3 resources. The policies also allow you to set permissions for a large number of objects with one statement.

Using Bucket Policies

Topics

- [Writing Bucket Policies \(p. 142\)](#)
- [Setting Bucket Policies on a Bucket \(p. 142\)](#)
- [Returning the Bucket Policies on a Bucket \(p. 143\)](#)
- [Deleting Bucket Policies on a Bucket \(p. 143\)](#)
- [Example Cases for Amazon S3 Bucket Policies \(p. 143\)](#)
- [How to Use Resources, Principals, Operations, and Conditions in Bucket Policies \(p. 147\)](#)

The following sections explain how to set and manage bucket policies on Amazon S3 resources.

Writing Bucket Policies

Bucket policies define access rights for Amazon S3 resources. Only a bucket owner can write bucket policies. A bucket owner can write a bucket policy to:

- Allow/deny bucket-level permissions.
- Deny permission on any objects in the bucket. Because the bucket owner is fiscally responsible for the bucket, the owner can write a bucket policy to deny permissions on any objects in a bucket.
- Grant permission on objects in the bucket only if the bucket owner is the object owner. For objects owned by other accounts the object owner must manage permissions using ACLs.

The policy itself is written in JSON and uses the access policy language. To learn about the details of the access policy language and how to write a bucket policy, see [The access policy language \(p. 213\)](#).

Setting Bucket Policies on a Bucket

To set a policy on a bucket, you use the `PUT Bucket` operation on the `policy` sub-resource and you include the bucket policy in the body of the request. The following request, for example, allows two users (1-22-333-4444, 3-55-678-9100) access execute a `GET` request (`s3:GetObject*`) for objects in `mybucket` (`arn:aws:s3:::mybucket/*`):

```
PUT /?policy HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Tue, 04 Apr 2010 20:34:56 GMT
Authorization: AWS VGhpcyBSAMPLEBieSB1bHZpbmc=

{
  "Version": "2008-10-17",
  "Id": "aaaa-bbbb-cccc-dddd",
  "Statement": [
    {
      "Effect": "Allow",
      "Sid": "1",
      "Principal": {
        "AWS": [ "1-22-333-4444", "3-55-678-9100" ]
      },
      "Action": [ "s3:GetObject*" ],
      "Resource": "arn:aws:s3:::mybucket/*",
    }
  ]
}
```



Note

The `Resource` value must include the bucket name.

To attach a policy to a bucket, you must be the bucket owner or someone authorized by the bucket owner to set a policy on the bucket by being granted `PutPolicy` permissions. The bucket owner by default has permissions to attach bucket policies to their buckets using `PUT Bucket policy`. If the bucket already has a policy, the one in this request completely replaces it

For more information, go to [PUT Bucket policy](#) in the *Amazon S3 API Reference*.

Returning the Bucket Policies on a Bucket

To return the bucket policy on a specified bucket, use the `GET Bucket` operation with the *policy* sub-resource. The following request returns the bucket policy for the bucket, `mybucket.s3.amazonaws.com`:

```
GET ?policy HTTP/1.1
Host: mybucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS 02236Q3V0WHVSRSAMPLEX5sCYVf1bNRuU=
```

The bucket owner by default has permissions to retrieve bucket policies using `GET Bucket policy`.

For more information, go to [GET Bucket policy](#) in the *Amazon S3 API Reference*.

Deleting Bucket Policies on a Bucket

To delete a policy associated with a bucket, use the `DELETE Bucket` operation with the *policy* sub-resource. The following request deletes the bucket policy associated with `mybucket`:

```
DELETE /?policy HTTP/1.1
Host: mybucket.s3.amazonaws.com
Date: Tue, 04 Apr 2010 20:34:56 GMT
Authorization: AWS VGhpcyBSAMPLEeSB1bHZpbmc=
```

To use the delete operation, you must have *DeletePolicy* permissions on the specified bucket and be the bucket owner. The bucket owner by default has permissions to delete bucket policies.

For more information, go to [DELETE Bucket policy](#) in the *Amazon S3 API Reference*.

Example Cases for Amazon S3 Bucket Policies

This section gives a few examples of typical use cases for bucket policies.

Granting Permissions to Multiple Accounts with Added Restrictions

The following example policy grants `PutObject`, and `PutObjectAcl` permissions to multiple accounts and requires that the public-read canned acl is included.

Example

```
{
  "Version": "2008-10-17",
  "Statement": [{
    "Sid": "AddCannedAcl",
    "Effect": "Allow",
    "Principal": {
      "AWS": [
        "arn:aws:iam::778695189650:root", "arn:aws:iam::178784293420:root" ]
    },
    "Action": [ "s3:PutObject", "s3:PutObjectAcl" ],
    "Resource": [ "arn:aws:s3:::bucket/*" ],
    "Condition": {
      "StringEquals": {
        "s3:x-amz-acl": [ "public-read" ]
      }
    }
  ]
}
```

Granting Permission to an Anonymous User

The following example policy grants permissions to anonymous users.

Example

```
{
  "Version": "2008-10-17",
  "Statement": [{
    "Sid": "AddPerm",
    "Effect": "Allow",
    "Principal": {
      "AWS": "*"
    },
    "Action": [ "s3:GetObject" ],
    "Resource": [ "arn:aws:s3:::bucket/*" ]
  ]
}
```

Restricting Access to Specific IP Addresses

This statement grants permissions to any user to perform any S3 action 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 192.168.143.* range of allowed IP addresses with one exception, 192.168.143.188.

Note that the `IPAddress` and `NotIpAddress` values specified in the condition uses CIDR notation described in RFC 2632. For more information, go to <http://www.rfc-editor.org/rfc/rfc4632.txt>.

Example

```
{
  "Version": "2008-10-17",
  "Id": "S3PolicyId1",
  "Statement": [
    {
      "Sid": "IPAllow",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::bucket/*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": "192.168.143.0/24"
        },
        "NotIpAddress": {
          "aws:SourceIp": "192.168.143.188/32"
        }
      }
    }
  ]
}
```

Restricting Access to Specific HTTP Referer

The following example policy restricts access based on HTTP Referer.

Example

```
{
  "Version": "2008-10-17",
  "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:::example-bucket/*",
      "Condition": {
        "StringLike": {
          "aws:Referer": [
            "http://www.mysite.com/*",
            "http://mysite.com/*"
          ]
        }
      }
    }
  ]
}
```

Granting Permissions to Enable Log Delivery to an S3 Bucket

The following example policy enables log delivery to your Amazon S3 bucket. The account specified in the following policy is the Log Delivery group. You must use the ARN specified in this policy because it identifies the Log Delivery group. For more information, see [Setting Up Server Access Logging \(p. 208\)](#).

Example

```
{
  "Version": "2008-10-17",
  "Id": "LogPolicy",
  "Statement": [{
    "Sid": "Enables the log delivery group to publish logs to your bucket ",
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::858827067514:root"
    },
    "Action": [ "s3:GetBucketAcl",
      "s3:GetObjectAcl",
      "s3:PutObject"
    ],
    "Resource": [ "arn:aws:s3:::example-bucket",
      "arn:aws:s3:::example-bucket/*"
    ]
  }
]
```

Granting Permission, Using Canonical ID, to a CloudFront Origin Identify

The following example bucket policy grants a CloudFront Origin Identity permission to GET 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.

Example

```
{
  "Version": "2008-10-17",
  "Id": "PolicyForCloudFrontPrivateContent",
  "Statement": [{
    "Sid": "Grant a CloudFront Origin Identity access to support private content",

    "Effect": "Allow",
    "Principal": {
      "CanonicalUser": "79a59df900b949e55d96a1e698fbaced
fd6e09d98eacf8f8d5218e7cd47ef2be"
    },
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::example-bucket/*"
  }]
}
```

How to Use Resources, Principals, Operations, and Conditions in Bucket Policies

Topics

- [Specifying Amazon S3 Resources in Bucket Policies \(p. 147\)](#)
- [Specifying Principals in Bucket Policies \(p. 147\)](#)
- [Amazon S3 Actions \(p. 148\)](#)
- [Bucket Keys in Amazon S3 Policies \(p. 149\)](#)
- [Object Keys in Amazon S3 Policies \(p. 151\)](#)

Specifying Amazon S3 Resources in Bucket Policies

You can refer to buckets and objects in bucket policies. Amazon S3 policies use the Amazon Resource Name (ARN) format for specifying them, as follows:

```
arn:aws:s3:::[resourcename]
```

The resource name is the fully qualified name of a bucket or object that the user is requesting access to. For buckets, the resource name is `bucketname`, where *bucketname* is the name of the bucket. For objects, the format for the resource's name is `bucketname/keyname`, where *bucketname* is the name of the bucket and *keyname* is the full name of the object. For example, if you have a bucket called "Ooyala" and an object with the name `shared/developer/settings.conf`, the resource name for the bucket would be `Ooyala`; for the object it would be `Ooyala/shared/developer/settings.conf`.

Specifying Principals in Bucket Policies

The `Principal` is one or more people who receive or are denied permission according to the policy. You must specify the principal by using the principal's AWS account ID (e.g., 1234-5678-9012, with or without the hyphens). You can specify multiple principals, or a wildcard (*) to indicate all possible users. You can view your account ID by logging in to your AWS account at <http://aws.amazon.com> and clicking **Account Activity** in the **Accounts** tab.

Instead of specifying an AWS account ID you can specify a Canonical User ID when granting permission to an AWS Account. You can view your Canonical User ID by logging in to your AWS account at <http://aws.amazon.com> and, clicking **Security Credentials** in the **Accounts** tab. You can also grant a CloudFront Origin Access Identify using the Canonical User ID associated with that identify. To learn more about CloudFront's support for serving private content, go to [Serving Private Content](#) topic in Amazon CloudFront Developer Guide. You must specify the Canonical User ID for your CloudFront distribution's origin identity, not your AWS Account.

In JSON, you use "AWS" : as a prefix for the principal's AWS account ID and the "CanonicalUser" : prefix for the principal's AWS Canonical User ID.

Amazon S3 Actions

The following list shows the format for the Amazon S3 actions that you can reference in a policy.

Actions Related to Objects

- `s3:GetObject` (covers REST GET Object, REST HEAD Object, REST GET Object torrent, SOAP `GetObject`, and SOAP `GetObjectExtended`)
- `s3:GetObjectVersion` (covers REST GET Object, REST HEAD Object, REST GET Object torrent, SOAP `GetObject`, and SOAP `GetObjectExtended`)
- `s3:PutObject` (covers the REST PUT Object, REST POST Object, REST Initiate Multipart Upload, REST Upload Part, REST Complete Multipart Upload, SOAP `PutObject`, and SOAP `PutObjectInline`)
- `s3:GetObjectAcl`
- `s3:GetObjectVersionAcl`
- `s3:PutObjectAcl`
- `s3:PutObjectAclVersion`
- `s3:DeleteObject`
- `s3:DeleteObjectVersion`
- `s3:ListMultipartUploadParts`
- `s3:AbortMultipartUpload`

Actions Related to Buckets

- `s3:CreateBucket`
- `s3:DeleteBucket`
- `s3:ListBucket`
- `s3:ListBucketVersions`
- `s3:ListAllMyBuckets` (covers REST GET Service and SOAP `ListAllMyBuckets`)
- `s3:ListBucketMultipartUploads`

Actions Related to Bucket Sub-Resources

- `s3:GetBucketAcl`
- `s3:PutBucketAcl`
- `s3:GetBucketVersioning`
- `s3:PutBucketVersioning`
- `s3:GetBucketRequesterPays`
- `s3:PutBucketRequesterPays`
- `s3:GetBucketLocation`

- `s3:PutBucketPolicy`
- `s3:GetBucketPolicy`
- `s3:PutBucketNotification`
- `s3:GetBucketNotification`

Bucket Keys in Amazon S3 Policies

The following table shows the keys related to buckets that can be in Amazon S3 policies.

Action	Applicable Keys	Description
<code>s3:CreateBucket</code>	<code>s3:x-amz-acl</code>	The Amazon S3 canned ACL that is applied to the bucket. A canned ACL represents a predefined permission that can be applied to the bucket being created. Valid values: <code>private</code> <code>public-read</code> <code>public-read-write</code> <code>authenticated-read</code> <code>bucket-owner-read</code> <code>bucket-owner-full-control</code> <code>log-delivery-write</code> . Example value: <code>public-read</code>
	<code>s3:LocationConstraint</code>	Specifies the Region where the bucket will be created. Valid values are <code>us-west-1</code> (for Northern California) or <code>EU</code> (for Ireland). Do not specify a value for US Standard. Example value: <code>us-west-1</code>
<code>s3:ListBucket</code>	<code>s3:prefix</code>	Limits the response to objects that begin with the specified prefix. Use this to allow or deny access to objects that begin with the prefix. Example value: <code>home</code>
	<code>s3:delimiter</code>	The character you use to group objects. Example value: <code>/</code>
	<code>s3:max-keys</code>	The number of objects to return from the call. The maximum allowed value (and default) is 1000. For use with access policy language numeric conditions (for more information, see Numeric Conditions (p. 229)). Example value: <code>100</code>

Amazon Simple Storage Service Developer Guide
How to Use Resources, Principals, Operations, and
Conditions in Bucket Policies

Action	Applicable Keys	Description
s3:ListBucketVersions	s3:prefix	Header that lets you limit the response to include only keys that begin with the specified prefix. Example value: home
	s3:delimiter	The character you use to group objects. Example value: /
	s3:max-keys	The number of objects to return from the call. The maximum allowed value (and default) is 1000. For use with access policy language numeric conditions (for more information, see Numeric Conditions (p. 229)). Example value: 100
s3:PutBucketAcl	s3:x-amz-acl	The Amazon S3 canned ACL that is applied to the bucket. A canned ACL represents a predefined permission that can be applied to the bucket being created. Valid values: private public-read public-read-write authenticated-read bucket-owner-read bucket-owner-full-control log-delivery-write. Example value: public-read

Object Keys in Amazon S3 Policies

The following list shows the keys related to objects that can be in Amazon S3 policies.

Action	Applicable Keys	Description
s3:PutObject	s3:x-amz-acl	The Amazon S3 canned ACL that is applied to the object. A canned ACL represents a predefined permission that can be applied to the object being PUT to Amazon S3. Valid values: private public-read public-read-write authenticated-read bucket-owner-read bucket-owner-full-control log-delivery-write. Example value: public-read
	s3:x-amz-copy-source	Used when copying an object. Header that specifies the name of the source bucket and key name of the source object, separated by a slash (/). Example value: /bucketname/keyname
	s3:x-amz-metadata-directive	Used when copying an object. Header that specifies whether the metadata is copied from the source object or replaced with metadata provided in the request. If copied, the metadata, except for the version ID, remains unchanged. Otherwise, all original metadata is replaced by the metadata you specify. Valid values: COPY REPLACE. The default is COPY. Example value: REPLACE
s3:PutObjectAcl	s3:x-amz-acl	The Amazon S3 canned ACL that is applied to the object. A canned ACL represents a predefined permission that can be applied to the object being PUT to Amazon S3. Valid values: private public-read public-read-write authenticated-read bucket-owner-read bucket-owner-full-control log-delivery-write. Example value: public-read

Action	Applicable Keys	Description
s3:GetObjectVersion	s3:VersionId	The version ID of the object being retrieved. Example value: Upfdndhfd8438MNFDN93 jdnJFkdmqnh893
s3:GetObjectVersionAcl	s3:VersionId	The version ID of the object ACL being retrieved. Example value: Upfdndhfd8438MNFDN93 jdnJFkdmqnh893
s3:PutObjectVersionAcl	s3:VersionId	The version ID of the object ACL being PUT. Example value: Upfdndhfd8438MNFDN93 jdnJFkdmqnh893
	s3:x-amz-acl	The Amazon S3 canned ACL that is applied to the object. A canned ACL represents a predefined permission that can be applied to the object being PUT to Amazon S3. Valid values: private public-read public-read-write authenticated-read bucket-owner-read bucket-owner-full-control log-delivery-write. Example value: public-read
s3:DeleteObjectVersion	s3:VersionId	The version ID of the object being deleted. Example value: Upfdndhfd8438MNFDN93 jdnJFkdmqnh893

Using ACLs

An ACL can contain up to 100 grants. If no ACL is provided when a bucket is created or an object written, a default ACL is created. The default ACL consists of a single grant that gives the owner (i.e., the creator) the FULL_CONTROL permission. If you overwrite an existing object, the ACL for the existing object is overwritten and will default to FULL_CONTROL for the owner if no ACL is specified.

You can change the ACL of a resource without changing the resource itself. However, like Amazon S3 objects, there is no way to modify an existing ACL—you can only overwrite it with a new version. Therefore, to modify an ACL, read the ACL from Amazon S3, modify it locally, and write the entire updated ACL back to Amazon S3.



Note

The method of reading and writing ACLs differs depending on which API you are using. For more information, see the API-specific documentation for details.

Regardless of which API you are using, the XML representation of an ACL stored in Amazon S3 (and returned when the ACL is read) is the same. In the following example ACL, the owner has the default FULL_CONTROL, the "Frank" and "Jose" users both have WRITE and READ_ACP permissions, and all users have permission to READ.

```
<AccessControlPolicy>
  <Owner>
    <ID>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666e9</ID>
    <DisplayName>chriscustomer</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666e9</ID>

        <DisplayName>chriscustomer</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be</ID>

        <DisplayName>Frank</DisplayName>
      </Grantee>
      <Permission>WRITE</Permission>
    </Grant>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be</ID>

        <DisplayName>Frank</DisplayName>
      </Grantee>
      <Permission>READ_ACP</Permission>
    </Grant>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>e019164ebb0724ff67188e243eae9ccbebdde523717cc312255d9a82498e394a</ID>

        <DisplayName>Jose</DisplayName>
      </Grantee>
      <Permission>WRITE</Permission>
    </Grant>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>e019164ebb0724ff67188e243eae9ccbeb
dde523717cc312255d9a82498e394a</ID>
```

```
<DisplayName>Jose</DisplayName>
</Grantee>
<Permission>READ_ACP</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>
</AccessControlList>
</AccessControlPolicy>
```



Note

When you write an ACL to Amazon S3 that AmazonCustomerByEmail grantees, they will be converted to the CanonicalUser type prior to committing the ACL.

Using ACLs and Bucket Policies Together

When you have ACLs and bucket policies assigned to buckets, Amazon S3 evaluates the existing Amazon S3 ACLs as well as the bucket policy when determining an account's access permissions to an Amazon S3 resource. If an account has access to resources that an ACL or policy specifies, they are able to access the requested resource.

With existing Amazon S3 ACLs, a grant always provides access to a bucket or object. When using policies, a deny always overrides a grant.



Note

Bucket policies have their own set of rules when prioritizing grants and denies. For more information, see [Evaluation Logic \(p. 219\)](#).

You can migrate existing Amazon S3 ACLs to policies.

To migrate ACLs to bucket policies

1	Associate a policy with a specific user, group, or bucket.
2	Add a grant to the policy for each Amazon S3 resource that the user or group has been granted access to in the ACL

Once completed, you can begin to manage your account permissions with policies instead of ACLs.

Relationship Between Actions and Permissions

Policies can allow or deny certain actions. ACLs can grant certain permissions. The actions allowed or denied by policies are a superset of the permissions that you can grant by ACLs. The relationship between actions and permissions is summarized in the following sections.

Object ACL Permissions

- **READ**—Granting *READ* permission in an object ACL allows the *s3:GetObject*, *s3:GetObjectVersion*, and *s3:GetObjectTorrent* actions to be performed on that object.
- **READ_ACP**—Granting *READ_ACP* permission in an object ACL allows the *s3:GetObjectAcl* and *s3:GetObjectVersionAcl* actions to be performed on that object.
- **WRITE_ACP**—Granting *WRITE_ACP* permission in an object ACL allows the *s3:PutObjectAcl* and *s3:PutObjectVersionAcl* actions to be performed on that object.
- **FULL_CONTROL**—Granting *FULL_CONTROL* permission in an object ACL is equivalent to granting *READ*, *READ_ACP*, and *WRITE_ACP* permission.

Bucket ACL Permissions

- **READ**—Granting *READ* permission in a bucket ACL allows the *s3:ListBucket*, *s3:ListBucketVersions*, and *s3:ListBucketMultipartUploads* actions to be performed on that bucket.
- **WRITE**—Granting *WRITE* permission in a bucket ACL allows the *s3:PutObject* and *s3:DeleteObject* actions to be performed on any object in that bucket. 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.
- **READ_ACP**—Granting *READ_ACP* permission in a bucket ACL allows the *s3:GetBucketAcl* action to be performed on that bucket.
- **WRITE_ACP**—Granting *WRITE_ACP permission* in a bucket ACL allows the *s3:PutBucketAcl* action to be performed on that bucket.
- **FULL_CONTROL**—Granting *FULL_CONTROL* permission in a bucket ACL is equivalent to granting *READ*, *WRITE*, *READ_ACP*, and *WRITE_ACP* permission.

For more information on the actions that can be allowed or denied by policies, see [Writing Bucket Policies](#) (p. 142).

Using Query String Authentication

Query string authentication is useful for giving HTTP or browser access to resources that would normally require authentication. The signature in the query string secures the request. Query string authentication requests require an expiration date. You can specify any future expiration time in epoch or UNIX time (number of seconds since January 1, 1970).

Using Query String Authentication

1	Create a query.
2	Specify an expiration time for the query.
3	Sign it with your signature.
4	Place the data in an HTTP request.
5	Distribute the request to a user or embed the request in a web page

For example, a query URL is similar to the following example.

```
http://quotes.s3.amazonaws.com/nelson?AWSAccessKeyId=44CF9SAMPLEF252F707&Expires=1177363698&Signature=vjSAMPLENmGa%2ByT272YEAiv4%3D
```

For information on how to sign requests, see [Query String Request Authentication Alternative \(p. 36\)](#)

Data Protection

Topics

- [Using Reduced Redundancy Storage \(p. 157\)](#)
- [Using Versioning \(p. 159\)](#)

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 ensure durability, Amazon S3 `PUT` and `PUT Object copy` operations synchronously store your data across multiple facilities before returning `SUCCESS`. Once stored, Amazon S3 maintains the durability of your objects by quickly detecting and repairing any lost redundancy.

Amazon S3 also regularly verifies the integrity of data stored using checksums. If Amazon S3 detects 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 99.999999999% 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.

Using Reduced Redundancy Storage

The Amazon S3 Reduced Redundancy Storage (RRS) option provides a less durable, highly available storage option that is designed to provide 99.99% durability of objects over a given year.

Amazon S3 stores objects according to their storage class, which Amazon S3 assigns to an object when it is written to Amazon S3. The default storage class is `STANDARD`. You can assign objects a specific storage class (`STANDARD` or `REDUCED_REDUNDANCY`) only when writing the objects or when copying objects stored in Amazon S3. RRS is specified per object, at the time the object is written."

For a general overview of this feature, see [Reduced Redundancy Storage \(p. 10\)](#).

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 15B4D3461F177624206A: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

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 0223123223RW0EXG2:0RQf4/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 02236Q123123EXG2:0RQf4/cRonhpaBX5sCYVf1bNRuU=
x-amz-storage-class: STANDARD
x-amz-metadata-directive: COPY
```



Note

If you copy an RSS 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 \(p. 159\)](#).

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

Topics

- [Enabling a Bucket's Versioning State \(p. 160\)](#)
- [Adding Objects to Versioning-Enabled Buckets \(p. 164\)](#)
- [Listing the Objects in a Versioning-Enabled Bucket \(p. 165\)](#)
- [Retrieving Object Versions \(p. 166\)](#)
- [Deleting Object Versions \(p. 168\)](#)
- [Restoring Previous Versions \(p. 173\)](#)

- [Versioned Object Permissions and ACLs \(p. 174\)](#)
- [Working with Versioning-Suspended Buckets \(p. 175\)](#)

Versioning is a means of keeping multiple variants of an object in the same bucket. 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).



You might enable versioning to prevent objects from being deleted or overwritten by mistake, or to archive objects so that you can retrieve previous versions of them.



Note

The SOAP API does not support versioning.

Enabling a Bucket's Versioning State

Topics

- [MFA Delete \(p. 160\)](#)
- [Enabling Versioning \(p. 161\)](#)
- [Suspending Versioning \(p. 162\)](#)
- [Determining the Versioning State \(p. 163\)](#)

Buckets can be in one of three states: unversioned (the default), versioning-enabled, or versioning-suspended. Once you version enable a bucket, it can never return to an unversioned state. You can, however, suspend versioning on that bucket.

Only the bucket owner can configure the versioning state of a 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.

MFA Delete

You can add another layer of security by configuring a bucket to enable MFA (Multi-Factor Authentication) Delete. By enabling MFA Delete on your Amazon S3 bucket, you can only change the versioning state of your bucket or permanently delete an object version when you provide two forms of authentication together:

- Your AWS account credentials
- The concatenation of a valid serial number, a space, and the six-digit code displayed on an approved authentication device

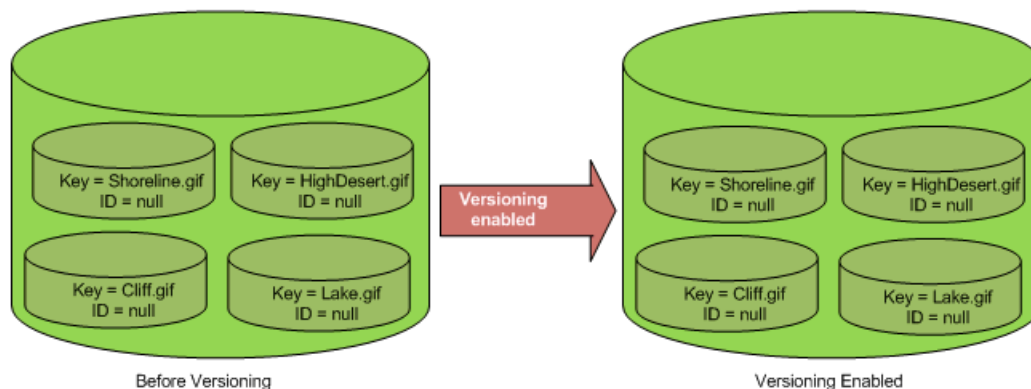
To use MFA Delete, you must purchase a third-party device that automatically updates the authentication code displayed on it, as shown.



For more information on how to purchase and activate an authentication device, go to <http://aws.amazon.com/mfa/>.

Enabling Versioning

Objects stored in your bucket before you set the versioning state have a version ID of `null`. When you enable versioning the objects in your bucket do not change, as shown in the following figure.



What changes is how Amazon S3 handles the objects in future requests in that bucket.

To enable object versioning

1. In a `PUT Bucket` request, include the `versioning` sub-resource.
2. In the body of the request, use `Enabled` for the value of the `Status` request element.

Example Enabling Versioning

The following request enables versioning on the bucket, `bucketName`.

```
PUT /?versioning HTTP/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
Content-Length: 124

<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Enabled</Status>
</VersioningConfiguration>
```

Configuring a Bucket with MFA Delete

You have the option of enabling MFA Delete at the same time you specify the versioning state of the bucket. Once you configure a bucket so that it is MFA Delete enabled, all future requests to change the versioning state or delete a version require the request header `x-amz-mfa: [SerialNumber] [AuthenticationCode]`. Note the space between `[SerialNumber]` and `[AuthenticationCode]`. Requests that include `x-amz-mfa` must use HTTPS.

Example Enabling Versioning and MFA Delete

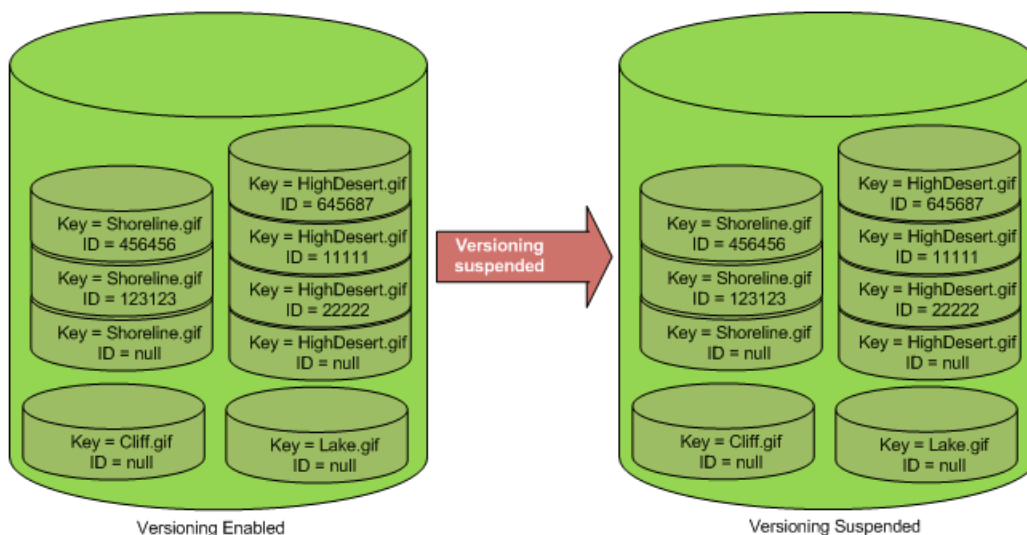
The following request enables versioning and MFA Delete on *bucketName*.

```
PUT /?versioning HTTPS/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
Content-Length: 124
x-amz-mfa: 20899872 301749

<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Enabled</Status>
  <MfaDelete>Enabled</MfaDelete>
</VersioningConfiguration>
```

Suspending Versioning

The bucket owner can suspend versioning to stop accruing object versions. When you suspend versioning, the objects in your bucket do not change, as shown in the following figure.



What changes is how Amazon S3 handles objects in future requests. For a more detailed explanation of the effects of suspending versioning, see [Working with Versioning-Suspended Buckets \(p. 175\)](#).

To suspend object versioning

1. In the header of a `PUT Bucket` request, include the `versioning` sub-resource.
2. In the body of the request, use `Suspended` for the value of the `Status` request element.

Example Suspending Versioning

The following request suspends versioning on *bucketName*.

```
PUT /?versioning HTTP/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
Content-Length: 124

<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Suspended</Status>
</VersioningConfiguration>
```

If you configured a bucket to be MFA Delete enabled, you must include the *x-amz-mfa* request header and the *MfaDelete* request element in the request to change the bucket's versioning state.

Example Suspending Versioning Using MFA Delete

The following request suspends versioning on *bucketName* that is configured with MFA Delete.

```
PUT /?versioning HTTPS/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
x-amz-mfa: 20899872 301749
Content-Type: text/plain
Content-Length: 124

<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Suspended</Status>
  <MfaDelete>Enabled</MfaDelete>
</VersioningConfiguration>
```

Note that requests that include *x-amz-mfa* must use HTTPS.

Determining the Versioning State

You can use the *versioning* sub-resource to retrieve the versioning state of a bucket.

To retrieve the versioning state of a bucket

1. In the header of a GET *Bucket* request, include the *versioning* sub-resource.

```
GET /?versioning HTTP/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
```

2. In the response, check the value of the response element *Status*. There are three possible results:
 - If Versioning on a bucket is enabled, the response is:

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Enabled</Status>
</VersioningConfiguration>
```

- If Versioning on a bucket is suspended, the response is:

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Suspended</Status>
</VersioningConfiguration>
```

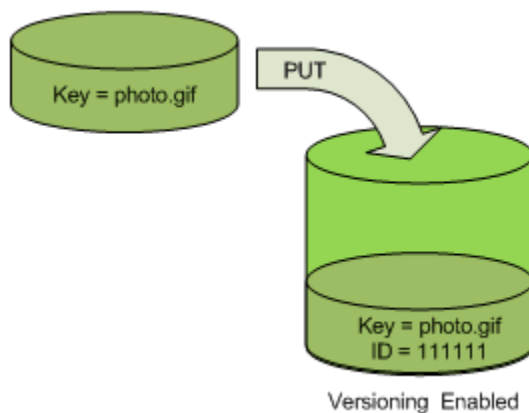
- If Versioning on a bucket has never been enabled, the response doesn't retrieve a *status* element:

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

Adding Objects to Versioning-Enabled Buckets

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.



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-version-id` 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.

Listing the Objects in a Versioning-Enabled Bucket

To list all of the versions of all of the objects in a bucket, you use the *versions* sub-resource in a GET Bucket request. Amazon S3 can only retrieve 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 02236Q3V0WHVSRW0EXG2:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

Retrieving a Subset of Objects in Buckets

Topics

- [Retrieving All Versions of a Key \(p. 165\)](#)
- [Retrieving Additional Object Versions after Exceeding Max-Keys \(p. 166\)](#)

There might be times when you either want to retrieve a subset of all object versions in a bucket, or the number of object versions 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 must use the request parameters for GET Bucket. For more information, go to [GET Bucket](#).

Retrieving All Versions of a Key

You can retrieve all versions of an object using the *versions* sub-resource 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> sub-resource and <i>prefix</i> . GET /?versions&prefix=objectName HTTP/1.1

Example Retrieving Objects Using 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 02236Q3V0WHVSRW0EXG2:0RQf4/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](#).

Retrieving Additional Object Versions after Exceeding Max-Keys

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

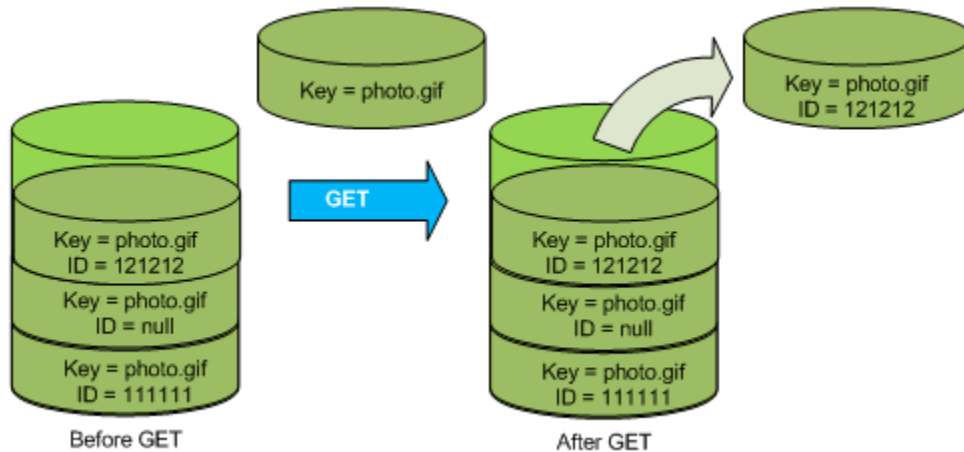
1	Set the value of <code>key-marker</code> to the key returned in <code>NextKeyMarker</code> in the previous response.
2	Set the value of <code>version-id-marker</code> to the version ID returned in <code>NextVersionIdMarker</code> in the previous response.
3	Send a <code>GET Bucket versions</code> request using <code>key-marker</code> and <code>version-id-marker</code> .

Example Retrieving Objects Starting with a Specified Key and Version ID

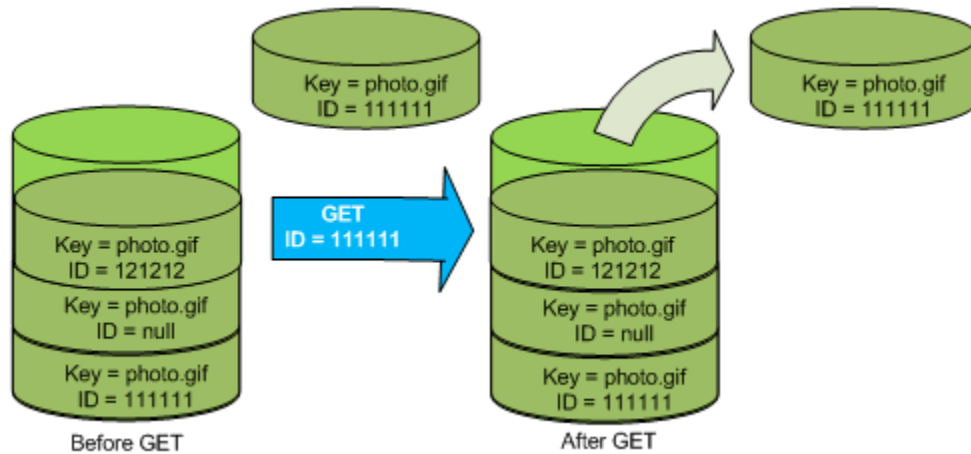
```
GET /?versions&key-marker=myObject&version-id-marker=298459348571 HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS 02236Q3V0WHVSRW0EXG2:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

Retrieving Object Versions

A simple `GET` request retrieves the latest version of an object. The following figure shows how `GET` returns the latest 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 latest).



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 02236Q3V0WHVSRW0EXG2:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

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 3HL4kqCxf3vjVBH40NrjfkD of `my-image.jpg`.

```
HEAD /my-image.jpg?versionId=3HL4kqCxf3vjVBH40NrjfkD HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS 02236Q3V0WpaBX5sCYVf1bNRuU=
```

The following shows a sample response.

```
HTTP/1.1 200 OK
x-amz-id-2: ef8yU9ASled4OpIszj7UDNEHGran
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

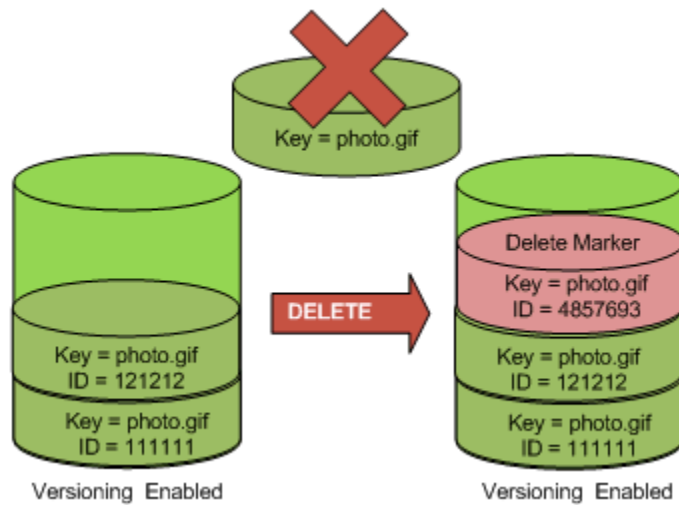
Topics

- [Using MFA Delete \(p. 170\)](#)
- [Working With Delete Markers \(p. 170\)](#)
- [Removing Delete Markers \(p. 171\)](#)

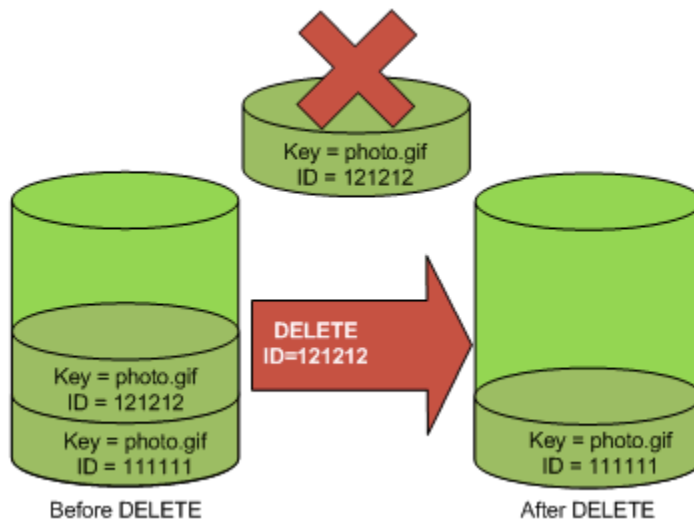
A `DELETE` request has the following use cases:

- When versioning is enabled, a simple `DELETE` cannot permanently delete an object. Instead, Amazon S3 inserts a delete marker in the bucket and that becomes the latest version of the object with a new ID. When you try to `GET` an object whose latest 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.
- To permanently delete versioned objects, you must use `DELETE Object versionId`.

The following figure shows that a simple `DELETE` does not actually remove the specified object. Instead, Amazon S3 inserts a delete marker.



The following figure shows that deleting a specified object version permanently removes that object.



To delete a specific version of an object

- In a `DELETE` request 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 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
Content-Length: 0
```

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. 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.

Requests that include `x-amz-mfa` must use HTTPS.

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.

```
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 02236Q3V0WpaBX5sCYVf1bNRuU=
```

Note the space between `[SerialNumber]` and `[AuthenticationCode]`. For more information, go to [DELETE Object](#).

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 is like any other object (it has a key and version ID) except that it:

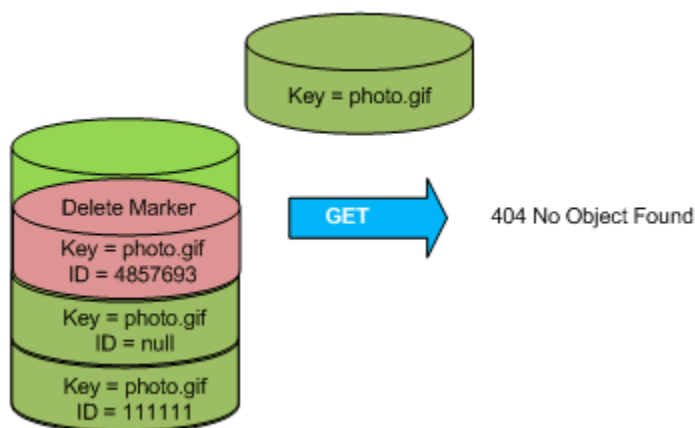
- Does not have data associated with it
- Is not associated with an access control list (ACL) value
- Does not retrieve anything from a `GET` request because it has no data; you get a 404 error
- The only operation you can use on a delete marker is `DELETE` and only the bucket owner can issue such a request

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 latest version of the object. (The object's key becomes the key of the Delete Marker.) If you try to get an object and its latest version is a delete marker, Amazon S3 responds with:

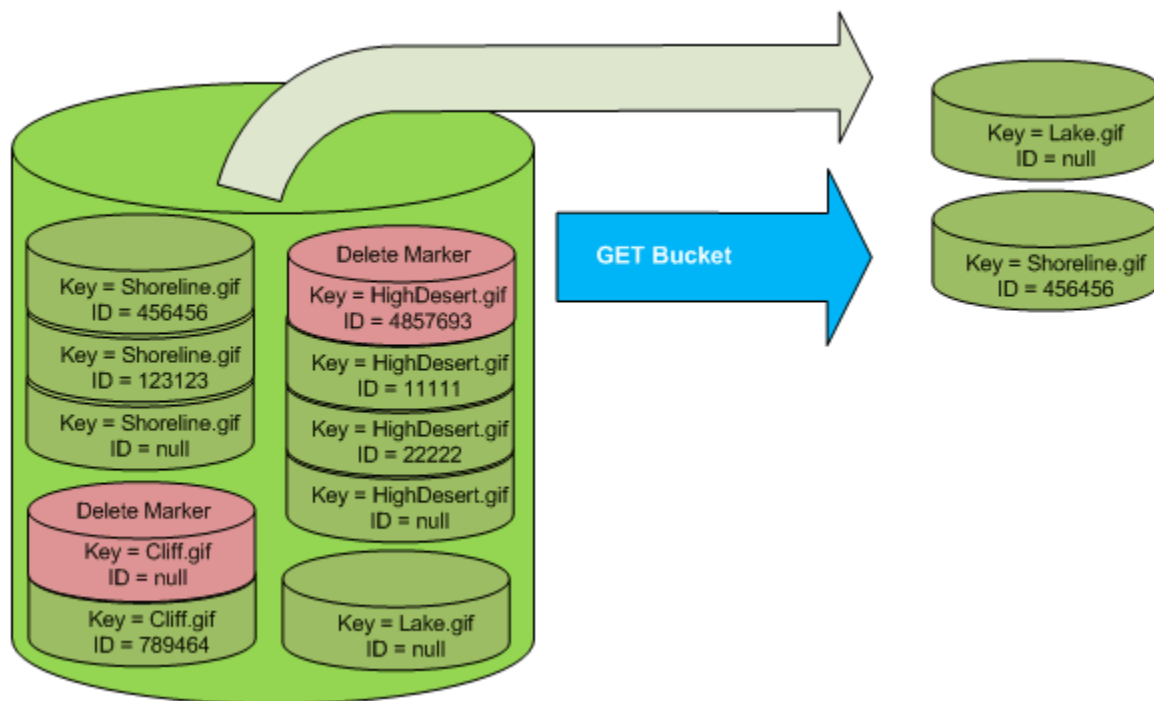
- A 404 (Object not found) error
- A response header, `x-amz-delete-marker: true`

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 latest version is a delete marker, returns a 404 No Object Found error.



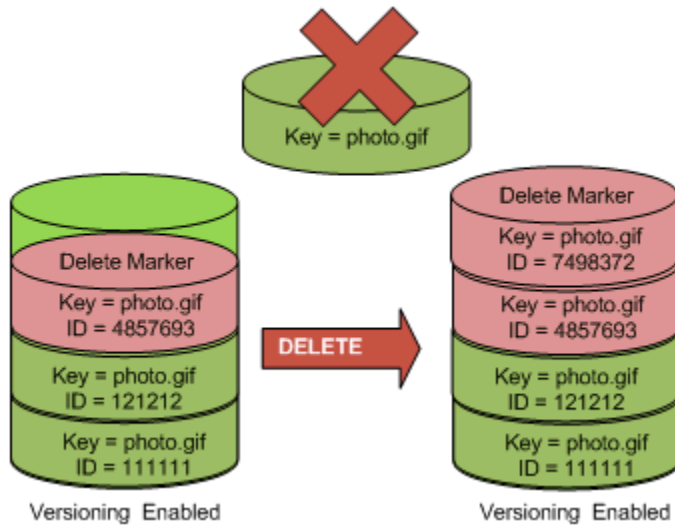
The only way to list delete markers (and other versions of an object) is by using the *versions* sub-resource in a `GET Bucket versions` request. A simple `GET` does not retrieve delete marker objects. The following figure shows that a `GET Bucket` request does not return objects whose latest version is a delete marker.



Removing Delete Markers

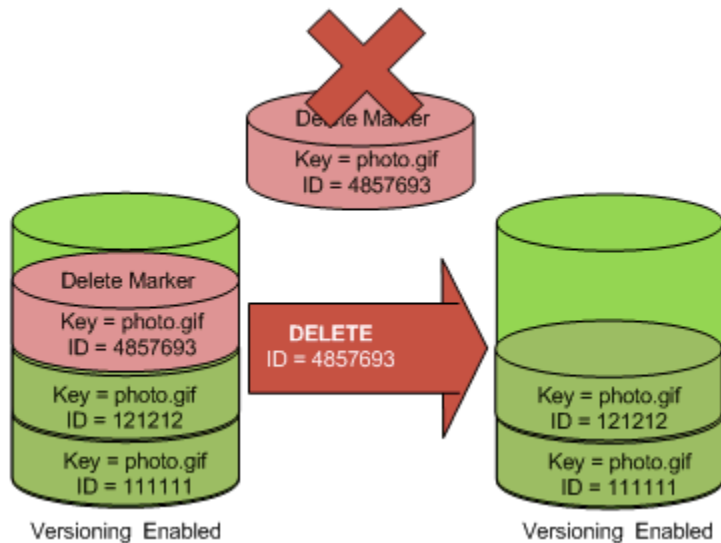
To delete a delete marker, you must specify its version ID in a `DELETE Object versionId` request. If you use `DELETE` to delete a delete marker (without specifying the version ID of the delete marker), Amazon S3 does not delete the delete marker, but instead, inserts another delete marker.

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 latest 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 02236Q3V0WHVSRW0EXG2:0RQf4/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
```



Note

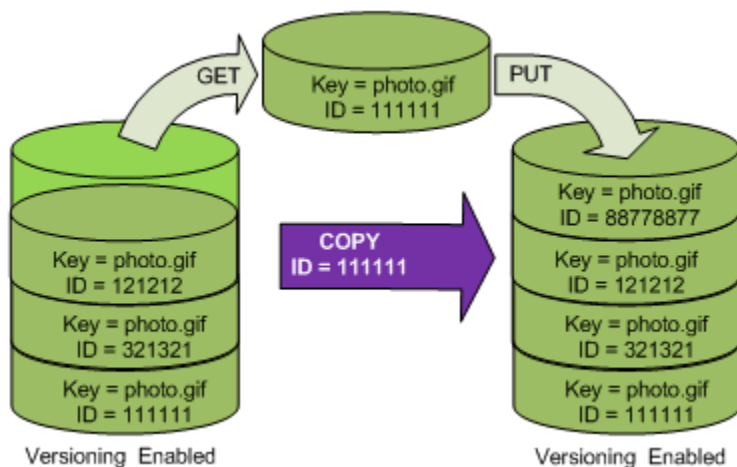
Delete markers accrue a nominal charge for storage in Amazon S3. Delete markers are equal to the size of your key name, e.g., a Delete Marker for the key "photo.gif" adds 9 bytes of storage to your bucket.

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:

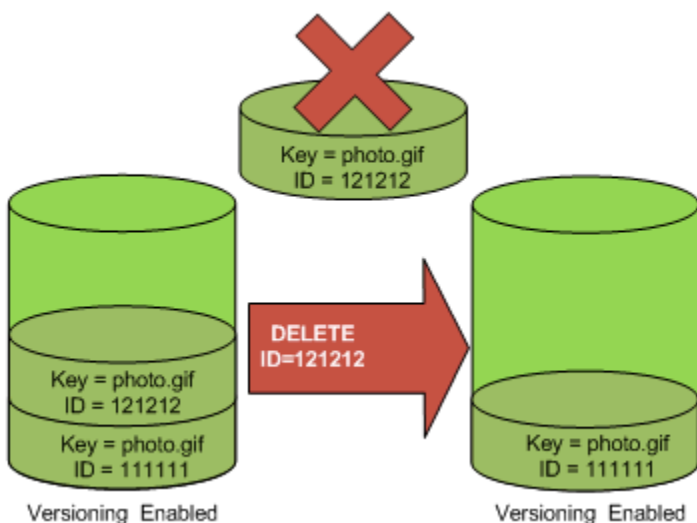
- Copy a previous version of the object into the same bucket
The copied object becomes the latest version of that object and all object versions are preserved.
- Permanently delete the latest version of the object
When you delete the latest object version you, in effect, turn the previous version into the latest version of that object.

Because all object versions are preserved, you can make any earlier version the latest 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 latest 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 latest version (121212) of an object, which leaves the previous version (111111) as the latest object.



A subsequent `GET` will retrieve version 111111.

Versioned Object Permissions and ACLs

Permissions are set at the version level. Each version has its own object owner—whoever `PUT` the version. 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 [Authentication and Access Control](#) (p. 141).

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 02236Q3V0WHVSRW0EXG2:0RQf4/cRonhpaBX5sCYVf1bNRuU=
Content-Length: 124

<AccessControlPolicy>
  <Owner>
    <ID>8a6925ce4adf5f21c32aa379004fef</ID>
    <DisplayName>mtd@amazon.com</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>8a6925ce4adf588a4532142d3f74dd8c71fa124b1ddee97f21c32aa379004fef</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 latest 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 02236Q3V0WHVSRW0EXG2:0RQf4/cRonhpaBX5sCYVf1bNRuU
```

For more information, go to [GET Object acl](#).

Working with Versioning-Suspended Buckets

Topics

- [Adding Objects to Versioning-Suspended Buckets \(p. 176\)](#)
- [Retrieving Objects from Versioning-Suspended Buckets \(p. 177\)](#)
- [Deleting Objects from Versioning-Suspended Buckets \(p. 178\)](#)

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.



Note

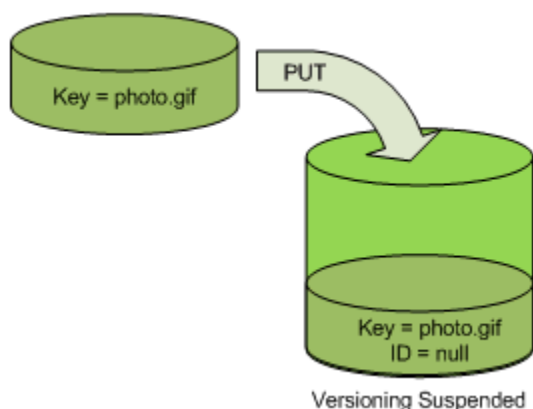
It is possible to have versioned objects in a versioning-suspended bucket if the objects were added when the bucket was versioning-enabled.

The following sections describe the behavior of buckets and objects when versioning is suspended.

Adding Objects to Versioning-Suspended Buckets

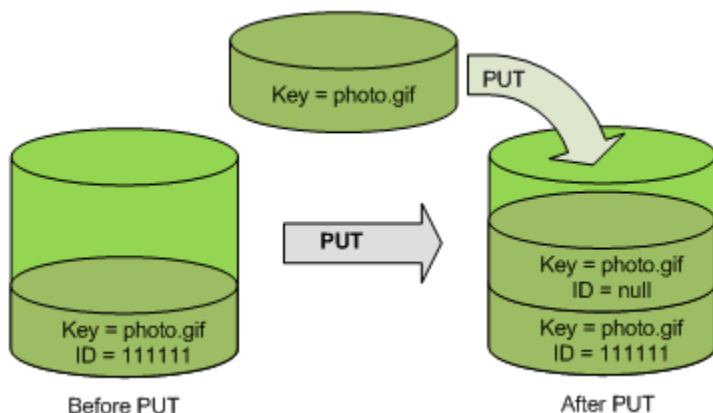
Once you suspend versioning on a bucket, Amazon S3 automatically adds a `null` version ID to every subsequent object stored thereafter (using `PUT`, `POST`, or `COPY`) in that bucket.

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.

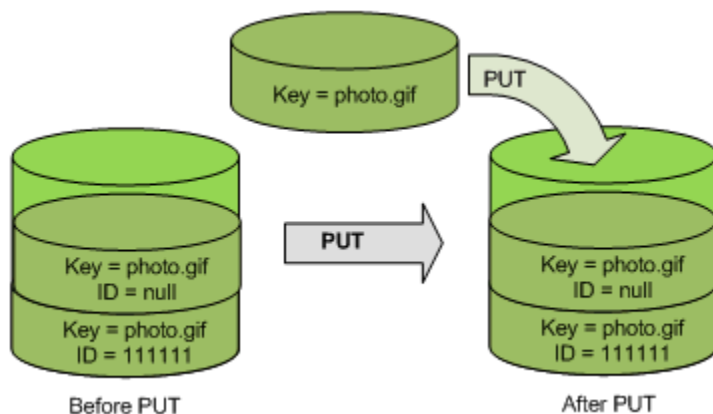


If a null version is already in the bucket and you add another object with the same key, the added object overwrites the original null version.

If there are versioned objects in the bucket, the version you `PUT` becomes the latest 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.



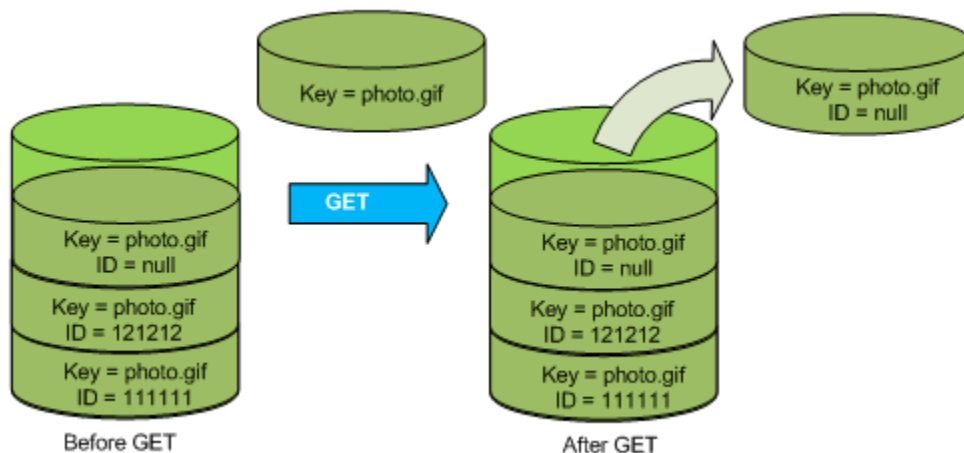
Note that although the key and version ID (`null`) of null version are the same before and after the `PUT`, the contents of the null version originally stored in the bucket is replaced by the contents of the object `PUT` into the bucket.

Adding Objects to Version-Suspended Buckets

- | | |
|---|--|
| 1 | Suspend 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. |

Retrieving Objects from Versioning-Suspended Buckets

A `GET Object` request returns the latest version of an object whether you've enabled versioning on a bucket or not. The following figure shows how a simple `GET` returns the latest version of an object.



Example Retrieving the Latest Version

The following request retrieves the latest version of `photo.gif`.

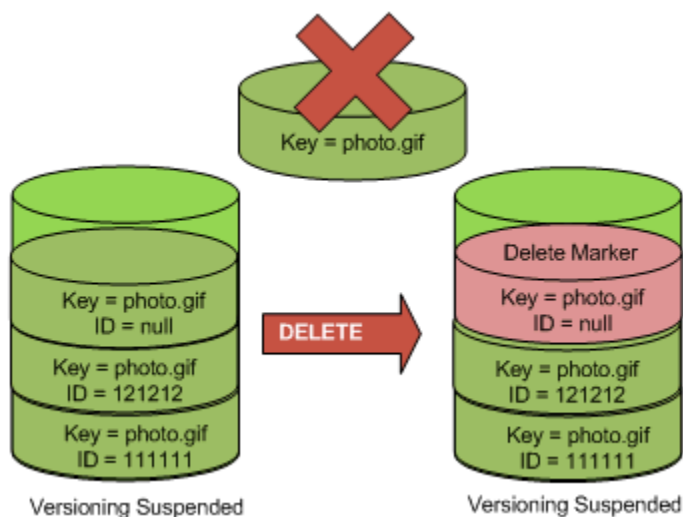
```
GET /photo.gif HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS 02236Q3V0WHVSRW0EXG2:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

Deleting Objects from Versioning-Suspended Buckets

If versioning is suspended, a `DELETE` request:

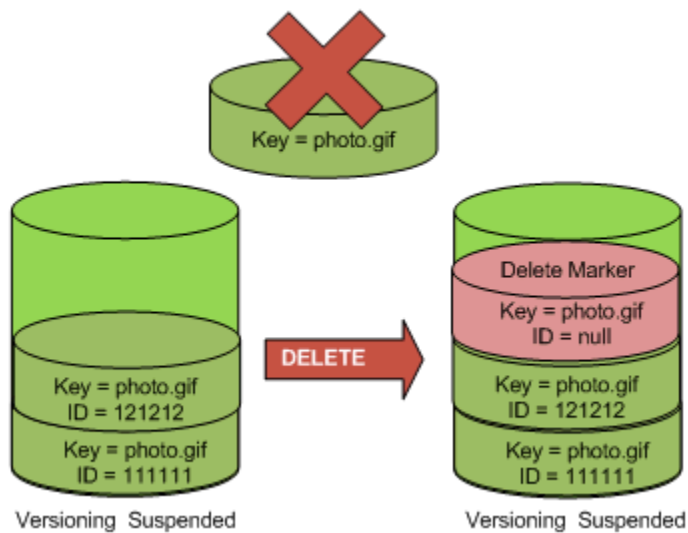
- Can only remove an object whose version ID is `null`
Doesn't remove anything if there isn't a null version of the object in the bucket.
- Inserts a delete marker into the bucket

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`.

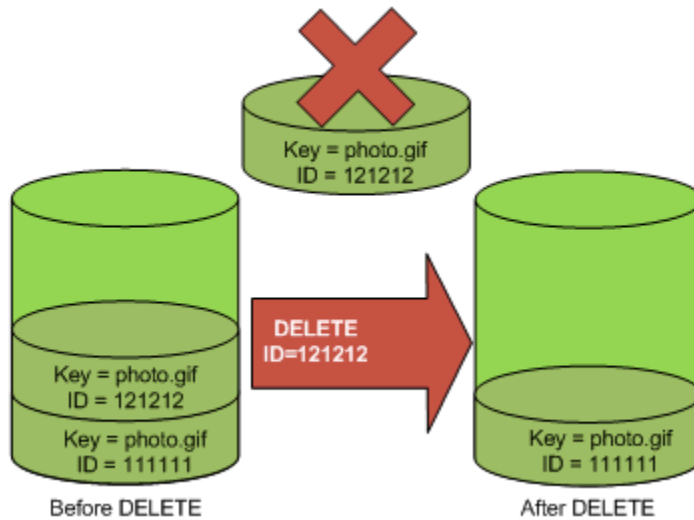


Remember that a delete marker doesn't have content so you lose the content of the null version when a delete marker replaces it.

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.



Setting Up Notification of Bucket Events

The Amazon S3 notifications feature enables you to configure a bucket so that Amazon S3 publishes a message to an Amazon Simple Notification Service (SNS) topic when Amazon S3 detects a key event on a bucket. Subscribers to this topic can have messages for bucket events delivered to an endpoint such as a web server, e-mail address, or an Amazon Simple Queue Service queue.

Note that Notification fees will be charged at the normal Amazon SNS rates. For details about pricing, go to <http://aws.amazon.com/sns/#pricing>. There are no additional Amazon S3 charges for publishing a message to an Amazon SNS topic beyond the request fee to enable and disable the feature. In order to receive notifications of a bucket's events, you must subscribe to the topic that has been configured for that bucket.

Currently, the `s3:ReducedRedundancyLostObject` event is the only event supported by Amazon S3. The `s3:ReducedRedundancyLostObject` event is triggered when Amazon S3 detects that it has lost all replicas of a Reduced Redundancy Storage (RRS) object and can no longer service requests for that object. Notification on this event can enable you to replace missing RRS objects proactively to ensure there is no customer impact.

Notifications are published to a specified Amazon SNS topic. Currently, only one topic can be configured for notifications. If the bucket owner and Amazon SNS topic owner are the same, the bucket owner has permission to publish notifications to the topic by default. Otherwise, the owner of the topic must create a policy to enable the bucket owner to publish to the topic. For more information about creating this policy, see [Example Cases for Amazon SNS Access Control](#).



Important

The Amazon SNS topic specified for a bucket's notification configuration must be in the same region as the bucket.

The following table lists the topic regions for the possible `LocationConstraint` values for buckets.

LocationConstraint	Topic Region
Empty string	us-east-1
EU	eu-west-1

LocationConstraint	Topic Region
us-west-1	us-west-1
ap-southeast-1	ap-southeast-1

By default, only the bucket owner has permission to set and view the notifications on a bucket. However, the bucket owner can use a bucket policy to grant permission to other users to set this configuration using the `s3:PutBucketNotification` permission or view this configuration via the `s3:GetBucketNotification` permission. For more information about bucket policy, see [Using Bucket Policies](#) (p. 141).

Similar to other bucket settings, there is a sub-resource for each bucket enabling you to set or view that bucket's notification settings. To configure notifications on a bucket, use the bucket's `notification` sub-resource with the Amazon S3 REST API. For more information about the `notification` sub-resource, see [Put Bucket notification](#) and [Get Bucket notification](#).

To set notifications for a bucket

- Use the `PUT` operation with a request body containing a `NotificationConfiguration` element that specifies the Amazon Resource Name (ARN) of the Amazon SNS topic, where the notification will be published, and what events will be reported.

```
PUT ?notification HTTP/1.1
Host: quotes.s3.amazonaws.com
Date: Wed, 02 Jun 2010 12:00:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
<NotificationConfiguration>
  <TopicConfiguration>
    <Topic>arn:aws:sns:us-east-1:123456789012:myTopic</Topic>
    <Event>s3:ReducedRedundancyLostObject</Event>
  </TopicConfiguration>
</NotificationConfiguration>
```

After you call the `PUT` operation to configure notifications on a bucket, Amazon S3 publishes a test notification to ensure that the topic exists and that the bucket owner has permission to publish to the specified topic. If the notification is successfully published to the SNS topic, the `PUT` operation updates the bucket configuration and returns a 200 OK response with a `x-amz-sns-test-message-id` header containing the message ID of the test notification sent to the topic. For information about `PUT` operation errors, see [Put Bucket notification](#).

The message published for an event to a topic contains the fields listed in the following table.

Name	Description
Service	Amazon S3.
Event	Currently, the only supported event is <code>s3:ReducedRedundancyLostObject</code> .
Time	The time that the event was triggered.
Bucket	The name of the bucket.

Name	Description
Key	The object name.
VersionId	If versioning is enabled, the version ID. If versioning is not enabled, empty string.
RequestId	A unique ID that identifies which Amazon S3 operation published the notification.
HostID	A unique ID that identifies which host sent the message.

The following message is an example of a message publishing an *s3:ReducedRedundancyLostObject* event:

```
{
  "Service": "Amazon S3",
  "Event": "s3:ReducedRedundancyLostObject",
  "Time": "2010-02-11T23:48:22.000Z",
  "Bucket": "myphotos",
  "Key": "Home/2009/10/carvingpumpkins.jpg",
  "VersionId": "qfXHy5689N7n9mWVwanN_hIroMn_rzXl",
  "RequestId": "4442587FB7D0A2F9",
  "HostId": "fHZOHZ+oQlKAFQ7RgVSkIvcDNYI0v05gsOWWeJcyTTXWgGzI6CC5FZVICjD9bUzT"
}
```

The following message is an example of the test message that Amazon S3 sends when a bucket is enabled for notifications:

```
{
  "Service": "Amazon S3",
  "Event": "s3:TestEvent",
  "Time": "2010-02-11T23:48:22.000Z",
  "Bucket": "myphotos",
  "RequestId": "4442587FB7D0A2F9",
  "HostId": "fHZOHZ+oQlKAFQ7RgVSkIvcDNYI0v05gsOWWeJcyTTXWgGzI6CC5FZVICjD9bUzT"
}
```

To read the notification configuration for a bucket

- Use the `GET` operation, which returns a *NotificationConfiguration* element containing the Amazon SNS topic and events set for notification on the bucket.

```
GET ?notification HTTP/1.1
Host: quotes.s3.amazonaws.com
Date: Wed, 09 June 2010 12:00:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

To turn off notifications on a bucket

- Use the `PUT` operation with a request body containing an empty *NotificationConfiguration* element.

Request Routing

Topics

- [Request Redirection and the REST API \(p. 184\)](#)
- [DNS Considerations \(p. 187\)](#)

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.



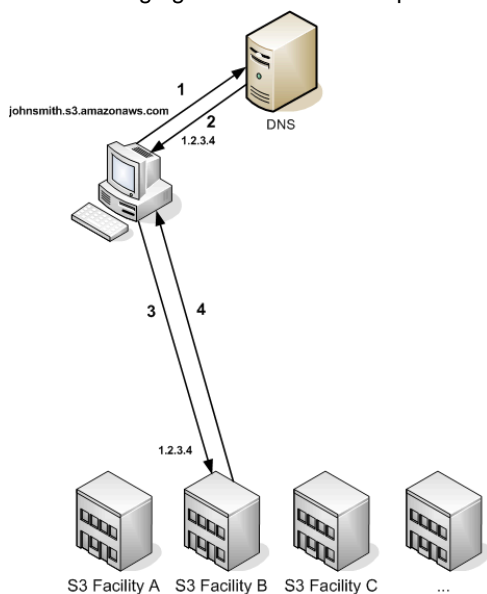
Important

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 [Buckets and Regions \(p. 67\)](#).

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 will receive a temporary redirect. After information about the bucket propagates through DNS, redirects will be rare.

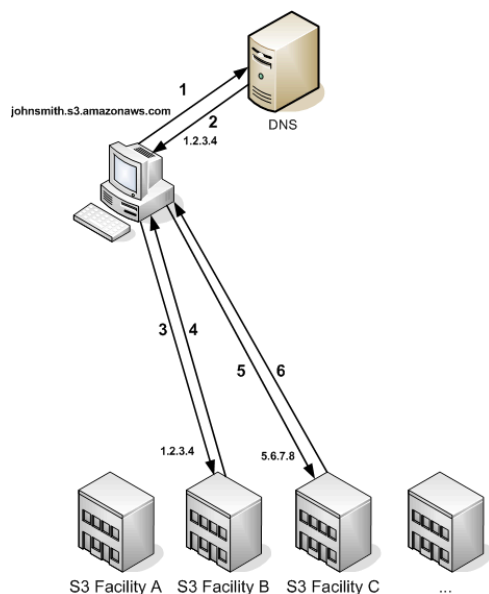
Temporary redirects contain a URI to the correct facility which you can use to immediately resend the request.



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 [Using CreateBucketConfiguration \(p. 67\)](#).

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/pho
tos/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

```
<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.</Fault
string>
    <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

- [TCP Window Scaling \(p. 188\)](#)
- [TCP Selective Acknowledgement \(p. 188\)](#)

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/>.

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 \(p. 189\)](#)
- [Using BitTorrent to Retrieve Objects Stored in Amazon S3 \(p. 190\)](#)
- [Publishing Content Using Amazon S3 and BitTorrent \(p. 191\)](#)

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.

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.

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 [Access Control \(p. 141\)](#).

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 \(p. 192\)](#)
- [Amazon DevPay Token Mechanism \(p. 193\)](#)
- [Amazon S3 and Amazon DevPay Authentication \(p. 193\)](#)
- [Amazon S3 Bucket Limitation \(p. 194\)](#)
- [Amazon S3 and Amazon DevPay Process \(p. 194\)](#)
- [Additional Information \(p. 195\)](#)

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

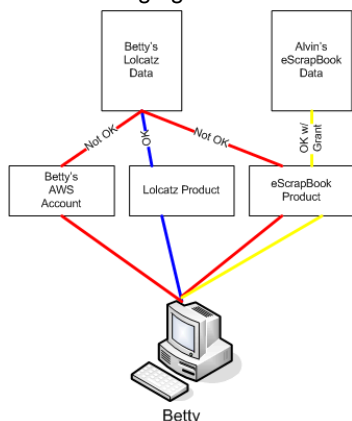
Amazon DevPay requests store and access data on behalf of the users of your product. The resources created by your application are owned by your users; unless you modify the ACL, you cannot read or modify the user's data.

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 other's 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 Developer Guide*.

Amazon S3 and Amazon DevPay Authentication

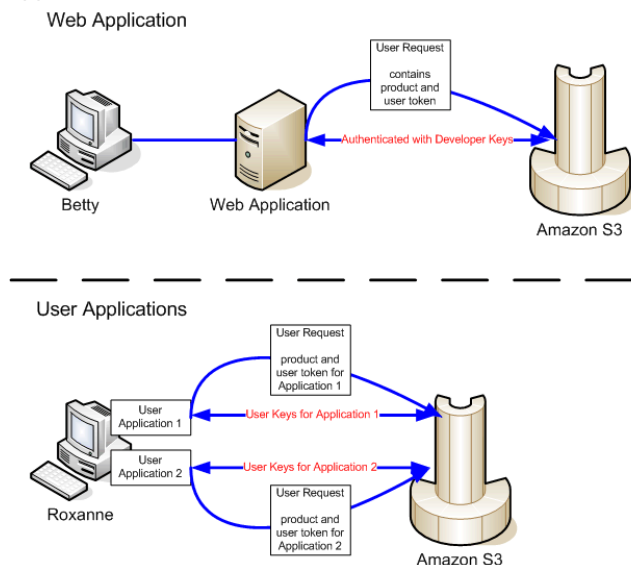
Although the token mechanism uniquely identifies a customer and product, it does not provide authentication.

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, refer to the *Amazon DevPay Developer Guide*.

Handling Errors

Topics

- [The REST Error Response](#) (p. 196)
- [The SOAP Error Response](#) (p. 197)
- [Amazon S3 Error Best Practices](#) (p. 198)

This section describes REST and SOAP errors and how to handle them.

The REST Error Response

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. 197)
- [Error Message](#) (p. 197)
- [Further Details](#) (p. 197)

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.

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`.

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

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 the 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 <http://developer.amazonwebservices.com/connect/forum.jspa?forumID=24> or sign up for AWS Premium Support <http://aws.amazon.com/premiumsupport/>.

Isolate Errors

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

Topics

- [Server Access Logging Configuration API \(p. 201\)](#)
- [Delivery of Server Access Logs \(p. 203\)](#)
- [Server Access Log Format \(p. 204\)](#)
- [Setting Up Server Access Logging \(p. 208\)](#)



Important

This section describes Beta functionality that is subject to change in future releases. Please provide feedback on this functionality in the [Amazon S3 Developer Forum](#).

An Amazon S3 bucket can be configured to create access log records for the requests made against it. An access log record contains details about the request such as the request type, the resource with which the request worked, and the time and date that the request was processed. Server access logs are useful for many applications, because they give bucket owners insight into the nature of requests made by clients not under their control.

By default, server access logs are not collected for a bucket. To learn how to enable server access logging, see [Server Access Logging Configuration API \(p. 201\)](#).

Once logging is enabled for a bucket, available log records are periodically aggregated into log files and delivered to you via an Amazon S3 bucket of your choosing. For a detailed description of this process, see [Delivery of Server Access Logs \(p. 203\)](#).

For information on how to interpret the contents of log files, see [Server Access Log Format \(p. 204\)](#).

To walk through the process of enabling logging for your bucket, see [Setting Up Server Access Logging \(p. 208\)](#).



Note

There is no extra charge for enabling the server access logging feature 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 for data transfer in the usual way.

Server Access Logging Configuration API



Important

This section describes Beta functionality that is subject to change in future releases. Please provide feedback on this functionality in the [Amazon S3 Developer Forum](#).

Each Amazon S3 bucket has an associated XML sub-resource that you can read and write in order to inspect or change the logging status for that bucket. The XML schema for the bucket logging status resource is common across SOAP and REST.

The `BucketLoggingStatus` element has the following structure.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <LoggingEnabled>
    <TargetBucket>mylogs</TargetBucket>
    <TargetPrefix>access_log-</TargetPrefix>
    <TargetGrants>
      <Grant>
        <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="AmazonCustomerByEmail">
          <EmailAddress>email_address</EmailAddress>
        </Grantee>
        <Permission>permission</Permission>
      </Grant>
    </TargetGrants>
  </LoggingEnabled>
</BucketLoggingStatus>
```

Following is a list of elements that belong to the `BucketLoggingStatus` element.

- *LoggingEnabled*

The presence of this element indicates that server access logging is enabled for the bucket. The absence of this element (and all nested elements) indicates that logging is disabled for the bucket.

- *TargetBucket*

This element specifies the bucket where server access logs will be delivered. You can have your logs delivered to any bucket that you own, including the same bucket that is being logged. You can also configure multiple buckets to deliver their logs to the same target bucket. In this case you should choose a different `TargetPrefix` for each source bucket so that the delivered log files can be distinguished by key.



Note

The source and the target buckets must be in the same location. For more information about bucket location constraints, see [Buckets and Regions \(p. 67\)](#)

- *TargetPrefix*

This element lets you specify a prefix for the keys that the delivered log files will be stored under. For information on how the key name for log files is constructed, see [Delivery of Server Access Logs \(p. 203\)](#).

- *TargetGrants*

The bucket owner is automatically granted FULL_CONTROL to all logs delivered to the bucket. This optional element enables you grant access to others. Any specified TargetGrants are added to the default ACL. For more information about ACLs, see [Access Control Lists \(p. 12\)](#).

To enable server access logging, Set or PUT a BucketLoggingStatus with a nested LoggingEnabled element. To disable server access logging, Set or PUT an empty BucketLoggingStatus element.

In REST, the address of the BucketLoggingStatus resource for a bucket 'mybucket' is `http://s3.amazonaws.com/mybucket?logging`. The PUT and GET methods are valid for this resource. For example, the following request fetches the BucketLoggingStatus resource for mybucket.

```
GET ?logging HTTP/1.1
Host: mybucket.s3.amazonaws.com
Date: Wed, 25 Nov 2009 12:00:00 GMT
Authorization: AWS YOUR_AWS_ACCESS_KEY_ID:YOUR_SIGNATURE_HERE

HTTP/1.1 200 OK
Date: Wed, 25 Nov 2009 12:00:00 GMT
Connection: close
Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <LoggingEnabled>
    <TargetBucket>mybucketlogs</TargetBucket>
    <TargetPrefix>mybucket-access_log-</TargetPrefix>
    <TargetGrants>
      <Grant>
        <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="AmazonCustomerByEmail">
          <EmailAddress>user@company.com</EmailAddress>
        </Grantee>
        <Permission>READ</Permission>
      </Grant>
    </TargetGrants>

  </LoggingEnabled>
</BucketLoggingStatus>
```

In SOAP, you can work with BucketLoggingStatus resource using the [SOAPSetBucketLoggingStatus](#) and [SOAPGetBucketLoggingStatus](#) operations.

Amazon S3 checks the validity of the proposed BucketLoggingStatus when you try to Set or PUT to it. If the TargetBucket does not exist, is not owned by you, or does not have the appropriate grants, you will receive the InvalidTargetBucketForLogging error. If your proposed BucketLoggingStatus document is not well-formed XML or does not match our published schema, you will receive the MalformedXML error.

BucketLoggingStatus Changes Take Effect Over Time

Changes to the logging status for a bucket are visible in the configuration API immediately, but they 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. Or, 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.

Delivery of Server Access Logs



Important

This section describes Beta functionality that is subject to change in future releases. Please provide feedback on this functionality in the [Amazon S3 Developer Forum](#).

Server access logs are written to the bucket of your choice, which can be the bucket from which the logs originate or a different bucket. If you choose a different bucket, it must have the same owner as the source bucket. Otherwise, no logs will be delivered.



Note

The source and the target buckets must be in the same location. For more information about bucket location constraints, see [Buckets and Regions \(p. 67\)](#).

When a log file is delivered to the target bucket, it is stored under a key in the following format.

*TargetPrefix*YYYY-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 TargetPrefix component of the key is a string provided by the bucket owner using the logging configuration API. For more information, see [Server Access Logging Configuration API \(p. 201\)](#).

The UniqueString component of the key carries no meaning and should be ignored by log processing software.

The system does not delete old log files. If you do not want server logs to accumulate, you must delete them yourself. To do so, use the List operation with the `prefix` parameter to locate old logs to delete. For more information, see [Listing Object Keys \(p. 132\)](#).

Access Control Interaction

Log files will be written to the target bucket under the identity of a member of the `http://acs.amazonaws.com/groups/s3/LogDelivery` group. These writes are subject to the usual access control restrictions. Therefore, logs will not be delivered unless the access control policy of the target bucket grants the log delivery group `WRITE` access. To ensure log files are delivered correctly, the log delivery group must also have `READ_ACP` permission on the target bucket. For more information about access control lists and groups, see [Access Control \(p. 141\)](#). For more information about correctly configuring your target bucket's access control policy, see the [Setting Up Server Access Logging \(p. 208\)](#).

Log files created in the target bucket have an access control list entry that consists of a `FULL_CONTROL` grant to the bucket owner and grants to any users specified through the `TargetGrants` element.

Best Effort Server Log Delivery

The server access logging feature is designed for best effort. You can expect that most requests against a bucket that is properly configured for logging will result in a delivered log record, and that most log records will be delivered within a few hours of the time that they were recorded.

However, the server logging feature is offered on a best-effort basis. The completeness and timeliness of server logging 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 the bucket owner an idea of the nature of traffic against his or her bucket. It is not meant to be a complete accounting of all requests.

Usage Report Consistency

It follows from the best-effort nature of the server logging feature that the usage reports available at the AWS portal might include usage that does not correspond to any request in a delivered server log.

Server Access Log Format



Important

This section describes Beta functionality that is subject to change in future releases. Please provide feedback on this functionality in the [Amazon S3 Developer Forum](#).

The log files consist of a sequence of new-line delimited log records. Log records appear in no particular order. Each log record represents one request and consists of space delimited fields described in the following table.

Field Name	Example Entry	Notes
Bucket Owner	314159b66967d86f031c7249d1d9a8024	The canonical user id of the owner of the source bucket.
	9109428335cd0ef1cdc487b4566cb1b	

Amazon Simple Storage Service Developer Guide
Server Access Log Format

Field Name	Example Entry	Notes
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	[04/Aug/2006:22:34:02 +0000]	The time at which the request was received. The format, using <code>strftime()</code> terminology, is <code>[%d/%B/%Y:%H:%M:%S %z]</code>
Remote IP	72.21.206.5	The apparent Internet address of the requester. Intermediate proxies and firewalls might obscure the actual address of the machine making the request.
Requester	314159b66967d86f031c7249d1d9a80	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.
	249109428335cd0ef1cdc487b4566cb1b	
Request ID	3E57427F33A59F07	The request ID is a string generated by Amazon S3 to uniquely identify each request.
Operation	SOAP.CreateBucket	Either SOAP. <i>operation</i> or REST. <i>HTTP_method.resource_type</i>
	or	
	REST.PUT.OBJECT	
Key	/photos/2006/08/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/2006/08/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. 197) , or "-" if no error occurred.

Field Name	Example Entry	Notes
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.
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/webservices"	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	3HL4kqtJvjVBH40Nrjfd	The version ID in the request, or "-" if the operation does not take a <i>versionId</i> parameter.

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.

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/2006/08/puppy.jpg?x-user=johndoe" will work the same as the same request for "s3.amazonaws.com/mybucket/photos/2006/08/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.

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	314159b66967d86f031c7249d1d9a8024 9109428335cd0ef1cdc487b4566cb1b	The owner of the bucket that stores the object being copied.
Bucket	mybucket	The name of the bucket that stores the object being copied.
Time	[04/Aug/2006:22:34:02 +0000]	The time at which the request was received. The format, using <code>strftime()</code> terminology, is <code>[%d/%B/%Y:%H:%M:%S %z]</code>
Remote IP	72.21.206.5	The apparent Internet address of the requester. Intermediate proxies and firewalls might obscure the actual address of the machine making the request.
Requester	314159b66967d86f031c7249d1d9a80 249109428335cd0ef1cdc487b4566cb1b	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> or <code>REST.HTTP_method.resource_type</code>
Key	/photos/2006/08/puppy.jpg	The "key" of the object being copied or "-" if the operation does not take a key parameter.
Request-URI	"GET /mybucket/photos/2006/08/ puppy.jpg?x-foo=bar"	The Request-URI part of the HTTP request message.

Field Name	Example Entry	Notes
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. 197) , 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.
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/webservices"	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	3HL4kqtJvjVBH40Nrjfk	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.

Setting Up Server Access Logging



Important

This section describes Beta functionality that is subject to change in future releases. Please provide feedback on this functionality in the [Amazon S3 Developer Forum](#).

The Amazon S3 server access logging feature lets you generate access log files for buckets that you own. These log files are delivered to you by writing them into a (possibly different) bucket that you own. Once delivered, the access logs are ordinary objects that you can read, list or delete at your convenience.

These instructions assume that you want to enable server access logging on one of your pre-existing buckets, and that you want to have those logs delivered into a new bucket you will create just for logging. We suppose that the bucket you want to log access to is called 'mybucket' and the new bucket you will create to hold your access logs is called 'mylogs'. This makes 'mybucket' the source bucket for logging and 'mylogs' the target bucket for logging. Whenever you see 'mybucket' or 'mylogs' in the example, replace them with the name of your bucket that you want to log, and the bucket you want to store your access logs, respectively.

This tutorial makes use of s3curl (go to [s3curl.pl sample program](#)) to work with the Amazon S3 REST API. Make sure you use the most recent version of s3curl, as it has been updated to support this tutorial. After invoking s3curl, always check for a 200 OK HTTP response. If you get some other response code, refer to the XML error response which likely contains information about what went wrong.

Preparing the Target Bucket

To prepare the target bucket

1. First, decide if you want your logs delivered to an existing bucket, or if you want to create a new bucket just for access log files. Following is a command that creates a new target bucket for logging. Notice the canned ACL argument that grants the system permission to write log files to this bucket.



Note

The source and the target buckets must be in the same location. For more information about bucket location constraints, see [Buckets and Regions \(p. 67\)](#)

```
$ ./s3curl.pl --id YOUR_AWS_ACCESS_KEY_ID --key YOUR_AWS_SECRET_ACCESS_KEY -  
-acl log-delivery-write --put /dev/null -- -s -v http://s3.amazonaws.com/mylogs
```

2. If you just created a new bucket for logging, skip to the next section. Otherwise, to have your access logs files delivered to an existing bucket, you must modify the access control policy of that bucket by hand. Fetch the ?acl sub-resource of the target bucket and save it to a local file:

```
$ ./s3curl.pl --id YOUR_AWS_ACCESS_KEY_ID --key YOUR_AWS_SECRET_ACCESS_KEY -  
- -s -v 'http://s3.amazonaws.com/mylogs?acl' > mylogs.acl
```

3. Now open the local copy of the logging resource in your favorite text editor and insert a new <Grant> element to the <AccessControlList> section that gives the log delivery group WRITE and READ_ACP permission to your bucket.

```
<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>
```

4. Finally, apply the modified access control policy by writing it back to Amazon S3.

```
$ ./s3curl.pl --id YOUR_AWS_ACCESS_KEY_ID --key YOUR_AWS_SECRET_ACCESS_KEY -
-put mylogs.acl -- -s -v 'http://s3.amazonaws.com/mylogs?acl'
```

Enabling Server Access Logging on the Source Bucket

Now that the target bucket can accept log files, we'll update the `?logging` sub-resource of the source bucket to turn on server access logging. Remember that you must be the bucket owner to read or write this resource.

Fetch the `?logging` sub-resource for modification using the command shown in the following example.

Example

```
$ ./s3curl.pl --id YOUR_AWS_ACCESS_KEY_ID --key YOUR_AWS_SECRET_ACCESS_KEY --
-s -v 'http://s3.amazonaws.com/mybucket?logging' > mybucket.logging
```


Open `mybucket.logging` in your favorite text editor and uncomment the `<LoggingSettings>` section. Replace the contents of the `<TargetBucket>` and `<TargetPrefix>` with `'mylogs'` and `'mybucket-access_log-'` respectively.

Additionally, to grant users access to log files within the bucket, you can specify one or more users in the `<TargetGrants>` section. You can specify users through their e-mail address (`EmailAddress`) or canonical user ID (`CanonicalUser`). Permissions include `READ`, `WRITE`, and `FULL_CONTROL`. The result should be similar to the following.

Example

```
<?xml version="1.0" encoding="UTF-8"?>

<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">

  <LoggingEnabled>

    <TargetBucket>mylogs</TargetBucket>

    <TargetPrefix>mybucket-access_log-</TargetPrefix>

    <TargetGrants>

      <Grant>

        <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="AmazonCustomerByEmail">

          <EmailAddress>user@company.com</EmailAddress>

        </Grantee>

        <Permission>READ</Permission>

      </Grant>

    </TargetGrants>

  </LoggingEnabled>

</BucketLoggingStatus>
```



Note

For general information about authentication, see [Access Control \(p. 141\)](#).

Now apply your modifications by writing the document back to the `?logging` sub-resource in Amazon S3.

Example

```
$ ./s3curl.pl --id YOUR_AWS_ACCESS_KEY_ID --key YOUR_AWS_SECRET_ACCESS_KEY -  
-put mybucket.logging -- -s -v 'http://s3.amazonaws.com/mybucket?logging'
```

You can confirm your changes by fetching the `?logging` sub-resource and comparing it to what you just wrote.

Server access logging should now be enabled. Make a few requests against the source bucket now, and your access logs should begin to be delivered to the target bucket within the next few hours.

Disabling Server Logging for a Bucket

Fetch, modify, and apply the `?logging` sub resource in the same way as described in the preceding procedure, except use your text editor to remove the `<LoggingEnabled>` element.



Note

Log changes do not take effect immediately; logs will be delivered for a while after disabling logging.

Appendix: The Access Policy Language

Topics

- [Overview \(p. 213\)](#)
- [How to Write a Policy \(p. 222\)](#)
- [Special Information for Amazon S3 Policies \(p. 233\)](#)

This appendix is for Amazon S3 users who want to write their own access control policies. You don't need to write your own policies if you want to allow access based only on AWS account ID and basic permissions. If you want to explicitly deny access or allow it based on finer conditions (like the time the request comes in or the IP address of the requester), you need to write your own policies and upload them to AWS.



Note

To write your own policies, you must be familiar with JSON. For more information, go to <http://json.org>.

The main portion of this appendix includes basic concepts you need to understand, how to write a policy, and the logic AWS uses to evaluate policies and decide whether to give the requester access to the resource. Although most of the information in this appendix is service-agnostic, there are some Amazon S3-specific details you need to know. For more information, see [Special Information for Amazon S3 Policies \(p. 233\)](#).

Overview

Topics

- [Key Concepts \(p. 214\)](#)
- [Architectural Overview \(p. 216\)](#)
- [Using the Access Policy Language \(p. 218\)](#)
- [Evaluation Logic \(p. 219\)](#)

This section describes basic concepts you need to understand to use access policy language to write policies. It also describes the general process for how access control works with the access policy language, and how policies are evaluated.

Key Concepts

The following sections describe the concepts you need to understand to use the access policy language. They're presented in a logical order, with the first terms you need to know at the top of the list.

Permission

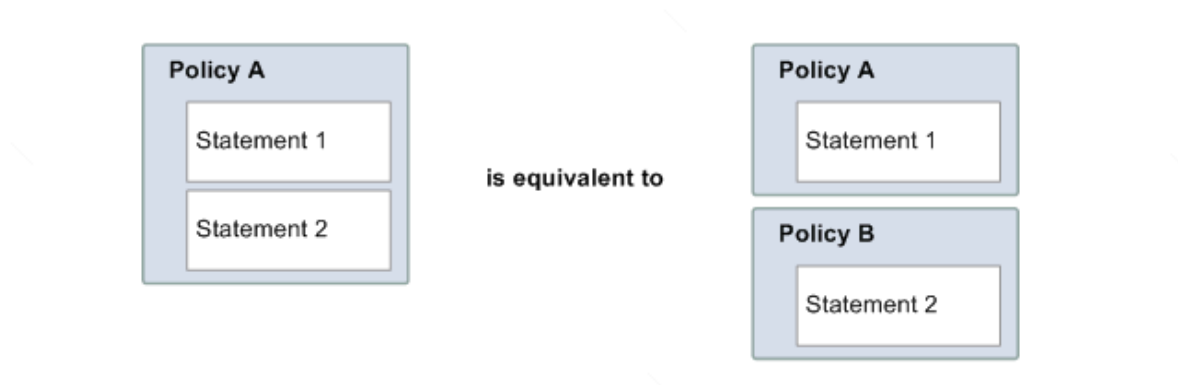
A *permission* is the concept of allowing or disallowing some kind of access to a particular resource. Permissions essentially follow this form: "A is/isn't allowed to do B to C where D applies." For example, *Jane* (A) has permission to *put an object* (B) in *AcmeProductsBucket* (C) as long as an IP address is within a certain range (D). Whenever Jane sends a request to Amazon S3 to put an object in that bucket, the service checks to see if she has permission and if the request satisfies the conditions the bucket owner set forth in the policy.

Statement

A *statement* is the formal description of a single permission, written in the access policy language. You always write a statement as part of a broader container document known as a *policy* (see the next concept).

Policy

A *policy* is a document (written in JSON) that acts as a container for one or more statements. For example, a policy could have two statements in it: one that states that Jane can put objects in a specified bucket, and another that states that Bob cannot put objects into the same bucket. As shown in the following figure, an equivalent scenario would be to have two policies.



Amazon S3 uses the information in the statements (whether they're contained in a single policy or multiple) to determine if someone requesting access to a resource should be granted access.

Issuer

The *issuer* is the person who writes a policy to grant permissions for a resource. The issuer (by definition) is always the resource owner. AWS does not permit AWS service users to create policies for resources they don't own. If John is the resource owner, AWS authenticates John's identity when he submits the policy he's written to grant permissions for that resource.

Principal

The *principal* is the person or persons who receive the permission in the policy. The principal is A in the statement "A has permission to do B to C where D applies." In a policy, you can set the principal to "anyone" (i.e., you can specify a wildcard to represent all people). You might do this, for example, if you want to grant everyone (whether they are authenticated or anonymous) access to a subset of your Amazon S3 resources.

Action

The *action* is the activity the principal has permission to perform. The action is B in the statement "A has permission to do B to C where D applies." Typically, the action is just the operation in the request to AWS. You can specify one or multiple actions in a policy.

For the names of the Amazon S3 actions you can specify in a policy, see [Amazon S3 Operations \(p. 148\)](#).

Resource

The *resource* is the bucket or object the principal is requesting access to. The resource is C in the statement "A has permission to do B to C where D applies." You can specify one or more resources in a policy. For information about to specify a resource in a policy, see [Specifying Amazon S3 Resources in Bucket Policies \(p. 147\)](#).

Conditions and Keys

The *conditions* are any restrictions or details about the permission. The condition is D in the statement "A has permission to do B to C where D applies." The part of the policy that specifies the conditions can be the most detailed and complex of all the parts. Typical conditions are related to:

- Date and time (e.g., the request must arrive before a specific day)
- IP address (e.g., the requester's IP address must be part of a particular CIDR range)

A *key* is the specific characteristic that is the basis for access restriction. For example, the date and time of a request.

You use both *conditions* and *keys* together to express the restriction. The easiest way to understand how you actually implement a restriction is with an example: If you want to restrict access to before May 30, 2010, you use the condition called `DateLessThan`. You use the key called `and` and set it to the value `2010-05-30T00:00:00Z`. AWS defines the conditions and keys you can use. The might also define service-specific keys. For more information about conditions, see [Condition \(p. 226\)](#). For more information about the available keys, see [Bucket Keys in Amazon S3 Policies \(p. 149\)](#) and [Object Keys in Amazon S3 Policies \(p. 151\)](#).

Requester

The *requester* is the person who sends a request to an AWS service and asks for access to a particular resource. The requester sends a request to AWS that essentially says: "Will you allow me to do B to C where D applies?"

Evaluation

Evaluation is the process the AWS service uses to determine if an incoming request should be denied or allowed based on the applicable policies. For information about the evaluation logic, see [Evaluation Logic \(p. 219\)](#).

Effect

The *effect* is the result that you want a policy statement to return at evaluation time. You specify this value when you write the statements in a policy, and the possible values are *deny* and *allow*.

For example, you could write a policy that has a statement that *denies* all requests that come from Antarctica (effect=deny given that the request uses an IP address allocated to Antarctica). Alternately, you could write a policy that has a statement that *allows* all requests that *don't* come from Antarctica (effect=allow, given that the request doesn't come from Antarctica). Although the two statements sound like they do the same thing, in the access policy language logic, they are different. For more information, see [Evaluation Logic \(p. 219\)](#).

Although there are only two possible values you can specify for the effect (allow or deny), there can be three different results at policy evaluation time: *default deny*, *allow*, or *explicit deny*. For more information, see the following concepts and [Evaluation Logic \(p. 219\)](#).

Default Deny

A *default deny* is the default result from a policy in the absence of an allow or explicit deny.

Allow

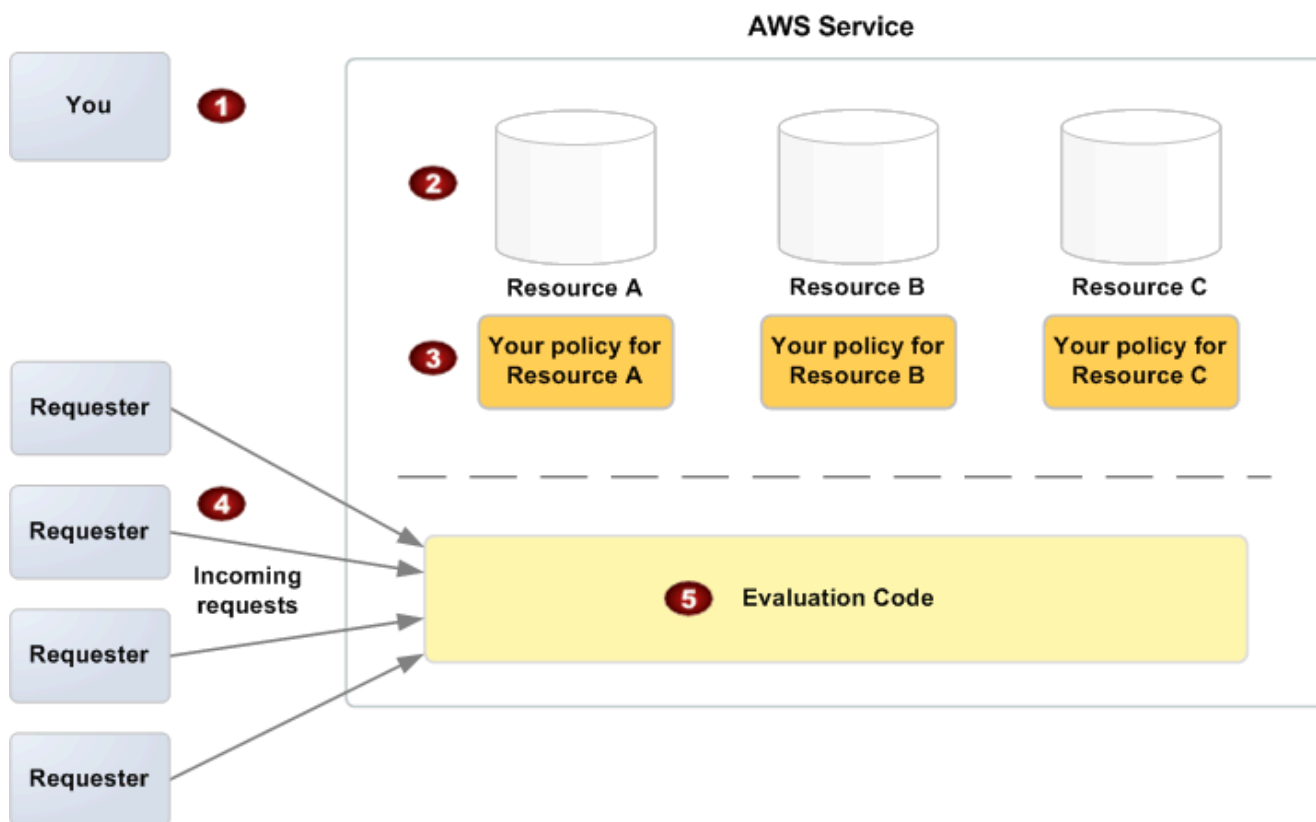
An *allow* results from a statement that has effect=allow, assuming any stated conditions are met. Example: Allow requests if they are received before 1:00 p.m. on April 30, 2010. An allow overrides all default denies, but never an explicit deny.

Explicit Deny

An *explicit deny* results from a statement that has effect=deny, assuming any stated conditions are met. Example: Deny all requests if they are from Antarctica. Any request that comes from Antarctica will always be denied no matter what any other policies might allow.

Architectural Overview

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

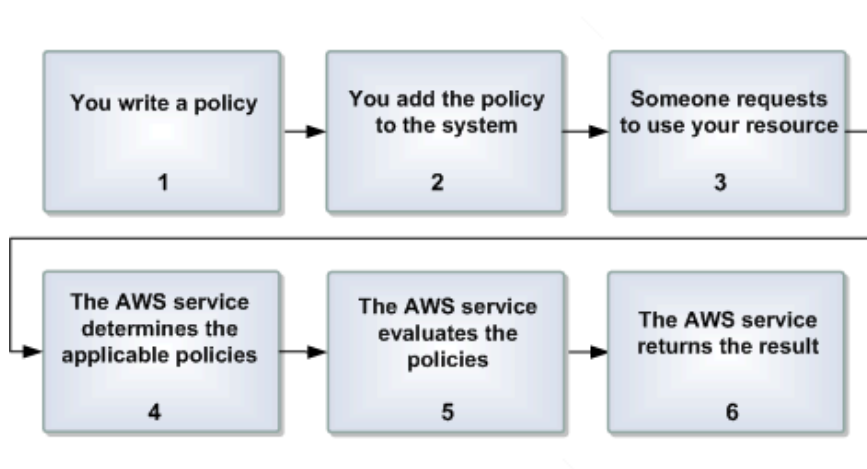


1	You, the resource owner.
2	Your resources (contained within the AWS service; e.g., Amazon S3 buckets or objects).
3	Your policies. In Amazon S3 there is only one policy per bucket. Amazon S3 provides an API that enables to you to upload and manage your bucket policy. For information about the content of the policies, see How to Write a Policy (p. 222) .
4	Requesters and their incoming requests to the AWS service.
5	The access policy language evaluation code. This is the set of code within the AWS service that evaluates incoming requests against the applicable policies and determines whether the requester is allowed access to the resource. For information about how the service makes the decision, see Evaluation Logic (p. 219) .

For the typical process of how the components work together, see [Using the Access Policy Language \(p. 218\)](#).

Using the Access Policy Language

The following figure and table describe the general process of how access control works with the access policy language.



Process for Using Access Control with access policy language

1	You write a policy for your resource. For example, you write a policy to specify permissions for your Amazon S3 objects. For more information, see How to Write a Policy (p. 222) .
2	You upload your policy to AWS. The AWS service itself provides an API you use to upload your policies. For example, you use the Amazon S3 <code>PUT Bucket policy</code> action to set a policy on a bucket.
3	Someone sends a request to use your resource. For example, a user sends a request to Amazon S3 to upload an object to a bucket.
4	The AWS service determines which policies are applicable to the request. For example, Amazon S3 looks at all the available Amazon S3 policies and determines which ones are applicable (based on what the resource is, who the requester is, etc.).
5	The AWS service evaluates the policies. For example, Amazon S3 evaluates the policies and determines if the requester is allowed to upload the object to the bucket. For information about the decision logic, see Evaluation Logic (p. 219) .
6	The AWS service either denies the request or continues to process it. For example, based on the policy evaluation result, the service either returns an "Access denied" error to the requester or continues to process the request.

Related Topics

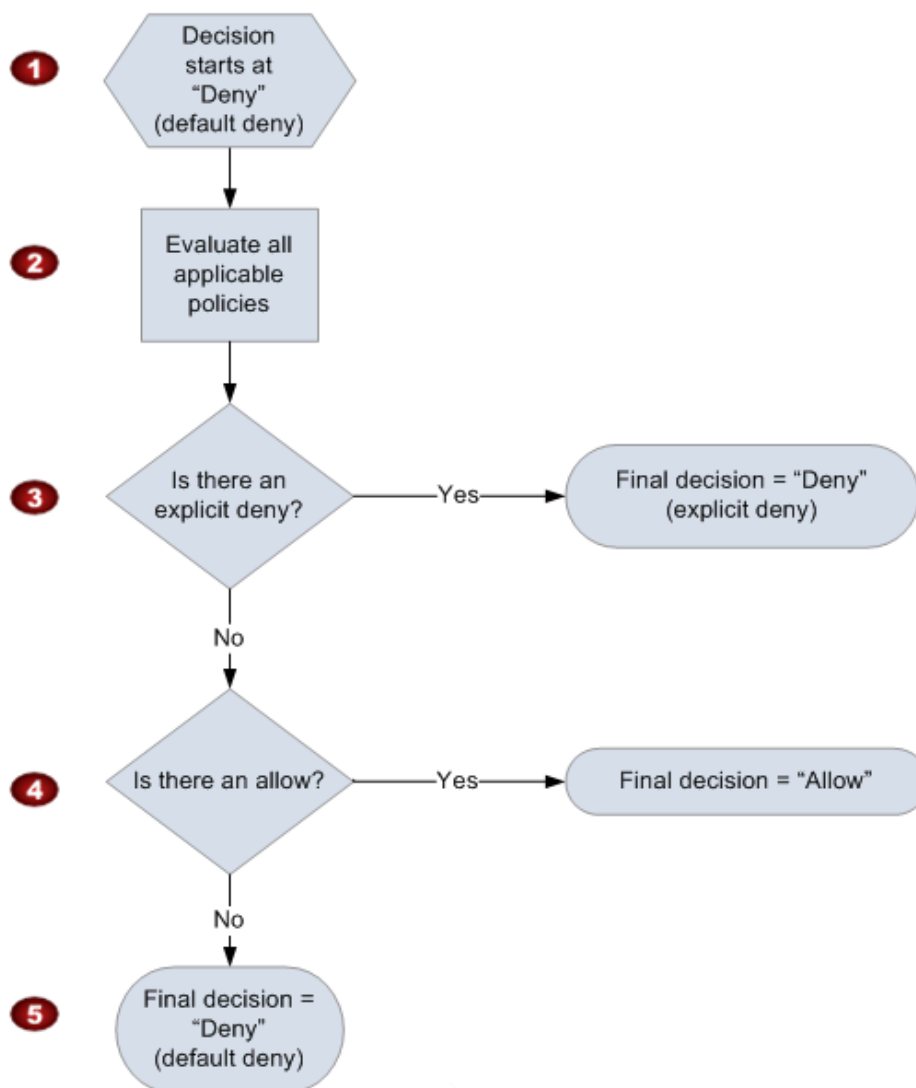
- [Architectural Overview \(p. 216\)](#)

Evaluation Logic

The goal at evaluation time is to decide whether a given request should be allowed or denied. The evaluation logic follows several basic rules:

- By default, all requests to use your resource coming from anyone but you are denied
- An allow overrides any default denies
- An explicit deny overrides any allows
- The order in which the policies are evaluated is not important

The following flow chart and discussion describe in more detail how the decision is made.



1	The decision starts with a default deny.
---	--

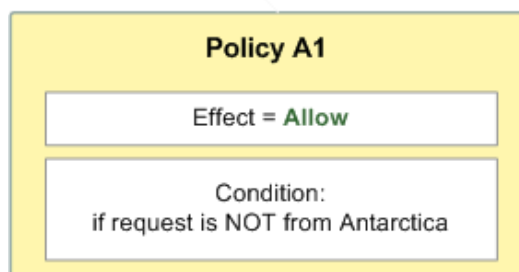
2	The enforcement code then evaluates that are applicable to the request (based on the resource, principal, action, and conditions). The order in which the enforcement code evaluates the policies is not important.
3	In all those policies, the enforcement code looks for an explicit deny instruction that would apply to the request. If it finds even one, the enforcement code returns a decision of "deny" and the process is finished (this is an explicit deny; for more information, see Explicit Deny (p. 216)).
4	If no explicit deny is found, the enforcement code looks for any "allow" instructions that would apply to the request. If it finds even one, the enforcement code returns a decision of "allow" and the process is done (the service continues to process the request).
5	If no allow is found, then the final decision is "deny" (because there was no explicit deny or allow, this is considered a default deny (for more information, see Default Deny (p. 216))).

The Interplay of Explicit and Default Denials

A policy results in a default deny if it doesn't directly apply to the request. For example, if an account requests to use Amazon S3, but the only policy that applies to the account states that the account can only list the contents of an Amazon S3 bucket, then that policy results in a default deny.

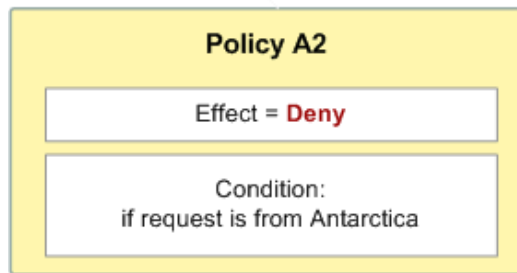
A policy also results in a default deny if a condition in a statement isn't met. If all conditions in the statement are met, then the policy results in either an allow or an explicit deny, based on the value of the *Effect* element in the policy.

For example, let's say you want to prevent requests coming in from Antarctica. You write a policy (called Policy A1) that allows a request only if it doesn't come from Antarctica. The following diagram illustrates the policy.



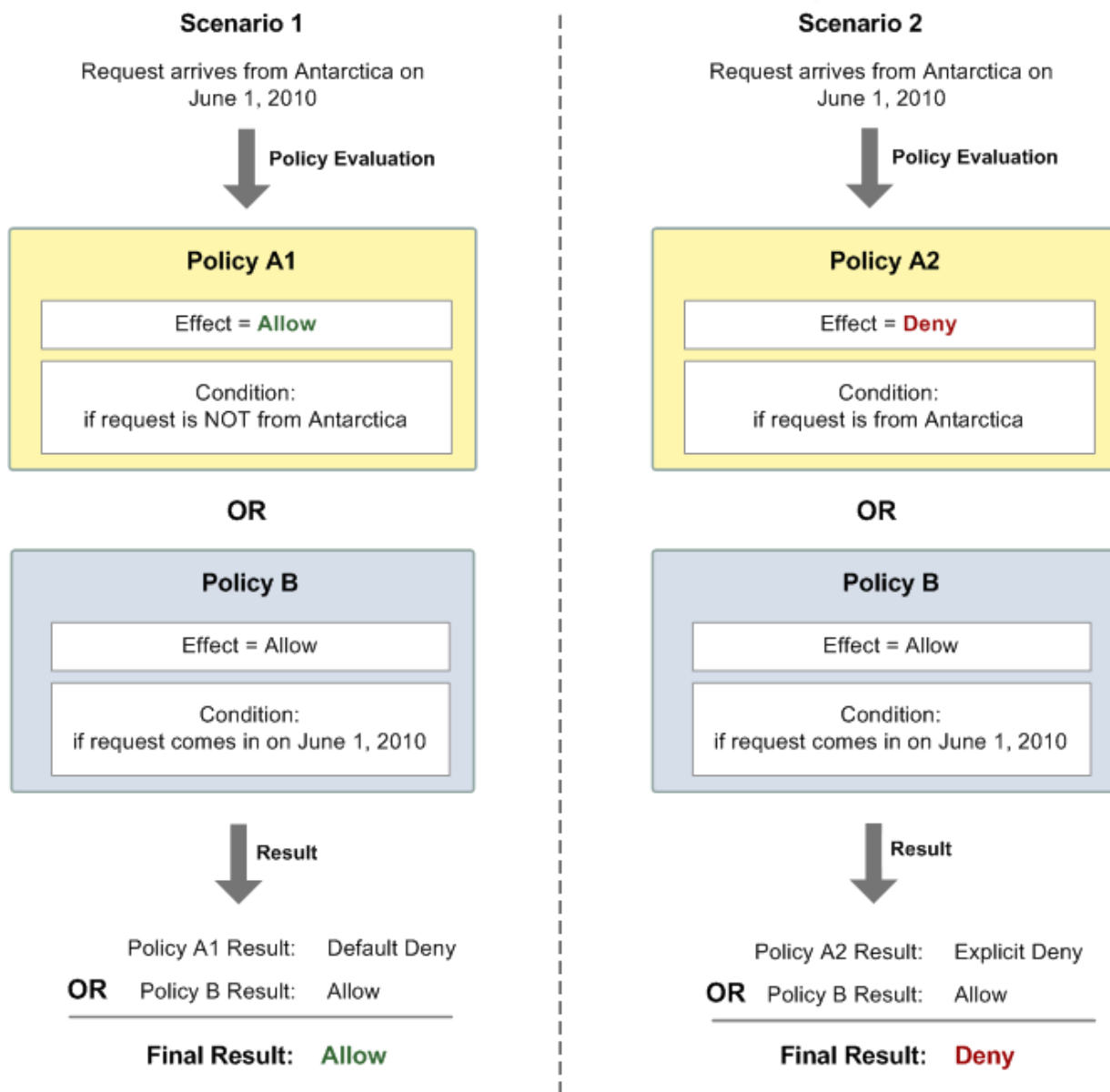
If someone sends a request from the U.S., the condition is met (the request is not from Antarctica). Therefore, the request is allowed. But, if someone sends a request from Antarctica, the condition isn't met, and the policy's result is therefore a default deny.

You could turn the result into an explicit deny by rewriting the policy (named Policy A2) as in the following diagram. Here, the policy explicitly denies a request if it comes from Antarctica.



If someone sends a request from Antarctica, the condition is met, and the policy's result is therefore an explicit deny.

The distinction between a default deny and an explicit deny is important because a default deny can be overridden by an allow, but an explicit deny can't. For example, let's say there's another policy that allows requests if they arrive on June 1, 2010. How does this policy affect the overall outcome when coupled with the policy restricting access from Antarctica? We'll compare the overall outcome when coupling the date-based policy (we'll call Policy B) with the preceding policies A1 and A2. Scenario 1 couples Policy A1 with Policy B, and Scenario 2 couples Policy A2 with Policy B. The following figure and discussion show the results when a request comes in from Antarctica on June 1, 2010.



In Scenario 1, Policy A1 returns a default deny, as described earlier in this section. Policy B returns an allow because the policy (by definition) allows requests that come in on June 1, 2010. The allow from Policy B overrides the default deny from Policy A1, and the request is therefore allowed.

In Scenario 2, Policy B2 returns an explicit deny, as described earlier in this section. Again, Policy B returns an allow. The explicit deny from Policy A2 overrides the allow from Policy B, and the request is therefore denied.

How to Write a Policy

Topics

- [Basic Policy Structure \(p. 223\)](#)

- [Element Descriptions](#) (p. 223)
- [Supported Data Types](#) (p. 232)

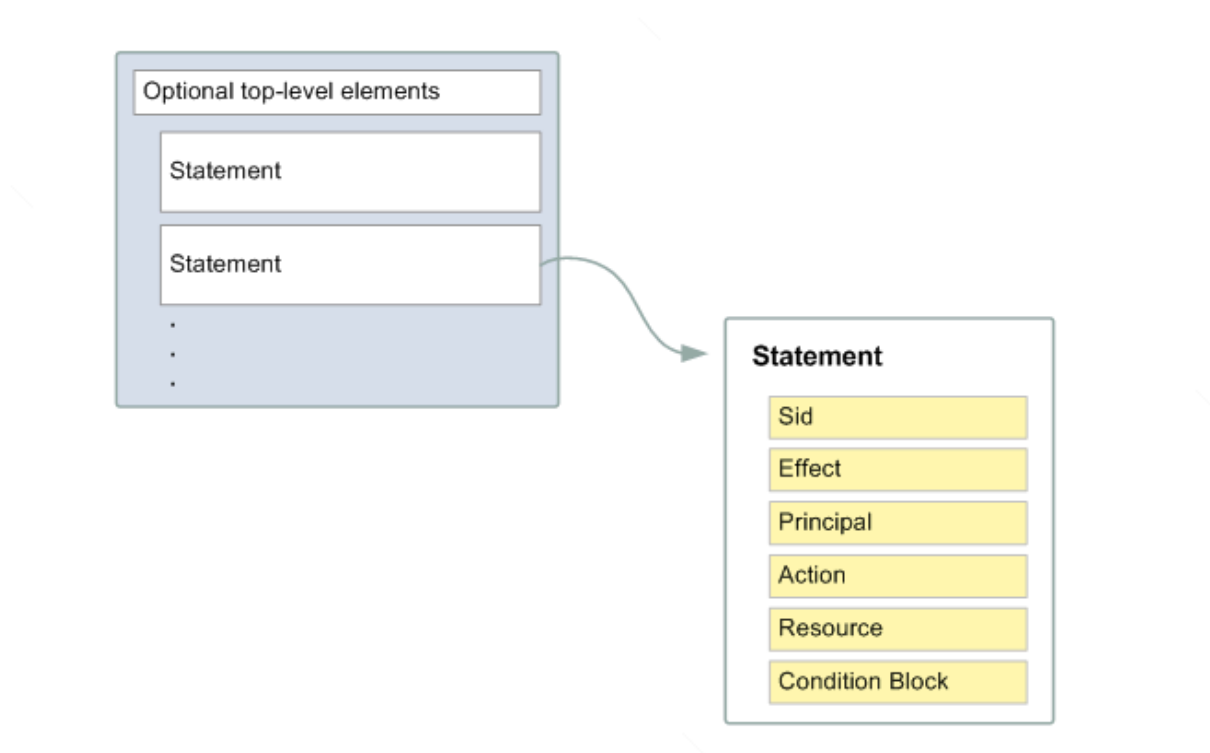
This section describes how to write policies and gives reference information about each policy element.

Basic Policy Structure

Each policy is a JSON document. As illustrated in the following figure, a policy includes:

- Optional policy-wide information (at the top of the document)
- One or more individual *statements*

Each statement includes the core information about a single permission. If a policy includes multiple statements, we apply a logical OR across the statements at evaluation time. If multiple policies are applicable to a request, we apply a logical OR across the policies at evaluation time.



The information in a statement is contained within a series of *elements*. For information about these elements, see [Element Descriptions](#) (p. 223).

Element Descriptions

Topics

- [Version](#) (p. 224)
- [Id](#) (p. 224)
- [Permission](#) (p. 224)
- [Statement](#) (p. 224)
- [Policy](#) (p. 225)

- [Issuer](#) (p. 225)
- [Sid](#) (p. 225)
- [Effect](#) (p. 225)
- [Principal](#) (p. 225)
- [Action](#) (p. 226)
- [NotAction](#) (p. 226)
- [Resource](#) (p. 226)
- [Condition](#) (p. 226)

This section describes the elements you can use in a policy and its statements. The elements are listed here in the general order you use them in a policy. The `Id`, `Version`, and `Statement` are top-level policy elements; the rest are statement-level elements. JSON examples are provided.

All elements are optional for the purposes of parsing the policy document itself. The order of the elements doesn't matter (e.g., the `Resource` element can come before the `Action` element). You're not required to specify any Conditions in the policy.

Version

The `Version` is the access policy language version. This is an optional element, and currently the only allowed value is `2008-10-17`.

```
"Version": "2008-10-17"
```

Id

The `Id` is an optional identifier for the policy. We recommend you use a UUID for the value, or incorporate a UUID as part of the ID to ensure uniqueness.



Important

Amazon S3, which implements the access policy language might require the `Id` element and have uniqueness requirements for it. For service-specific information about writing policies, see [Special Information for Amazon S3 Policies](#) (p. 233).

Permission

A permission is the concept of allowing or disallowing some kind of access to a particular resource. Permissions essentially follow this form: "A is/isn't allowed to do B to C where D applies." For example, Jane (A) has permission to receive messages (B) from John's Amazon SQS queue (C), as long as she asks to receive them before midnight on May 30, 2009 (D). Whenever Jane sends a request to Amazon SQS to use John's queue, the service checks to see if she has permission and if the request satisfies the conditions John set forth in the permission.

Statement

The `Statement` is the main element for a statement. It can include multiple elements (see the subsequent sections in this guide).

The `Statement` element contains an array of individual statements. Each individual statement is a distinct JSON block enclosed in curly brackets `{ }`.

```
"Statement": [{...}, {...}, {...}]
```

Policy

A policy is a document (written in the access policy language) that acts as a container for one or more statements. For example, a policy could have two statements in it: one that states that Jane can read John's object, and another that states that Bob cannot read John's object. An equivalent scenario would be to have two policies, one containing the statement that Jane can read John's object, and another containing the statement that Bob cannot read John's object.

Issuer

The issuer is the person who writes a policy to grant permissions for a resource. The issuer (by definition) is always the resource owner. AWS does not permit AWS product users to create policies for resources they don't own. If John is the resource owner, AWS authenticates John's identity when he submits the policy he's written to grant permissions for that resource.

Sid

The `Sid` (statement ID) is an optional identifier you provide for the policy statement. Essentially it is just a sub-ID of the policy document's ID.



Important

The Amazon S3 implementation of the access policy language might require this element and have uniqueness requirements for it. For service-specific information about writing policies, see [Special Information for Amazon S3 Policies \(p. 233\)](#).

```
"Sid" : "1"
```

Effect

The `Effect` is a required element that indicates whether you want the statement to result in an allow or an explicit deny (for more information, see [Explicit Deny \(p. 216\)](#)).

Valid values for `Effect` are `Allow` and `Deny`.

```
"Effect": "Allow"
```

Principal

The `Principal` is the person or persons who receive or are denied permission according to the policy. You must specify the principal by using the principal's AWS account ID (e.g., 1234-5678-9012, with or without the hyphens). You can specify multiple principals, or a wildcard (*) to indicate all possible users. You can view your account ID by logging in to your AWS account at <http://aws.amazon.com> and clicking **Account Activity**.

In JSON, you use `"AWS":` as a prefix for the principal's AWS account ID.

Action

The `Action` is the specific type or types of access allowed or denied (for example, read or write). You can specify multiple values for this element. The values are free-form but must match values the AWS service expects (for more information, see [Special Information for Amazon S3 Policies \(p. 233\)](#)). You can use a wildcard (*) to give the principal access to all the actions the specific AWS service lets you share with other developers.

NotAction

The `NotAction` element is useful if you want to make an exception to a list of actions. You could use this, for example, if you want your users to be able to use only the `GET Object`.

The following example refers to all actions *other* than `GET Object`. You would use this in a policy with `"Effect": "Deny"` to keep users from accessing any other actions.

```
"NotAction": "s3:GetObject"
```

Resource

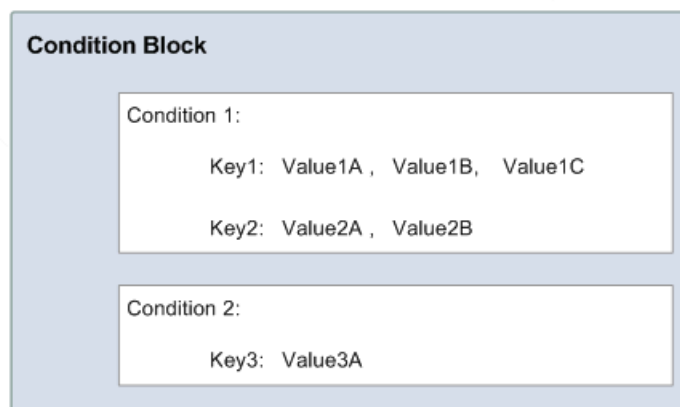
The `Resource` is the object or objects the policy covers. The value can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string. The values are free-form, but must follow the format the AWS service expects.

Condition

This section describes the `Condition` element and the information you can use inside the element.

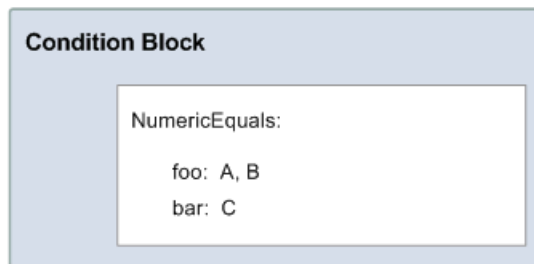
The Condition Block

The `Condition` element is the most complex part of the policy statement. We refer to it as the *condition block*, because although it has a single `Condition` element, it can contain multiple conditions, and each condition can contain multiple key-value pairs. The following figure illustrates this. Unless otherwise specified for a particular key, all keys can have multiple values.

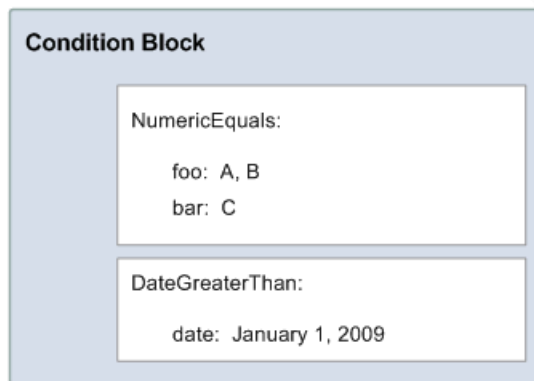


When creating a condition block, you specify the name of each condition, and at least one key-value pair for each condition. AWS defines the conditions and keys you can use (they're listed in the subsequent sections). An example of a condition is `NumericEquals`. Let's say you have a fictional resource, and

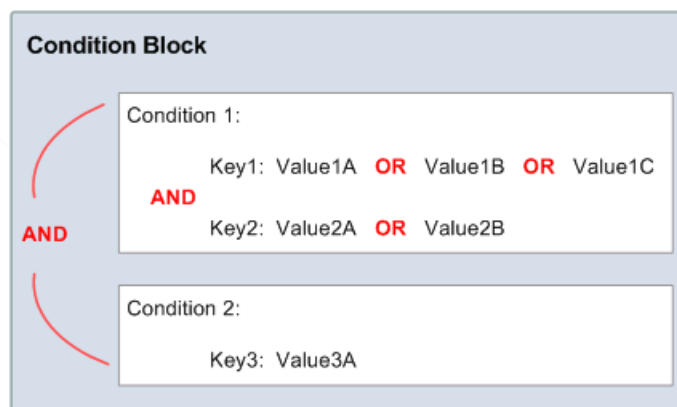
you want to let John use it only if some particular numeric value *foo* equals either A or B, and another numeric value *bar* equals C. Then you would create a condition block that looks like the following figure.



Let's say you also want to restrict John's access to after January 1, 2009. Then you would add another condition, `DateGreaterThan`, with a date equal to January 1, 2009. The condition block would then look like the following figure.



As illustrated in the following figure, we always apply a logical **AND** to the conditions within a condition block, and to the keys within a condition. We always apply a logical **OR** to the values for a single key. All conditions must be met to return an allow or an explicit deny decision. If a condition isn't met, the result is a default deny.



As mentioned, AWS defines the conditions and keys you can use (for example, one of the keys is `aws:CurrentTime`, which lets you restrict access based on the date and time). The AWS service itself can also define its own service-specific keys. For a list of available keys, see [Available Keys \(p. 228\)](#).

For a concrete example that uses real keys, let's say you want to let John upload an object under the following three conditions:

- The time is after 12:00 noon on 8/16/2010
- The time is before 3:00 p.m. on 8/16/2010
- The request comes from an IP address within the 192.168.176.0/24 range or the 192.168.143.0/24 range

Your condition block has three separate conditions, and all three of them must be met for John to have access to your bucket .

The following shows what the condition block looks like in your policy.

```
"Condition" : {
  "DateGreaterThan" : {
    "aws:CurrentTime" : "2009-04-16T12:00:00Z"
  }
  "DateLessThan": {
    "aws:CurrentTime" : "2009-04-16T15:00:00Z"
  }
  "IpAddress" : {
    "aws:SourceIp" : [ "192.168.176.0/24" , "192.168.143.0/24" ]
  }
}
```

Available Keys

AWS provides a set of common keys supported by all AWS products that adopt the access policy language for access control. These keys are:

- `aws:CurrentTime`—For date/time conditions (see [Date Conditions \(p. 230\)](#))
- `aws:SecureTransport`—Boolean representing whether the request was sent using SSL (see [Boolean Conditions \(p. 230\)](#))
- `aws:SourceIp`—The requester's IP address, for use with IP address conditions (see [IP Address \(p. 231\)](#))
- `aws:UserAgent`—Information about the requester's client application, for use with string conditions (see [String Conditions \(p. 229\)](#))
- `aws:EpochTime`—Number of seconds since epoch.
- `aws:Referer`—Same the the HTTP *referer* field.

The key names are case insensitive. For example, `aws:CurrentTime` is equivalent to `AWS:currenttime`.



Note

If you use `aws:SourceIp`, and the request comes from an Amazon EC2 instance, we evaluate the instance's public IP address to determine if access is allowed.

Each AWS service that uses the access policy language might also provide service-specific keys. For a list of any service-specific keys you can use, see [Special Information for Amazon S3 Policies \(p. 233\)](#).

Condition Types

These are the general types of conditions you can specify:

- String
- Numeric
- Date and time
- Boolean
- IP address
- Amazon Resource Name (ARN)

String Conditions

String conditions let you constrain using string matching rules. The actual data type you use is a string.

Condition	Description
<code>StringEquals</code>	Strict matching Short version: <code>streq</code>
<code>StringNotEquals</code>	Strict negated matching Short version: <code>strneq</code>
<code>StringEqualsIgnoreCase</code>	Strict matching, ignoring case Short version: <code>streqi</code>
<code>StringNotEqualsIgnoreCase</code>	Strict negated matching, ignoring case Short version: <code>strneqi</code>
<code>StringLike</code>	Loose case-insensitive matching. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string. Short version: <code>strl</code>
<code>StringNotLike</code>	Negated loose case-insensitive matching. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string. Short version: <code>strnl</code>

Numeric Conditions

Numeric conditions let you constrain using numeric matching rules. You can use both whole integers or decimal numbers. Fractional or irrational syntax is not supported.

Condition	Description
<code>NumericEquals</code>	Strict matching Short version: <code>numeq</code>
<code>NumericNotEquals</code>	Strict negated matching Short version: <code>numneq</code>

Condition	Description
NumericLessThan	"Less than" matching Short version: numlt
NumericLessThanEquals	"Less than or equals" matching Short version: numlteq
NumericGreaterThan	"Greater than" matching Short version: numgt
NumericGreaterThanEquals	"Greater than or equals" matching Short version: numgteq

Date Conditions

Date conditions let you constrain using date and time matching rules. You must specify all date/time values with one of the W3C implementations of the ISO 8601 date formats (for more information, go to <http://www.w3.org/TR/NOTE-datetime>). You use these conditions with the `aws:CurrentTime` key to restrict access based on request time.



Note

Wildcards are not permitted for date conditions.

Condition	Description
DateEquals	Strict matching Short version: dateeq
DateNotEquals	Strict negated matching Short version: dateneq
DateLessThan	A point in time at which a key stops taking effect Short version: datelt
DateLessThanEquals	A point in time at which a key stops taking effect Short version: datelteq
DateGreaterThan	A point in time at which a key starts taking effect Short version: dategt
DateGreaterThanEquals	A point in time at which a key starts taking effect Short version: dategteq

Boolean Conditions

Condition	Description
Bool	Strict Boolean matching

IP Address

IP address conditions let you constrain based on IP address matching rules. You use these with the `aws:SourceIp` key. The value must be in the standard CIDR format (for example, 10.52.176.0/24). For more information, go to [RFC 4632](#).

Condition	Description
<code>IpAddress</code>	Whitelisting based on the IP address or range
<code>NotIpAddress</code>	Blacklisting based on the IP address or range

Amazon Resource Name (ARN)

Amazon Resource Name (ARN) conditions let you constrain based on ARN matching rules. The actual data type you use is a string.

Condition	Description
<code>ArnEquals</code>	Strict matching for ARN Short version: <code>arneq</code>
<code>ArnNotEquals</code>	Strict negated matching for ARN Short version: <code>arnneq</code>
<code>ArnLike</code>	Loose case-insensitive matching of the ARN. Each of the six colon-delimited components of the ARN is checked separately and each can include a multi-character match wildcard (*) or a single-character match wildcard (?). Short version: <code>arnl</code>
<code>ArnNotLike</code>	Negated loose case-insensitive matching of the ARN. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string. Short version: <code>arnnl</code>

```
{
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "AWS": "210987654321"
    },
    "Action": "sqs:SendMessage",
    "Resource": "arn:aws:sqs:us-east-1:01234567891:your_queue_xyz",
    "Condition": {
      "ArnEquals": {
        "aws:SourceArn": "arn:aws:sns:us-east-1:123456789012:your_special_top
ic_1"
      }
    }
  }]
}
```

Supported Data Types

This section lists the set of data types the access policy language supports. The language doesn't support all types for each policy element (for the supported data types for each element, see [Element Descriptions](#) (p. 223)).

The access policy language supports the following data types:

- Strings
- Numbers (Ints and Floats)
- Boolean
- Null
- Lists
- Maps
- Structs (which are just nested Maps)

The following table maps each data type to the serialization. Note that all policies must be in UTF-8. For information about the JSON data types, go to [RFC 4627](#).

Type	JSON
String	String
Integer	Number
Float	Number
Boolean	true false
Null	null
Date	String adhering to the W3C Profile of ISO 8601
IpAddress	String adhering to RFC 4632
List	Array
Object	Object

Special Information for Amazon S3 Policies

The following list describes the restrictions on Amazon S3 policies:

- The maximum size of a policy is 20 KB
- The value for *Resource* must be prefixed with the bucket name or the bucket name and a path under it (bucket/*). If only the bucket name is specified, without the trailing /*, the policy applies to the bucket.
- Each policy must have a unique policy ID (*Id*)
- Each statement in a policy must have a unique statement ID (*sid*)
- Each policy must cover only a single bucket and resources within that bucket (when writing a policy, don't include statements that refer to other buckets or resources in other buckets)

Glossary

100-continue	A method that enables a client to see if a server can accept a request before actually sending it. For large <code>PUT</code> s, this can save both time and bandwidth charges.
account	AWS account associated with a particular developer.
authentication	The process of proving your identity to the system.
bucket	A container for objects stored in Amazon S3. Every object is contained within a bucket. For example, if the object named <code>photos/puppy.jpg</code> is stored in the <code>johnsmith</code> bucket, then it is addressable using the URL <code>http://johnsmith.s3.amazonaws.com/photos/puppy.jpg</code>
canned access policy	A standard access control policy that you can apply to a bucket or object. Options include: private, public-read, public-read-write, authenticated-read.
canonicalization	The process of converting data into a standard format that will be recognized by a service such as Amazon S3.
consistency model	The method through which Amazon S3 achieves high availability, which involves replicating data across multiple servers within Amazon's data centers. After a "success" is returned, your data is safely stored. However, information about the changes might not immediately replicate across Amazon S3.
delete marker	A Delete Marker is an object with a key and version ID but it has no content. Amazon S3 inserts Delete Markers automatically into buckets when a <code>DELETE</code> request is executed on an object in a bucket where Versioning is enabled.
default deny	A permission that defaults to DENY.
key	The unique identifier for an object within a bucket. Every object in a bucket has exactly one key. Since a bucket and key together uniquely identify each object, Amazon S3 can be thought of as a basic data map between "bucket + key" and the object itself. Every object in Amazon S3 can be uniquely addressed through the combination of the web service endpoint, bucket name, and key, as in <code>http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl</code> , where "doc" is the name of the bucket, and "2006-03-01/AmazonS3.wsdl" is the key.
metadata	The metadata is a set of name-value pairs that describe the object. These include default metadata such as the date last modified and standard HTTP metadata such as Content-Type. The developer can also specify custom metadata at the time the Object is stored.

null object	A null object is one whose version ID is <code>null</code> . Amazon S3 adds a null object to a bucket when version for that bucket is suspended. It is possible to have only one null object for each key in a bucket.
object	The fundamental entities stored in Amazon S3. Objects consist of object data and metadata. The data portion is opaque to Amazon S3.
part	A contiguous portion of object's data. The Amazon S3 multipart upload feature allows larger objects to be uploaded in parts. These object parts can be uploaded independently and in any order. If transmission of any part fails, that part can be retransmitted without affecting other parts. After all the object parts are uploaded, Amazon S3 assembles these parts and creates the object.
service endpoint	The host and port with which you are trying to communicate within the destination URL. For virtual hosted-style requests, this is <code>mybucket.s3.amazonaws.com</code> . For path-style requests, this is <code>s3.amazonaws.com</code>
Versioning	Every object in Amazon S3 has a key and a version ID. Objects with the same key but different version IDs can be stored in the same bucket. Versioning is enabled at the bucket layer using <code>PUT Bucket versioning</code> .

Document Conventions

This section lists the common typographical and symbol use conventions for AWS technical publications.

Typographical Conventions

This section describes common typographical use conventions.

Convention	Description/Example
Call-outs	<p>A call-out is a number in the body text to give you a visual reference. The reference point is for further discussion elsewhere.</p> <p>You can use this resource regularly. 1</p>
Code in text	<p>Inline code samples (including XML) and commands are identified with a special font.</p> <p>You can use the command <code>java -version</code>.</p>
Code blocks	<p>Blocks of sample code are set apart from the body and marked accordingly.</p> <pre># ls -l /var/www/html/index.html -rw-rw-r-- 1 root root 1872 Jun 21 09:33 /var/www/html/index.html # date Wed Jun 21 09:33:42 EDT 2006</pre>
Emphasis	<p>Unusual or important words and phrases are marked with a special font.</p> <p>You <i>must</i> sign up for an account before you can use the service.</p>
Internal cross references	<p>References to a section in the same document are marked.</p> <p>For more information, see Document Conventions (p. 236).</p>

Convention	Description/Example
Logical values, constants, and regular expressions, abstracta	A special font is used for expressions that are important to identify, but are not code. If the value is <code>null</code> , the returned response will be <code>false</code> .
Product and feature names	Named AWS products and features are identified on first use. Create an <i>Amazon Machine Image</i> (AMI).
Operations	In-text references to operations. Use the <code>GetHITResponse</code> operation.
Parameters	In-text references to parameters. The operation accepts the parameter <i>AccountID</i> .
Response elements	In-text references to responses. A container for one <code>CollectionParent</code> and one or more <code>CollectionItems</code> .
Technical publication references	References to other AWS publications. If the reference is hyperlinked, it is also underscored. For detailed conceptual information, refer to the <i>Amazon Mechanical Turk Developer Guide</i> .
User entered values	A special font marks text that the user types. At the password prompt, type MyPassword .
User interface controls and labels	Denotes named items on the UI for easy identification. On the File menu, click Properties .
Variables	When you see this style, you must change the value of the content when you copy the text of a sample to a command line. % ec2-register <i><your-s3-bucket></i> /image.manifest See also the following symbol convention.

Symbol Conventions

This section describes the common use of symbols.

Convention	Symbol	Description/Example
Mutually exclusive parameters	(Parentheses and vertical bars)	Within a code description, bar separators denote options from which one must be chosen.
		<code>% data = hdfread (start stride edge)</code>
Optional parameters XML variable text	[square brackets]	Within a code description, square brackets denote completely optional commands or parameters.
		<code>% sed [-n, -quiet]</code>
		Use square brackets in XML examples to differentiate them from tags.
Variables	<arrow brackets>	Within a code sample, arrow brackets denote a variable that must be replaced with a valid value.
		<code>% ec2-register <your-s3-bucket>/image.manifest</code>

Index

Symbols

100-continue, 45

A

- access control, 141
- access control policies, 213
- access logs, 203
- access policy
 - REST, 38
 - SOAP, 63
- Adobe Flash, 46
- Amazon DevPay, 192
- API, 19
 - REST, 19, 25
 - SOAP, 19, 61
- audience, 1
- authentication, 141
 - debugging, 36
 - REST, 27
 - SOAP, 62
- authentication header, 28

B

- billing, 19
- BitTorrent, 189
 - charges, 189
 - publishing, 191
 - retrieving objects, 190
- browser uploads, 46
- buckets, 7, 65
 - access control, 72
 - billing, 72
 - configuration, 67
 - location selection, 67
 - notifications of events, 180
 - restrictions, 66
 - virtual hosting, 40

C

- CanonicalizedAmzHeaders element, 30
- changes, 4
- charges, 19
- components, 7
- concepts
 - API, 19
 - buckets, 7
 - components, 7
 - keys, 7
 - objects, 7
 - operations, 18
 - REST API, 19
 - SOAP API, 19

- configuring logging, 201
- consistency model, 8
- copying objects, overview, 127
- costs, 19

D

- data model, 8
- delimiter, 133
- DevPay, 192
- DNS, 187
- DNS routing, 184, 185, 186

E

- elements
 - REST, 26, 76, 77, 79, 80, 84, 94, 109, 115
 - SOAP, 62
- errors, 197
 - details, 197
 - isolation, 198
 - messages, 197
 - response, 197
 - REST response, 196
 - SlowDown, 198
 - SOAP response, 197
- events
 - setting up notification of, 180

F

- features, 6
- file size, maximum, 6
- Flash, Adobe, 46

G

- guide organization, 2

H

- HTTP user agents, 44

I

- introduction, 6

J

- Java, 187
- JVM cache, 187

K

- keys, 7
 - listing hierarchically, 133
 - multi-page results, 133
 - using, 132

L

- listing keys, hierarchically, 133

- location constraints, 67
- logs, 200
 - best effort delivery, 204
 - changing settings, 203
 - configuration, 201
 - delivery, 203
 - format, 204
 - setting up, 208

M

- metadata, using, 75
- model, 8

N

- notifications
 - setting up, 180

O

- object size, maximum, 6
- objects, 7
 - copying, 127
 - getting, 120
 - using, 74
- operations, 18
- organization of guide, 2
- overview, 6

P

- pagination, 133
- paying, 19
- performance optimization, 188
- PHP virtual machine, 187
- policies, 213
- POST, 46
- prefix, 133

R

- redirection, 44
 - permanent, 186
 - request, 184
 - temporary, 185
- Reduced Redundancy Storage, 157
- referrer, 204
- Region, 62, 67
- request redirection, 184
 - access policy, 44
- request routing, 184
- resources, related, 2
- REST
 - access policy, 38
 - API, 25
 - authentication, 27
 - examples, 31
 - header, 28
 - debugging authentication, 36

- elements, 26, 76, 77, 79, 80, 84, 94, 109, 115
- POST, 46
 - StringToSign, 31
 - time stamp, 31
- restrictions, 66
- routing, 184
 - DNS, 184
- RRS, 157

S

- server access logs, 200, 203
- SetObjectAccessControlPolicy
 - SOAP, 189
- shared queues, 213
- signature, creating, 36
- size, object, 6
- SOAP
 - access policy, 63
 - API, 61
 - authentication, 62
 - elements, 62
 - error response, 197
- storage limit, 6
- StringToSign, 31
- system metadata, 75

T

- TCP optimization, 188
- time stamp, 31
- TTLs, clients, 187

U

- uploads, browser, 46
- user metadata, 75

V

- versioning, 159
- virtual hosted buckets, 40
- virtual machines, 187