Amazon S3

Developer Guide (API Version 2006-03-01)

Amazon S3: Developer Guide (API Version 2006-03-01)

Published 2006-June-12

Table of Contents

Welcome to Amazon S3

Thank you for your interest in Amazon S3. This developer guide describes the S3 interfaces and functionality in detail.

You may find the following resources useful as companions to this guide:

- Resource Center: Browse the resource center for code samples, the Getting Started Guide, release notes and more information. Subscribe to RSS feeds or set up email watches to receive alerts about the latest S3 developments.
- Amazon S3 WSDL
- Developer Forum
- The Amazon S3 Homepage

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 Developer Forum.

Amazon S3 Overview

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 websites. The service aims to maximize benefits of scale and to pass those benefits on to developers.

Service Highlights

Amazon S3 is intentionally built with a minimal feature set. The focus is on simplicity and robustness.

- Write, read, and delete objects containing from 1 byte to 5 gigabytes of data each, with accompanying metadata. The number of objects you can store is unlimited.
- A straightforward flat object store model, where each object is stored and retrieved via a unique developer-assigned key.
- Authentication mechanisms are provided to ensure that data is kept secure from unauthorized access. Objects can be made private or public, rights granted to specific users.
- Standards-based REST and SOAP interfaces designed to work with any Internet-development toolkit.

Amazon S3 Design Requirements

Amazon S3 is based on the idea that quality Internet-based storage should be taken for granted. It helps free developers from worrying about where they are going to store data, whether it will be safe and secure, the costs associated with server maintenance, or whether they have enough storage available. The functionality is simple and robust: Store any amount of data inexpensively and securely, while ensuring that the data will always be available when you need it. Amazon S3 enables developers to focus on innovating with data, rather than figuring out how to store it.

Amazon built S3 to fulfill the following design requirements:

- *Scalable:* Amazon S3 must scale in terms of storage, request rate, and users to support an unlimited number of web-scale applications. It uses scale as an advantage: Adding nodes to the system increases, not decreases, its availability, speed, throughput, capacity, and robustness.

- *Reliable:* Store data durably, with 99.99% availability. There can be no single points of failure. All failures must be tolerated or repaired by the system without any downtime.

- *Fast:* Amazon S3 must be fast enough to support high-performance applications. Server-side latency must be insignificant relative to Internet latency. Any performance bottlenecks can be fixed by simply adding nodes to the system.

- *Inexpensive:* Amazon S3 is built from inexpensive commodity hardware components. As a result, frequent node failure is the norm and must not affect the overall system. It must be hardware-agnostic, so that savings can be captured as Amazon continues to drive down infrastructure costs.

- *Simple:* Building highly scalable, reliable, fast, and inexpensive storage is difficult. Doing so in a way that makes it easy to use for any application anywhere is more difficult. Amazon S3 must do both.

A forcing-function for the design was that a single Amazon S3 distributed system must support the needs of both internal Amazon applications and external developers of any application. This means that it must be fast and reliable enough to run the Amazon.com website, yet simple and generic enough that any developer can use it for any data storage need.

Amazon S3 Design Principles

Amazon used the following principles of distributed system design to meet S3 requirements:

- *Decentralization:* Use fully decentralized techniques to remove scaling bottlenecks and single points of failure.
- *Asynchrony:* The system makes progress under all circumstances.
- *Autonomy:* The system is designed such that individual components can make decisions based on local information.
- *Local responsibility:* Each individual component is responsible for achieving its consistency; this is never the burden of its peers.
- *Controlled concurrency:* Operations are designed such that no or limited concurrency control is required.
- *Failure tolerant:* The system considers the failure of components to be a normal mode of operation, and continues operation with no or minimal interruption.
- *Controlled parallelism:* Abstractions used in the system are of such granularity that parallelism can be used to improve performance and robustness of recovery or the introduction of new nodes.
- *Decompose into small well-understood building blocks:* Do not try to provide a single service that does everything for every one, but instead build small components that can be used as building blocks for other services.
- *Symmetry:* Nodes in the system are identical in terms of functionality, and require no or minimal node-specific configuration to function.
- *Simplicity:* The system should be made as simple as possible (- but no simpler).

Core Concepts and Operations

If you have already read the Amazon S3 Getting Started Guide (available in the S3 Resource Center), this section will not contain any new information for you. If you haven't read the Getting Started Guide, we recommend you take a look: It contains a handy tutorial-style overview of all the common Amazon S3 concepts and functionality, and a walkthrough of sample code.

This section contains the following topics:

- Buckets
- Objects
- Keys
- Operations

Buckets

A bucket is simply a container for objects stored in Amazon S3. Every object is contained within 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.

See the Using Buckets section for more information about buckets.

Objects

Objects are the fundamental entities stored in Amazon S3. Objects are composed 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 may also specify custom metadata at the time the Object is stored.

Keys

A key is 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 Service endpoint, bucket name, and key, as in http://s3.amazonaws.com/doc/2006-03-01/AmazonS3.wsdl, where "doc" is the name of the bucket, and "2006-03-01/AmazonS3.wsdl" is the key.

Operations

Amazon S3 offers APIs in REST and SOAP. The most common operations you'll execute through the API include the following:

- *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 an all other functionality are described in detail later in this guide.

Paying for Amazon S3

This section describes how Amazon will charge you for use of Amazon S3.

Pricing for Amazon S3 is based on the idea that you should not have to worry about planning ahead for the amount of storage your application will need. Most storage providers force you to purchase a pre-determined amount of storage and network transfer capacity: If you go over that capacity, your service is shut off or you are charged high overage fees, and if you stay under that capacity you nonetheless pay as if you had 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 still enjoying the cost advantages of Amazon's large infrastructure scale.

Before you can store 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 will automatically be charged for that month's usage.

This section contains the following topics:

Amazon S3 Pricing

Amazon S3 pricing is based on your actual usage. Your usage is measured to the nearest byte, and your charges are rounded up to the nearest cent.

S3 charges you for the following types of usage:

- *Storage Used:* $0.15 per GB-Month of storage. This fee applies to all object data and metadata stored in buckets that you created under your account.

  It does not matter who created the objects in your buckets, so think twice before you give somebody the right to write objects to your bucket!
- *Network Data Transferred:* $0.20 per GB of data transferred. This fee applies anytime data is read from or written to one of your buckets. It does not matter who is reading or writing the data, so consider this when you give public access to one of your objects that may become popular.

How Storage Charges are Calculated

You are charged for the amount of storage used by your objects and the amount of time that storage is used. Amazon S3 bills you based on your GB-Month usage of storage. GB-Months can also be thought of as your average storage used throughout the month: If you store 1 GB of data for a month, that's 1 GB-Month of usage, but if you store 0 GB for the first half of the month and 2 GB for the second half of the month, that's still just 1 GB-Month total usage.

At least twice a day, Amazon S3 measures the amount of storage taken up by all the objects in all of your buckets. Your total storage is multiplied by the amount of time past since your storage use was last calculated. For accuracy, this measurement is taken in units of Byte-Hours. These measurements are recorded twice a day for the entire month. At the end of the month, we add up the total number of Byte-Hours associated with your account, convert the result to GB-Month units, and apply the storage fee to come up with an amount to be billed.

Storage Billing Example

Let's say you subscribe to Amazon S3 on March 14, and soon after create a bucket for your account (there is no charge for this). For the remainder of the month you add and remove the following data to/from your bucket:

Date	Operation	Amount of Data
March 18, 13:00	Write a New Object	+2 GB
March 20, 9:00	Write a New Object	+50 KB
March 20, 11:00	Write a New Object	+2 GB
March 28, 13:00	Delete an Object	-2 GB

How is your bill calculated?Assume Amazon S3 checks your storage at noon and midnight every day (the actual times may differ, but your storage will be checked at least twice a day). At every checkpoint, Amazon S3 records the number of Byte-Hours used since the last checkpoint:

Checkpoint Time	Amount of Storage Used	Amount of Byte-Hours Recorded
March 14, 12:00	0	0
March 15, 00:00	0	0
...	etc..., every 12 hours, until	...
March 19, 00:00	2,147,483,648 Bytes	12 Hours * 2,147,483,648 Bytes = 25,769,803,776 Byte-Hours
March 19, 12:00	2,147,483,648 Bytes	12 Hours * 2,147,483,648 Bytes = 25,769,803,776 Byte-Hours
March 20, 00:00	2,147,483,648 Bytes	12 Hours * 2,147,483,648 Bytes = 25,769,803,776 Byte-Hours
March 20, 12:00	4,295,018,496 Bytes	12 Hours * 4,295,018,496 Bytes = 51,540,221,952 Byte-Hours

Checkpoint Time	Amount of Storage Used	Amount of Byte-Hours Recorded
March 21, 00:00	4,295,018,496 Bytes	12 Hours * 4,295,018,496 Bytes = 51,540,221,952 Byte-Hours
...	etc..., every 12 hours, until	...
March 29, 00:00	2,147,534,848 Bytes	12 Hours * 2,147,534,848 Bytes = 25,770,418,176 Byte-Hours
March 29, 12:00	2,147,534,848 Bytes	12 Hours * 2,147,534,848 Bytes = 25,770,418,176 Byte-Hours
...	etc..., every 12 hours, until the end of the month.	...

At the end of the month two data points will have been recorded for every day from March 14 through March 31. S3 will add up all these data points to calculate your bill. In the example above, your total usage would be 1,133,886,111,744 Byte-Hours, or equivalently 1.419 GB-Months, for a total storage charge of $0.22 for the month of March.

All of the raw data used to calculate your bill is available for you to view. See the section Tracking Your Usage to learn more.

How Network Data Transfer Charges are Calculated

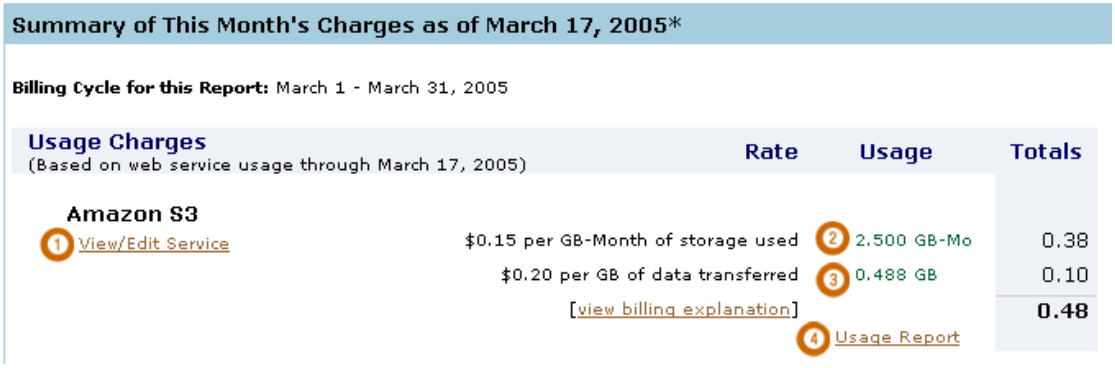
You are charged for the amount of data related to your buckets that is transferred into and out of the Amazon S3 service. Every time a get object, put object, or list bucket request is made to one of your buckets, Amazon S3 monitors the number of bytes transferred in the request and response. You are charged for transfer of all object data, metadata, and keys, but no for protocol overhead. The same fee applies whether the data is coming into Amazon S3 or leaving the service.

If Amazon kept a usage record for every request to your bucket, the amount of data generated could be huge for high-volume users. So Amazon S3 aggregates network usage over the course of an hour-long period and records your hourly usage at the end of each period. At the end of the month the total number of bytes transferred is totaled and the network transfer fees are applied to come up with your monthly network transfer charges.

All of the raw hourly data used to calculate your bill is available for you to view. For details, please see the topic Tracking Your Usage.

Viewing Your Bill

You can view your charges for the current billing period at any time through the AWS portal at aws.amazon.com. Login to your AWS account, and click "Account Activity" under "Your Web Services Account." You will see a summary of charges for all of your subscribed web services. Keep in mind that the charges displayed only represent activity to-date in the billing period. If you continue to use storage or network resources in the remainder of the billing period, the amount charged will be a greater than what you may see on the bill earlier in the month.



1 View/Edit Service: Click this link to view or change the status of your Amazon S3 subscription.

2 Storage Usage: The number of GB-months of storage use billed to your account thus far in the billing period. Note that this value is dependent on the amount of data you have stored as well as where you are in the billing cycle.

3 Data Transfer Usage: The total number of GB data transfer use billed to your account thus far in the billing period.

4 Usage Report: Click this link to view detailed usage data that is the basis for your bill.

Tracking Your Usage

You can view usage statistics for your Amazon S3 buckets through the AWS portal at aws.amazon.com. Log into your AWS account and click "Usage Report" under Your Web Services Account.

The data available for viewing in a usage report is organized according to usage type and operation. The usage type is the category of usage data that you want to report. The data under each usage type is further categorized by the operation or type of storage that is associated with each data point in the report.

Amazon S3 reports the following usage types:

- *`TimedStorage-ByteHrs:`* This contains records of the amount of storage you've used over time. At least twice a day, S3 looks at how much storage is being used by all the objects in all of your buckets, multiplies that by the number of hours since the last checkpoint, and records the result under the TimedStorage-ByteHrs usage type. This is the same data that S3 uses to calculate your storage fees at the end of the billing period. The data is provided in units of byte-hours.

- *`AverageStorage-Bytes:`* This usage type contains another, more intuitive view of your storage usage. This is the average total storage used by all the objects in all your buckets per day. The data is provided in units of bytes. This data is directly calculated from the data stored with the TimedStorage-ByteHrs usage type. If you're just interested in knowing how much storage you're using day-to-day, this is an easier view than trying to convert the Byte-Hours data in TimedStorage to a more intuitive number. Data for this usage type is only available in daily granularity.

- *`Network-Bytes:`* This contains records of network data transfer associated with your account. This is the same data used to calculate your charges for network use at the end of the billing period. Every time a request is received to put an object, get an object, or list a bucket, the amount of network traffic involved in transmitting the object data, metadata, or keys is recorded here. You can view only the network usage associated with one of these operations by specifying the operation of choice before generating the usage report.

- *`Request:`* This usage type contains information about the number of requests received for various common Amazon S3 operations. This data is provided for information purposes only and does not impact your bill. You can choose to view the number of requests to put an object, get an object, delete an object, or list a bucket related to your account.

Using Amazon S3

This section discusses Amazon S3 concepts that apply regardless of the API style you choose. The following topics are included:

How Amazon S3 Organizes Objects

Amazon S3 is used to store objects - that's what it is all about. An object has four parts: value, key, metadata, and an access control policy. Objects are stored in buckets. This section describes these things, and how they are related to each other, in the following topics:

- Object Value
- Buckets
- Keys
- Metadata

Buckets

Buckets Organize the Amazon S3 Namespace

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 name your objects whatever you like, but bucket names themselves must be unique across all of Amazon S3.

An analogy with Internet domain names is useful for understanding the namespace aspect of buckets. Consider the URL `http://johnsmith.net/homepage.html`. Only one person or entity can have the domain name `johnsmith.net` registered at a time because Internet domain names share a single global namespace. Similarly, only one Amazon S3 developer can own a bucket name at one time. The owner of `johnsmith.net` has control over the names of the resources available under his domain. Within `johnsmith.net`, there can be no naming conflicts with any other Internet resource because the path name is scoped by the domain name. Similarly, once you create a uniquely named bucket in Amazon S3, you can organize and name the objects contained in your bucket any way you'd like.

The similarities between buckets and domain names is not a coincidence: In fact, 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` was stored in the Amazon S3 bucket `johnsmith` its address would be `http://johnsmith.s3.amazonaws.com/homepage.html`. See the Virtual Hosted Buckets section for more information.

Bucket Restrictions and Limitations

Since the namespace for bucket names is global, you must select a bucket name that is not currently owned by somebody else. A bucket is owned by the developer (identified by AWS Access Key ID) that created it. Each developer is limited to owning 100 buckets at a time. Bucket ownership is not transferable, but if a bucket is empty, it can be deleted and its name can be reused.

Since buckets correspond to domain names, bucket names may only contain the characters A-Z, a-z, 0-9, '_', '.', and '-'. Bucket names must be at least 3 characters long, and at most 255 characters long. Buckets with names containing upper-case characters are not accessible using the virtual hosting method.

There is no limit to the number of objects that one bucket can hold, and no impact on performance when using many buckets versus just a few buckets. You could reasonably store all of your objects in a single bucket, or organize them across several different buckets.

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

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 the Using Access Control section.

Billing and Reporting of Buckets

Fees for object storage and network data transfer are always billed to the owner of the bucket containing the object in question.

The reporting tools available at the Amazon Web Services developer portal organize your Amazon S3 usage reports by bucket.

Object Value

The value is the meat of the object; it is the real content that you are storing. From the perspective of Amazon S3, the object value can be any sequence of bytes, except that its size is limited to 5 GB.

Keys

The key is the handle that you assign to an object so that you can fetch it later. No two objects in a bucket may have the same key. A key is a sequence of Unicode characters whose UTF-8 encoding is at most 1024 bytes long.

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 delimit hierarchy, you can use the list operation to select and browse keys hierarchically. This idea is similar to how groups of files are organized into directories in a filesystem. See the Listing Keys section for more information.

Keys are often given a suffix that indicates something about the type of data in the object, though this is not required. For example, often keys will use a suffix of ".jpg" to indicate that an object is an image.

Metadata

An Amazon S3 object's metadata is a set of key-value pairs associated with the object. The idea is that you store information about the object in its metadata. There are two kinds of metadata: system metadata, and user metadata.

System metadata means something to Amazon S3, and is sometimes computed by Amazon S3. System metadata behavior depends on which API (REST or SOAP) you are using.

User metadata entries are specified by you, the user of Amazon S3. Amazon S3 does not interpret this metadata - it simply stores it, and then passes it back when you ask for it. Metadata keys and values can be any length, but must conform to UTF-8.

Metadata Size

For both REST and SOAP requests to S3, user metadata size is limited to 2k bytes.

Metadata Interoperability

In REST, user metadata keys must begin with "x-amz-meta-" to distinguish them as custom HTTP headers. When such metadata is retrieved via SOAP, the x-amz-meta- prefix is removed. Similarly,

metadata stored via SOAP will have x-amz-meta- added as a prefix when it is retrieved via REST, except when the metadata fits an HTTP standard header as would be the case with "Content-Type" metadata.

When metadata is retrieved via the REST API, 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 and instead the "x-amz-missing-meta" header is returned with a value of the number of such unprintable metadata entries.

S3 Application Programming Interfaces (API)

The Amazon S3 architecture is designed to allow a plethora of different programming interfaces to store and retrieve objects. Any of the different supported interfaces can be used for any buckets and keys. Our goal is to support all functionality with each different interface, but, unfortunately, we will sometimes fall short. For example, in the REST interface, metadata is returned in HTTP headers, but because we only support HTTP requests of up to 4k (not including the body), you are restricted in how much metadata you can supply in that manner.

Amazon S3 provides both a REST and a SOAP interface.

- REST Interface
- SOAP Interface

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.

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 (available at http://s3.amazonaws.com/doc/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 S3.

Authentication and Access Control

S3 considers two things when deciding whether or not the request should be allowed to proceed. The first is who is making the request - this is determined using Amazon Web Services Authentication. The second is the access control policy on the bucket and object targeted by the request.

This section contains the following topics:

- Authentication
- Access Control Policy

Authentication

Authentication is the process of verifying the identity of the sender of a request. Authentication ensures that only those requests made by the person or party responsible for the AWS account specified in the request are accepted and allowed to access private data. Authentication also allows Amazon to track the usage of Amazon S3 on a per-request basis. This enables Amazon to charge and bill for use of Amazon S3.

A principal is an entity that can make a request to Amazon S3. Amazon S3 decides which principal is making a request based on the authentication information provided with the request.

If no authentication is provided, then the request is an considered anonymous. The principal for an anonymous request is called the anonymous principal.

Principals identify themselves for authentication purposes by specifying the principal's AWS Access Key ID as part of the authentication information. In addition, the request must include some kind of evidence that the principal knows the AWS Secret Access Key corresponding to the specified Access Key ID, for example by using the secret key to sign part of the request.

You are assigned an AWS Access Key ID and Secret Access Key when you register for an AWS account at http://aws.amazon.com. If you already have an AWS account, start at this page to view your account information, including your AWS Access Key ID and Secret Access Key. Your account must be subscribed to Amazon S3 before you can authenticate to the service.

Important

Important: Once assigned a Secret Access Key and Access Key ID, you have a responsibility to keep the Secret Access Key secret to prevent it from being used to make unauthorized AWS requests. Do not share your Secret Access Key outside your organization, even if an inquiry appears to come from Amazon.com. No one who legitimately represents Amazon will ever ask you for your Secret Access Key.

The accepted methods of authentication vary depending on the API you are using. To learn more, consult the reference section for each API.

Access Control Policy

Each bucket and object in S3 has an access control list (ACL) that defines its access control policy. When a request is made, S3 determines the principal making the request, and then checks the access control list to see if that principal is authorized to make the request. If the ACL contains an entry authorizing the the principal in question to make this request, the request is allowed to proceed, otherwise an error is returned.

An access control list is a sequence of grants. A grant is composed of one grantee and one permission. In this release, access control lists only confer permissions, they do not deny them.

This section contains the follwing API-independent information about S3 access control:

- Grantees
- Permissions
- Using Access Control Lists

Grantees

The grantee in a grant may be a user or a group.

There are two ways to identify a user grantee when specifying a grant, but internally S3 always stores user grantees in the same canonical way no matter how the user in the grant was originally specified. When you read an ACL, the user grantee will always be displayed in the canonical format, represented by the CanonicalUser XML element.

The other type of grantee is a group. The only groups available are those pre-defined by S3. In this release of S3, you cannot create your own group.

User Grantee: Amazon Customer by Email

You can grant access to an individual by specifying the email address that individual uses to log into their Amazon.com retail web site account. The individual must be registered as an Amazon.com customer, but does not have to be registered as an AWS customer. The XML format for specifying a grantee by their Amazon customer email address is:

```
<Grantee xsi:type="AmazonCustomerByEmail">
  <EmailAddress>chriscustomer@email.com</EmailAddress>
</Grantee>
```

Grants by email address are internally converted to the CanonicalUser representation when you write the ACL in question. Therefore, granting by email address confers permissions to the unique Amazon customer associated with the specified email address at the time of the call. In particular, the grantee can change his or her email address without affecting existing S3 permissions.

To ensure that access is granted only to the intended customer, adding a grantee by email address will only work if there is exactly one Amazon account that corresponds to the specified email address. A request to store an ACL containing an email grantee will fail with the AmbiguousGrantByEmail error code if the grantee has multiple Amazon accounts with the same email address. What this means is that the customer you are attempting to grant to can log into the Amazon.com retail web site with same email address but multiple passwords. This situation typically arises when the user forget his or her password at some time in the past and accidentally created a new account with the same email address. In this rare case, the customer with multiple accounts should call Amazon.com customer service to have these accounts merged, or permission should be granted to that customer using the CanonicalUser representation obtained from an existing grant or the ListAllMyBuckets response.

User Grantee: Canonical Representation

The identity of a user grantee is always returned by S3 in the following canonical representation. The grantee may also be specified under this representation:

```
<Grantee xsi:type="CanonicalUser">
  <ID>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666e9</ID>
  <DisplayName>chriscustomer<DisplayName>
</Grantee>
```

- *ID:* An S3-assigned string that uniquely and permanently identifies the user.

- *DisplayName:* A string that identifies the user grantee to a human. The DisplayName is calculated for an Amazon customer as follows: If the customer has assigned him/herself a public nickname using the Amazon.com retail website, use it as the display name. Otherwise, use the first component of the customer's email address. This value is not unique, and its value may change over time. For example, if the customer changes their email address or updates their retail website nickname, the change will be reflected in their DisplayName in subsequent ACL responses. You can set or change your Amazon.com retail website nickname using the following link: http://s1.amazon.com/exec/varzea/account-access/settings/104-1775788-0244713?account-seller=1

When writing an ACL containing CanonicalUser elements, the DisplayName element is ignored.

User Grantee: Owner

Every bucket and object in S3 has an attribute called "owner" associated with it. The owner of a bucket or object is permanently set to the user that created the bucket or object. The only way to change the owner of a bucket is to delete the bucket and create it again under a different user identity. The only way to change the owner of an object is to overwrite the object using a different identity. The owner of a bucket or object is subject to the access control policy of that bucket or object just like everybody else, with two notable exceptions: The owner of a resource always has the ability to read and write the ACL of that resource, no matter what the associated ACL says. For example, as the owner of an object, you could remove yourself from the associated access control list, and find that you can no longer read the object's data and metadata. However, by virtue of being owner, you always have the right to re-grant yourself permissions to it. This policy prevents the situation where an object becomes "stranded," with nobody able to ever modify or even delete it.

Group Grantees

A group of users may be specified as a grantee under the following example XML representation:

```
<Grantee xsi:type="Group">
  <URI>http://acs.amazonaws.com/groups/global/AllUsers<URI>
</Grantee>
```

- `http://acs.amazonaws.com/groups/global/AllUsers:` All principals, whether they are anonymous or authenticated, are considered part of this group.

- `http://acs.amazonaws.com/groups/global/AuthenticatedUsers:` Every non-anonymous principal is considered part of the group. Note that permission granted by virtue of

this grant does not trump other access control considerations. For example, if a user is registered as an AWS developer, they may be part of this group, but if they have not subscribed to S3, the will still not be granted access.

- *URI:* This field specifies the URI of the group grantee. You may choose from two pre-defined groups for this value:

Permissions

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

- READ: When applied to a bucket, this 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, this grants permission to create, overwrite, and delete any object in the bucket. This permission is not supported for objects (it is reserved for future use).
- READ_ACP: Grants permission to read the access control policy (ACL and owner) 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. Note that granting this permission is equivalent to granting FULL_CONTROL, because the grant recipient can now make any whatever changes to the ACP he or she likes!
- FULL_CONTROL: This permission is short-hand for the union of READ, WRITE, READ_ACP, and WRITE_ACP permissions. It does not convey additional rights, and is provide only for convenience.

Using Access Control Lists

An ACL may contain up to 100 grants. If no ACL is provided at the time a bucket is created or an object written then a default ACL is created for you. The default ACL for new resources consists of a single grant that gives the owner of the resource (i.e. the principal making the request) FULL_CONTROL permission. Note that if you overwrite an existing object, the ACL for the existing object is always overwritten as well, and defaulted back to Owner/FULL_CONTROL if no explicit ACL is provided.

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

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

Regardless of which API you are using, the XML representation of an ACL stored in S3 (and returned when the ACL is read) is the same. In the example ACL below, the owner has the default FULL_CONTROL, the "filesinc" and "storagehelper" 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 xsi:type="CanonicalUser">
<ID>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666e9</ID>
        <DisplayName>chriscustomer</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
    <Grant>
      <Grantee xsi:type="CanonicalUser">
<ID>79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be</ID>
        <DisplayName>filesinc</DisplayName>
      </Grantee>
      <Permission>WRITE</Permission>
    </Grant>
    <Grant>
      <Grantee xsi:type="CanonicalUser">
<ID>79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be</ID>
        <DisplayName>filesinc</DisplayName>
      </Grantee>
      <Permission>READ_ACP</Permission>
    </Grant>
    <Grant>
      <Grantee xsi:type="CanonicalUser">
<ID>e019164ebb0724ff67188e243eae9ccbebdde523717cc312255d9a82498e394a</ID>
        <DisplayName>storagehelper</DisplayName>
      </Grantee>
      <Permission>WRITE</Permission>
    </Grant>
    <Grant>
      <Grantee xsi:type="CanonicalUser">
<ID>e019164ebb0724ff67188e243eae9ccbebdde523717cc312255d9a82498e394a</ID>
        <DisplayName>storagehelper</DisplayName>
      </Grantee>
      <Permission>READ_ACP</Permission>
    </Grant>
    <Grant>
      <Grantee xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/global/AllUsers<URI>
      </Grantee>
      <Permission>READ</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

Remember: When you write an ACL to S3 that contains grantees of type `AmazonCustomerByEmail`, they will be converted to the `CanonicalUser` type prior to committing the ACL.

Listing Keys

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.

See the Common List Request Parameters section for API independent information about composing a list request.

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. This common XML response document is documented in detail in the Common List Response Elements section.

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 only retrieve information about the keys 'quack' through 'quartermaster.' But a subsequent request would retrieve 'quarters' through 'quince', and so on.

Consult the Iterating Through List Results section for instructions on how to correctly handle large list result sets.

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 allows applications to organize and browse their keys hierarchically, much like how you would organize your files into directories in a filesystem. 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, like "French/logiciel". 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.

The Listing Keys Hierarchically section describes this aspect of list.

List Implementation is Efficient

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.

Common List Request Parameters

Both SOAP and REST list requests accept the following parameters:

- *Prefix*

  Restricts the response to only contain results that begin with the specified prefix. If you omit this optional argument, the value of Prefix for your query will be the empty string. In other words, the results will be not be restricted by prefix.
- *Marker*

  This optional parameter enables pagination of large result sets. Marker specifies where in the result set to resume listing. It restricts the response to only contain results that occur alphabetically after the value of marker. To retrieve the next page of results, use the last key from the current page of results as the marker in your next request. See also NextMarker, below. If Marker is omitted, the first page of results is returned.
- *Delimiter*

If this optional, Unicode string parameter is included with your request, then keys that contain the same string between the prefix and the first occurrence of the delimiter will be rolled up into a single result element in the CommonPrefixes collection. These rolled-up keys are not returned elsewhere in the response.

For example, with Prefix="USA/" and Delimiter="/", the matching keys "USA/Oregon/Salem" and "USA/Oregon/Portland" would be summarized in the response as a single "USA/Oregon" element in the CommonPrefixes collection. If an otherwise matching key does not contain the delimiter after the prefix, it appears in the Contents collection.

Each element in the CommonPrefixes collection counts as one against the MaxKeys limit. The rolled-up keys represented by each CommonPrefixes element do not.

If the Delimiter parameter is not present in your request, keys in the result set will not be rolled-up and neither the CommonPrefixes collection nor the NextMarker element will be present in the response.

- *MaxKeys*

  This optional argument limits the number of results returned in response to your query. Amazon S3 will return no more than this number of results, but possibly less. Even if MaxKeys is not specified, Amazon S3 will limit the number of results in the response. Check the IsTruncated flag to see if your results are incomplete. If so, use the Marker parameter to request the next page of results.

  For the purpose of counting MaxKeys, a 'result' is either a key in the 'Contents' collection, or a delimited prefix in the 'CommonPrefixes' collection. So for delimiter requests, MaxKeys limits the total number of list results, not just the number of keys.

While the SOAP and REST list parameters are substantially the same, the parameter names and the mechanics of submitting the request are different. A SOAP list request is an XML document, with the parameters as elements, while a REST list request is a GET on the bucket resource, with parameters in the query-string. See the API-specific sections for details:

- SOAP ListBucket
- REST Bucket GET

Access Control

The list operation requires `READ` permission on the bucket in question. Permission to list is conferred for any value of Prefix, Marker, Delimiter and MaxKeys.

Common List Response Elements

The SOAP and REST XML list response share the same structure and element names.

```
<?xml version="1.0" encoding="UTF-8"?>
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>johnsmith</Name>
  <Prefix>photos/2006/</Prefix>
  <Marker/>
  <MaxKeys>1000</MaxKeys>
  <Delimiter>/</Delimiter>
  <IsTruncated>false</IsTruncated>
  <Contents>
```

```
    <Key>photos/2006/index.html</Key>
    <LastModified>2006-06-09T17:38:37.000Z</LastModified>
    <ETag>"ce1acdafcc879d7eee54cf4e97334078"</ETag>
    <Size>1234</Size>
    <Owner>
<ID>214153b66967d86f031c7249d1d9a80249109428335cd08f1cdc487b4566cb1b</ID>
      <DisplayName>John Smith</DisplayName>
    </Owner>
    <StorageClass>STANDARD</StorageClass>
  </Contents>
  <CommonPrefixes>
    <Prefix>photos/2006/January/</Prefix>
  </CommonPrefixes>
</ListBucketResult>
```

ListBucketResult is the root element of the list response document. To make the list response self-describing, ListBucketResult echoes back the list request parameters that generated it. ListBucketResult also contains the following elements:

- *IsTruncated*

  A flag that indicates whether or not all results of your query were returned in this response. If your results were truncated, you can make a follow-up paginated request using the Marker parameter to retrieve the rest of the results.
- *NextMarker*

  A convenience element, useful when paginating with delimiters. The value of NextMarker, if present, is the largest (alphabetically) of all key names and all CommonPrefixes prefixes in the response. If the IsTruncated flag is set, request the next page of results by setting Marker to NextMarker. This element is only present in the response if the Delimiter parameter was sent with the request.

The Contents Element (of type ListEntry) contains information about each key that is part of the list results.

- *Key*

  The object's key.
- *LastModified*

  The time that the object was placed into Amazon S3.
- *ETag*

  The object's entity tag is an opaque string used to quickly check an object for changes. With high probability, the object data associated with a key is unchanged if and only if the entity tag is unchanged. Entity tags are useful in conditional gets.
- *Size*

  The number of bytes of object data stored under this key. Size does not include metadata or the size of the key.
- *Owner*

  This element represents the identity of the principal who created the object. It is only present if you have permission to view it. See the 'Access Control' discussion, below.
- *StorageClass*

Always has the value `STANDARD`

The CommonPrefixes element may be present when you make a list request with the delimiter parameter. Each element in this collection represents a group of keys that share a common prefix terminated by the specified delimiter. To expand the list of keys under this prefix, make a new list request formed by substituting the value of the CommonPrefixes/Prefix response element for the the Prefix request parameter.

- *`Prefix`*

  The shared common prefix.

Access Control

The Owner element is only present in a given ListEntry element if you have `READ_ACP` permission on the object in question, or if you own the containing bucket. Otherwise, it is omitted.

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. The following pseudo-code procedure demonstrates how to iteratively fetch an exhaustive list of results, given a prefix, marker and delimiter:

```
function exhaustiveList(bucket, prefix, marker, delimiter) :
    do {
        result = AmazonS3.list(bucket, prefix, marker, delimiter);
        // ... work with incremental list results ...

        marker = max(result.Contents.Keys, result.CommonPrefixes.Prefixes)
        // or more conveniently, when delimiter != null
        // marker = result.NextMarker;
    }
while (result.IsTruncated);
```

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 allow you to organize, and then browse, your keys hierarchically. To do this, first pick a delimiter for your bucket, say '/', 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. Since these names don't usually contain punctuation, you might select '/' as the delimiter. In that case, you would name your keys like so:

- 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. But, by using the Prefix and Delimiter parameters with the list operation, you can list using the hierarchy you've built into your data. For example, to list all the cities in California, set Delimiter='/' and Prefix='/North America/USA/California/'. 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.

Error Response

When an 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 the following common elements:

- Error Code
- Error Message
- Further details
- List of Error Codes

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.

List of Error Codes

The following table lists the Amazon S3 Error Codes.

Error Code	Description	HTTP Status Code	SOAP Fault Code Prefix
Access-Denied	Access Denied	403 Forbidden	Client
Account-Problem	There is a problem with your AWS account that prevents the operation from completing successfully. Please contact	403 Forbidden	Client

Error Code	Description	HTTP Status Code	SOAP Fault Code Prefix
	customer service at webservices@amazon.com.		
AllAccessDisabled	All access to this object has been disabled.	401 Unauthorized	Client
AmbiguousGrantByEmailAddress	The e-mail address you provided is associated with more than one account.	400 Bad Request	Client
BadDigest	The Content-MD5 you specified did not match what we received.	400 Bad Request	Client
BucketAlreadyExists	The named bucket you tried to create already exists.	409 Conflict	Client
BucketNotEmpty	The bucket you tried to delete is not empty.	409 Conflict	Client
CredentialsNotSupported	This request does not support credentials.	400 Bad Request	Client
EntityTooLarge	Your proposed upload exceeds the maximum allowed object size.	400 Bad Request	Client
MetadataTooLarge	Your metadata headers exceed the maximum allowed metadata size.	400 Bad Request	Client
IncompleteBody	You did not provide the number of bytes specified by the Content-Length HTTP header	400 Bad Request	Client
InternalError	We encountered an internal error. Please try again.	500 Internal Server Error	Server
InvalidAccessKeyId	The AWS Access Key Id you provided does not exist in our records.	403 Forbidden	Client
InvalidAddressingHeader	You must specify the Anonymous role.	N/A	Client
InvalidArgument	Invalid Argument	400 Bad Request	Client
InvalidBucketName	The specified bucket is not valid.	400 Bad Request	Client
InvalidDigest	The Content-MD5 you specified was an invalid.	400 Bad Request	Client
InvalidRange	The requested range is not satisfiable.	416 Requested Range Not Satisfiable	Client
InvalidSe-	The provided security credentials are not	403 Forbidden	Client

Error Code	Description	HTTP Status Code	SOAP Fault Code Prefix
curity	valid.		
Invalid-SOAPRe-quest	The SOAP request body is invalid.	400 Bad Request	Client
InvalidStor-ageClass	The storage class you specified is not val-id.	400 Bad Request	Client
InvalidURI	Couldn't parse the specified URI.	400 Bad Request	Client
Malforme-dACLError	The XML you provided was not well-formed or did not validate against our published schema.	400 Bad Request	Client
MethodNot-Allowed	The specified method is not allowed against this resource.	405 Method Not Allowed	Client
MissingAt-tachment	A SOAP attachment was expected, but none were found.	N/A	Client
Missing-Conten-tLength	You must provide the Content-Length HTTP header.	411 Length Required	Client
MissingSe-curityEle-ment	The SOAP 1.1 request is missing a secur-ity element.	400 Bad Request	Client
MissingSe-curityHead-er	Your request was missing a required head-er.	400 Bad Request	Client
NoSuch-Bucket	The specified bucket does not exist.	404 Not Found	Client
NoSuchKey	The specified key does not exist.	404 Not Found	Client
NotImple-mented	A header you provided implies functional-ity that is not implemented.	501 Not Implemented	Server
Not-SignedUp	Your account is not signed up for the S3 service. You must sign up before you can use S3. You can sign up at the following URL: http://aws.amazon.com/s3	403 Forbidden	Client
Precondi-tionFailed	At least one of the pre-conditions you spe-cified did not hold.	412 Precondition Failed	Client
Request-Timeout	Your socket connection to the server was not read from or written to within the timeout period.	400 Bad Request	Client
RequestTi-meToo-Skewed	The difference between the request time and the server's time is too large.	403 Forbidden	Client
RequestTor-rentOfBuck-	Requesting the torrent file of a bucket is not permitted.	400 Bad Request	Client

Error Code	Description	HTTP Status Code	SOAP Fault Code Prefix
etError			
Signature-DoesNot-Match	The request signature we calculated does not match the signature you provided. Check your AWS Secret Access Key and signing method. Consult the documentation under Authenticating REST Requests and Authenticating SOAP Requests for details.	403 Forbidden	Client
TooManyBuckets	You have attempted to create more buckets than allowed.	400 Bad Request	Client
Unexpected-Content	This request does not support content.	400 Bad Request	Client
UnresolvableGrant-ByEmailAddress	The e-mail address you provided does not match any account on record.	400 Bad Request	Client

Using the REST API

This section contains information specific to the Amazon S3 REST API. The following sections are included:

- Common REST API Elements
- The REST Error Response
- Authenticating REST Requests
- Setting Access Policy with REST
- Virtual Hosted Buckets

Operation on the Service:

- GET Service

Operations on Buckets:

- PUT Bucket
- DELETE Bucket
- GET Bucket

Operations on Objects:

- PUT Object
- GET Object
- HEAD Object
- DELETE Object

Common REST API Elements

S3 REST Operations are in fact HTTP requests, as defined by RFC 2616 (http://www.ietf.org/rfc/rfc2616.txt). This section describes how S3 uses HTTP, and the parts of HTTP requests and responses that all 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 S3, followed by waiting for the HTTP response sent from S3 back to you. Like any HTTP request, a request to S3 contains a request method, a URI, a query string, request headers, and sometimes a request body. The response contains a status code, response headers, and sometimes a response body.

Here's an example of a request to get an object named "Nelson" in the "quotes" bucket:

Sample Request

```
GET /quotes/Nelson HTTP/1.0
Authorization: AWS 15B4D3461F177624206A:Iuyz3d3P0aTou39dzbq7RrtSFmw=
Date: Thu, 17 Nov 2005 02:23:04 GMT
```

Sample Response

```
HTTP/1.1 200 OK
x-amz-id-2: qBmKRcEWBBhH6XAqsKU/eg24V3jf/kWKN9dJip1L/FpbYr9FDy7wWFurfdQOEMcY
x-amz-request-id: F2A8CCCA26B4B26D
Date: Thu, 17 Nov 2005 02:23:04 GMT
Last-Modified: Thu, 17 Nov 2005 02:23:06 GMT
ETag: "828ef3fdfa96f00ad9f27c383fc9ac7f"
Content-Type: text/plain
Content-Length: 5
Connection: close
Server: AmazonS3

ha-ha
```

REST Endpoint

All REST requests to Amazon S3 should be sent to the host s3.amazonaws.com.

Request Method

The GET operation in the preceding example indicates the operation you are trying to carry out. The same operation can have slightly different meanings depending on the URI.

URI

The URI indicates the target of the request. If it contains only one slash, as in =/quotes=, then the URI refers to a bucket. A URI that begins and ends with a slash, and has no slashes in between, like /quotes/, also refers to a bucket. If the URI contains a second slash, with characters after the second slash, as in /quotes/Nelson, then it refers to a specific object.

Request Headers

The following request header can be used with any request:

- *`Authorization`*: Provides authentication information to Amazon S3. If no authentication information is provided the request is considered anonymous. See Authenticating REST Requests for more information.
- *`Host`*: Ordinarily, the value of Host is 's3.amazonaws.com'. A Host header with a value other than 's3.amazonaws.com' selects the bucket for the request as described in the Virtual Hosted Buckets section.

  For HTTP/1.0 requests, the Host header may be omitted.

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

An example 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 S3 errors, please see the Error Response topic.

Response Headers

The following response headers are returned by all operations:

- *x-amz-request-id:* This is a unique id assigned to each request by the system. In the unlikely event that you have problems with S3, Amazon can use this to help troubleshoot the problem.

- *x-amz-id-2:* This is a special token that will help us to troubleshoot problems.

Authenticating REST Requests

Every non-anonymous request to S3 must contain authentication information to establish the identity of the principal making the request. In REST, this is done by first putting the headers in a canonical format, then signing the headers using your AWS Secret Access Key.

There are two ways to send your signature with a request. The first is to put your AWS Access Key ID and the signature you computed into the Authorization header:

Example

```
"Authorization: AWS " + AWSAccessKeyId + ":"  + base64(hmac-sha1(VERB + "\n"
                                                              + CONTENT-MD5 +
"\n"
                                                              + CONTENT-TYPE +
"\n"
                                                              + DATE + "\n"
                                                              + Canonic-
alizedAmzHeaders + "\n"
                                                              + Canonic-
alizedResource))
```

You can also send a signature as a URL-encoded query-string parameter in the URL for the request. This is useful if you want to enable a third party to access S3 on your behalf without your having to proxy the data transfer. For example, if you want to enable a user to download your private data directly from S3, you can insert a pre-signed URL into a web page before giving it to your user. The canonicalized string that you sign is the same, except that you replace the DATE field in the string with an Expires field that indicates when you want the signature to expire. The Expires field is given as the number of seconds since epoch time, and is also included as a query string parameter along with your AWS Access Key ID:

Example

```
GET /my-bucket/foo
                            ?Signature=<urlencode(base64(hmac-sha1(VERB +
"\n" + CONTENT-MD5 + "\n" + CONTENT-TYPE + "\n" + Expires + "\n" + Canonic-
alizedAmzHeaders + "\n" + CanonicalizedResource)))>
                            &Expires=<seconds since epoch>
                            &AWSAccessKeyId=<aws-id>
```

Canonicalization for Authorization Header Authentication

When authenticating through the Authorization header, you create the string to be signed by concatenating the request verb with canonicalized headers and the resource that the request is targeting.

The headers used for request signing are: content-md5, content-type, date, and anything that starts with x-amz-. The string to be signed is formed by appending the REST verb, content-md5 value, content-type value, date value, canonicalized x-amz headers (see recipe below), and the resource; all separated by newlines. (If you cannot set the Date header, use the x-amz-date header as described below.)

The resource is the bucket and key (if applicable), separated by a '/'. If the request you are signing is for an ACL or a torrent file, you should include ?acl or ?torrent in the resource part of the canonical string. No other query string parameters should be included, however.

Canonicalization for Query String Authentication

When authenticating via query string parameters, you create the string to be signed by concatenating the request verb with canonicalized headers and the resource that the request is targeting.

The headers used for request signing are the same as those for authorization header authentication, except that the Date field is replaced by the Expires parameter. The Expires parameter is the time when you want the signature to expire, specified as the number of seconds since the epoch time.

Thus, the string to be signed is formed by appending the REST verb, content-md5 value, content-type value, expires parameter value, canonicalized x-amz headers (see recipe below), and the resource; all separated by newlines.

The resource is the same as that for authorization header authentication: the bucket and key (if applicable), separated by a '/'. If the request you are signing is for an ACL or a torrent file, you should include ?acl or ?torrent in the resource part of the canonical string. No other query string parameters should be included, however.

x-amz headers are canonicalized by:

- Lower-case header name
- Headers sorted by header name
- The values of headers whose names occur more than once should be white space-trimmed and concatenated with comma separators to be compliant with section 4.2 of RFC 2616.
- remove any whitespace around the colon in the header
- remove any newlines ('\n') in continuation lines
- separate headers by newlines ('\n')

Some important points:

- The string to sign (verb, headers, resource) must be UTF-8 encoded.
- The content-type and content-md5 values are optional, but if you do not include them you must still insert a newline at the point where these values would normally be inserted.
- Some toolkits may 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.
- Some toolkits make it difficult to manually set the date. If you have trouble including the value of the 'Date' header in the canonicalized headers, you can include an 'x-amz-date' header prior to canonicalization. The value of the x-amz-date header must be in one of the RFC 2616 formats (http://www.ietf.org/rfc/rfc2616.txt). If S3 sees an x-amz-date header in the request, it will ignore the Date header when validating the request signature. If you include the x-amz-date header, you must still include a newline character in the canonicalized string at the point where the Date value would normally be inserted.
- The value of the Date or, if applicable, x-amz-date header must specify a time no more than 15 minutes away from the S3 webserver's clock.
- The hash function to compute the signature is HMAC-SHA1 defined in RFC 2104 (http://www.ietf.org/rfc/rfc2104.txt), using your Secret Access Key as the key.

REST Authentication: Example 1

Example

For example, imagine that you want to sign the following request:

```
PUT /quotes/nelson HTTP/1.0
Content-Md5: c8fdb181845a4ca6b8fec737b3581d76
Content-Type: text/html
Date: Thu, 17 Nov 2005 18:49:58 GMT
X-Amz-Meta-Author: foo@bar.com
X-Amz-Magic: abracadabra
```

The canonical string to be signed is:

```
PUT\nc8fdb181845a4ca6b8fec737b3581d76\ntext/html\nThu, 17 Nov 2005 18:49:58
GMT\nx-amz-magic:abracadabra\nx-amz-meta-author:foo@bar.com\n/quotes/nelson
```

Suppose your AWS Access Key ID is "44CF9590006BF252F707" and your AWS Secret Access Key is "OtxrzxIsfpFjA7SwPzILwy8Bw21TLhquhboDYROV". Then you could compute the signature as follows:

```
import base64
import hmac
import sha
h = hmac.new("OtxrzxIsfpFjA7SwPzILwy8Bw21TLhquhboDYROV",
             "PUT\nc8fdb181845a4ca6b8fec737b3581d76\ntext/html\nThu, 17 Nov
2005 18:49:58
GMT\nx-amz-magic:abracadabra\nx-amz-meta-author:foo@bar.com\n/quotes/nelson",
             sha)
base64.encodestring(h.digest()).strip()
```

The resulting signature would be "jZNOcbfWmD/A/f3hSvVzXZjM2HU=", and, you would add the Authorization header to your request to come up with the following result:

```
PUT /quotes/nelson HTTP/1.0
Authorization: AWS 44CF9590006BF252F707:jZNOcbfWmD/A/f3hSvVzXZjM2HU=
Content-Md5: c8fdb181845a4ca6b8fec737b3581d76
Content-Type: text/html
Date: Thu, 17 Nov 2005 18:49:58 GMT
X-Amz-Meta-Author: foo@bar.com
X-Amz-Magic: abracadabra
```

REST Authentication: Example 2

Example

Imagine that you can't set the Date header, and don't know what value your toolkit will assign that header. Then you need to include the x-amz-date header in your request:

```
GET /quotes/nelson HTTP/1.0
Date: XXXXXXXXX
X-Amz-Magic: abracadabra
X-Amz-Date: Thu, 17 Nov 2005 18:49:58 GMT
```

The canonical string to be signed is (note the included newlines even though there is no Content-Md5 or Content-Type header in the request):

```
GET\n\n\n\nx-amz-date:Thu, 17 Nov 2005 18:49:58
GMT\nx-amz-magic:abracadabra\n/quotes/nelson
```

Suppose your AWS Access Key ID is "44CF9590006BF252F707" and your AWS Secret Access Key is "OtxrzxIsfpFjA7SwPzILwy8Bw21TLhquhboDYROV". Then you could compute the signature as follows:

```
import base64
import hmac
import sha
h = hmac.new("OtxrzxIsfpFjA7SwPzILwy8Bw21TLhquhboDYROV",
             "GET\n\n\n\nx-amz-date:Thu, 17 Nov 2005 18:49:58
GMT\nx-amz-magic:abracadabra\n/quotes/nelson",
             sha)
base64.encodestring(h.digest()).strip()
```

The resulting signature would be "5m+HAmc5JsrgyDelh9+a2dNrzN8=", and, you would add the Authorization header to your request to come up with the following result:

```
GET /quotes/nelson HTTP/1.0
Authorization: AWS 44CF9590006BF252F707:5m+HAmc5JsrgyDelh9+a2dNrzN8=
Date: XXXXXXXXX
X-Amz-Magic: abracadabra
X-Amz-Date: Thu, 17 Nov 2005 18:49:58 GMT
```

REST Authentication Example 3: Query String Authentication Example

Example

Let's say you want to let someone else access http://s3.amazonaws.com/quotes/nelson via a web browser. Query String Auth URLs all have an Expires parameter which specifies until when the URL is still valid. The value of this parameter is expressed in seconds since epoch. On unix, you can run the command "date +%s", or in java, you can divide the result of System.currentTimeMillis() by 1000. So, if it's 1141889060 right now, and you want the url to be valid for 60 seconds, your Expires parameter would be 1141889120. This value will be used in place of the Date header when computing the canonical string.

The canonical string to be signed is:

```
GET\n\n\n1141889120\n/quotes/nelson
```

You know that when the browser makes the GET request, it won't provide a Content-Md5 or a Content-Type hader, nor will it set any x-amz- headers, so those parts are all kept empty.

Suppose your AWS Access Key ID is "44CF9590006BF252F707" and your AWS Secret Access Key is "OtxrzxIsfpFjA7SwPzILwy8Bw21TLhquhboDYROV". Then you could compute the signature as follows:

```
import base64
import hmac
import sha
import urllib
h = hmac.new("OtxrzxIsfpFjA7SwPzILwy8Bw21TLhquhboDYROV",
             "GET\n\n\n1141889120\n/quotes/nelson",
```

```
            sha)
urllib.quote_plus(base64.encodestring(h.digest()).strip())
```

Note that we also url-encoded the result this time. This is because the output from the base64 algorithm is not suitable for use as a query string parameter, so we add an additional layer of armor to make it acceptable.

The resulting signature would be "vjbyPxybdZaNmGa%2ByT272YEAiv4%3D", so the parameters we will use are:

Key	Value
AWSAccessKeyId	44CF9590006BF252F707
Expires	1141889120
Signature	vjbyPxybdZaNmGa%2ByT272YEAiv4%3D

And the resulting URL would be:

```
ht-
tp://s3.amazonaws.com/quotes/nelson?AWSAccessKeyId=44CF9590006BF252F707&Expir
es=1141889120&Signature=vjbyPxybdZaNmGa%2ByT272YEAiv4%3D
```

Setting Access Policy with REST

There are two ways to set the access control policy with REST. You can set the access control list (ACL) 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 ACL with the written resource.

Setting the ACL on an Existing Bucket or Object

You can set the ACL on an existing bucket or object by doing 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`.

The following example demonstrates how to set an existing object ACL so that only the owner has full access to the object. (The owner's canonical user grant information is first found by executing a `GET` on `/quotes/Neo?acl`.):

```
PUT /quotes/Neo?acl HTTP/1.0
Content-Length: 214
Host: s3.amazonaws.com
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIc8F2Cy8=
Date: Thu, 17 Nov 2005 07:48:33 GMT
Content-Type: text/plain

<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).

The following canned ACLs are supported for REST:

- *`private:`* Owner gets `FULL_CONTROL`. No one else has any access rights. This is the 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 is a useful policy to apply to a bucket, if you intend for any anonymous user to PUT objects into the bucket.

- *authenticated-read:* Owner gets FULL_CONTROL, and any principal authenticated as a registered Amazon S3 user is granted READ access.

The following example demonstrates how to write data to an object and makes the object readable by anonymous principals:

Sample Request

```
PUT /quotes/Neo HTTP/1.0
x-amz-acl: public-read
Content-Length: 4
Host: s3.amazonaws.com
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIc8F2Cy8=
Date: Thu, 17 Nov 2005 07:48:33 GMT
Content-Type: text/plain

woah
```

Sample Response

```
HTTP/1.1 200 OK
x-amz-id-2: LriYPLdmOdAiIfgSm/F1YsViT1LW94/xUQxMsF7xiEb1a0wiIOIxl+zbwZ163pt7
x-amz-request-id: 0A49CE4060975EAC
Date: Thu, 17 Nov 2005 07:48:32 GMT
ETag: "aba878a8"
Content-Length: 0
Connection: close
Server: AmazonS3
```

Virtual Hosting of Buckets

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's interpretation of `Host` means 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.

Using the HTTP Host Header to Specify the Bucket

If a GET request is anonymous (not signed, no `Authorization` header) and does not use the SSL endpoint, you may specify the bucket for the request using the HTTP `Host` header. The `Host` header in a REST request is interpreted as follows:

- If the 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 table below. 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 table below.
- 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 domain name as your bucket name, and have installed a CNAME record that makes your domain an alias for s3.amazonaws.com. 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 table below.

Examples

The following example illustrates these cases:

Example URL	Example Request	Bucket Name for Request	Key for Request	Notes
http://s3.amazonaws.com/johnsmith/homepage.html	`GET /johnsmith/homepage.html HTTP/1.1` `Host: s3.amazonaws.com`	johnsmith	homepage.html	The ordinary method
	`GET /john-`	johnsmith	homepage.html	HTTP 1.0 may omit the Host

Example URL	Example Request	Bucket Name for Request	Key for Request	Notes
	`smith/homepage.html HTTP/1.0`			header
http://johnsmith.s3.amazonaws.com/homepage.html	`GET /homepage.html HTTP/1.1 Host: johnsmith.s3.amazonaws.com`	johnsmith	homepage.html	All lower-case Amazon S3 buckets are automatically addressable by the sub-domain method.
	`GET /homepage.html HTTP/1.1 Host: JohnSmith.s3.amazonaws.com`	johnsmith	homepage.html	Note that the case of the Host header is insignificant. Upper-case bucket names are not accessible using this method.
http://www.johnsmith.net/homepage.html	`GET /homepage.html HTTP/1.1 Host: www.johnsmith.net`	www.johnsmith.net	homepage.html	To host a website in Amazon S3 using this method, you must setup a your domain name as a CNAME alias for s3.amazonaws.com.

Limitations

Because DNS names are case insensitive, only lower-case buckets are addressable using the virtual hosting method.

Specifying the bucket for the request using the HTTP `Host` header is supported:

- For non-SSL requests.
- For non-authenticated or anonymous requests. A `NotImplemented` error result code will be returned if an authenticated (signed) request specifies a Host: header other than 's3.amazonaws.com'
- Using the REST API. You cannot specify the bucket in SOAP by using a different endpoint.

Backwards compatibility note

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.

Operations on the Service

GET Service

GET Operation

The `GET` operation on the Service endpoint (s3.amazonaws.com) returns a list of all of the buckets owned by the authenticated sender of the request.

Sample Request

```
GET / HTTP/1.1
Host: s3.amazonaws.com
Date: Fri, 24 Feb 2006 23:21:59 +0000
Authorization: AWS 87345958AFJKLG383383:dHPvhW0577GbydEoDSVOzYOgscc=
```

Sample Response

```
<?xml version="1.0" encoding="UTF-8"?>
<ListAllMyBucketsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>bcaf1ffd86f41caff1a493dc2ad8c2c281e37522a640e161ca5fb16fd081034f</ID>
    <DisplayName>webfile</DisplayName>
  </Owner>
  <Buckets>
    <Bucket>
      <Name>quotes;/Name>
      <CreationDate>2006-02-03T16:45:09.000Z</CreationDate>
    </Bucket>
    <Bucket>
      <Name>samples</Name>
      <CreationDate>2006-02-03T16:41:58.000Z</CreationDate>
    </Bucket>
  </Buckets>
</ListAllMyBucketsResult>
```

Response Body

- *`Owner:`* This provides information that S3 uses to represent your identity for purposes of authentication and access control. ID is a unique and permanent identifier for the developer who made the request. DisplayName is a human-readable name representing the developer who made the request. It is not unique, and may change over time.
- *`Name:`* The name of a bucket.
- *`CreationDate:`* The time that the bucket was created.

Access Control

You must authenticate with a valid AWS Access Key ID that has been registered as a user of S3. Anonymous requests are never allowed to list buckets, and you cannot list buckets that you did not create.

Operations on Buckets

PUT Bucket

The `PUT` request operation with a bucket URI creates a new bucket. The length of the bucket name must be between 3 and 255 bytes. It can contain letters, numbers, dashes, and underscores.

Creates a bucket named "quotes"

Sample Request

```
PUT /quotes HTTP/1.0
Content-Length: 0
Authorization: AWS 15B4D3461F177624206A:YFhSWKDg3qDnGbV7JCnkfdz/IHY=
Date: Thu, 17 Nov 2005 02:40:52 GMT
```

Sample Response

```
HTTP/1.1 200 OK
x-amz-id-2: YgIPIfBiKa2bj0KMg95r/0zo3emzU4dzsD4rcKCHQUAdQkf3ShJTOOpXUueF6QKo
x-amz-request-id: 236A8905248E5A01
Date: Thu, 17 Nov 2005 02:40:54 GMT
Location: /quotes
Content-Length: 0
Connection: close
Server: AmazonS3
```

Access Control

You must authenticate with a valid AWS Access Key ID that has been registered as a user of S3. Anonymous requests are never allowed to create buckets.

GET Bucket

A `GET` request operation using a bucket URI lists information about the objects in the bucket.

For a general introduction to the list operation, see the Listing Keys section.

List up to 40 keys in the "quotes" bucket that have the prefix "N" and occur lexicographically after "Ned":

Sample Request

```
GET /quotes?prefix=N&marker=Ned&max-keys=40 HTTP/1.0
Authorization: AWS 15B4D3461F177624206A:J3dZMRrvG+00yAeI3zNev33KH8g=
Date: Thu, 17 Nov 2005 07:13:50 GMT
```

Sample Response

```
HTTP/1.1 200 OK
x-amz-id-2: gyB+3jRPnrkN98ZajxHXr3u7EFM67bNgSAxexeEHndCX/7GRnfTXxReKUQF28IfP
x-amz-request-id: 3B3C7C725673C630
Date: Thu, 17 Nov 2005 07:13:49 GMT
Content-Type: application/xml
Content-Length: 302
Connection: close
Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>quotes</Name>
  <Prefix>N</Prefix>
  <Marker>Ned</Marker>
  <MaxKeys>40</MaxKeys>
  <IsTruncated>false;</IsTruncated>
  <Contents>
    <Key>Nelson</Key>
    <LastModified>2005-11-17T07:13:48Z</LastModified>
    <ETag>&quot;828ef3fdfa96f00ad9f27c383fc9ac7f&quot;</ETag>
    <Size>5</Size>
    <StorageClass>STANDARD</StorageClass>
    <Owner>
<ID>bcaf1ffd86f41caff1a493dc2ad8c2c281e37522a640e161ca5fb16fd081034f</ID>
      <DisplayName>webfile</DisplayName>
     </Owner>
  </Contents>
  <Contents>
    <Key>Neo</Key>
    <LastModified>2005-11-17T07:13:48Z</LastModified>
    <ETag>&quot;828ef3fdfa96f00ad9f27c383fc9ac7f&quot;</ETag>
    <Size>4</Size>
    <StorageClass>STANDARD</StorageClass>
     <Owner>
<ID>bcaf1ffd86f41caff1a493dc2ad8c2c281e37522a640e161ca5fb16fd081034f</ID>
      <DisplayName>webfile</DisplayName>
    </Owner>
 </Contents>
</ListBucketResult>
```

Request Parameters

See the Common List Request Parameters section for comprehensive information about the list request parameters.

- *`prefix:`* Limits the response to keys which begin with the indicated prefix. You can use prefixes to separate a bucket into different sets of keys in a way similar to how a file system uses folders.

- *`marker:`* Indicates where in the bucket to begin listing. The list will only include keys that occur lexicographically after marker. This is convenient for pagination: To get the next page of results use the last key of the current page as the marker.

- *`max-keys:`* The maximum number of keys you'd like to see in the response body. The server may return fewer than this many keys, but will not return more.

- *`delimiter:`* Causes keys that contain the same string between the prefix and the first occurrence of

the delimiter to be rolled up into a single result element in the CommonPrefixes collection. These rolled-up keys are not returned elsewhere in the response.

Response Body

See the Common List Response Elements section for information about the list response.

Access Control

To list the keys of a bucket you need to have `READ` access to the bucket.

DELETE Bucket

The `DELETE` request operation deletes the bucket named in the URI. All objects in the bucket must be deleted before the bucket itself can be deleted.

Delete the bucket named "quotes"

Sample Request

```
DELETE /quotes HTTP/1.0
Authorization: AWS 15B4D3461F177624206A:1/7p5NQEjCNCv4WMkyElX0c/8cs=
Date: Thu, 17 Nov 2005 06:19:54 GMT
```

Sample Response

```
HTTP/1.1 204 No Content
x-amz-id-2: JuKZqmXuiwFeDQxhD7M8KtsKobSzWA1QEjLbTMTagkKdBX2z7Il/jGhDeJ3j6s80
x-amz-request-id: 32FE2CEB32F5EE25
Date: Thu, 17 Nov 2005 06:19:53 GMT
Connection: close
Server: AmazonS3
```

Access Control

Only the owner of a bucket is allowed to delete it, regardless of the bucket's access control policy.

Operations on Objects

PUT Object

The PUT request operation is used to add an object to a bucket.

The response indicates that the object has been successfully stored. S3 never stores partial objects: if you receive a successful response, then you can be confident that the entire object was stored.

If the object already exists in the bucket, the new object overwrites the existing object. S3 orders all of the requests that it receives. It is possible that if you send two requests nearly simultaneously, we will receive them in a different order than they were sent. The last request received is the one which is stored in S3. Note that this means if multiple parties are simultaneously writing to the same object, they may all get a successful response even though only one of them wins in the end. This is because S3 is a distributed system and it may take a few seconds for one part of the system to realize that another part has received an object update. In this release of Amazon S3, there is no ability to lock an object for writing -- such functionality, if required, should be provided at the application layer.

Write some text and metadata into the "Neo" object in the "quotes" bucket:

Sample Request

```
PUT /quotes/Neo HTTP/1.0
x-amz-meta-family: Anderson
Content-Length: 4
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIc8F2Cy8=
Date: Thu, 17 Nov 2005 07:48:33 GMT
Content-Type: text/plain

woah
```

Sample Response

```
HTTP/1.1 200 OK
x-amz-id-2: LriYPLdmOdAiIfgSm/F1YsViT1LW94/xUQxMsF7xiEb1a0wiIOIxl+zbwZ163pt7
x-amz-meta-family: Anderson
x-amz-request-id: 0A49CE4060975EAC
Date: Thu, 17 Nov 2005 07:48:32 GMT
ETag: "828ef3fdfa96f00ad9f27c383fc9ac7f"
Content-Length: 0
Connection: close
Server: AmazonS3
```

Request Headers

- *Cache-Control:* Can be used to specify caching behavior along the request/reply chain. See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9.
- *Content-Type:* A standard MIME type describing the format of the contents. If none is provided, the default is binary/octet-stream. See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.17.
- *Content-Length:* The size of the object, in bytes. This is required. See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.13.
- *Content-MD5:* An MD-5 hash of the object's value. This is used to verify integrity of the object in transport. See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.15.
- *Content-Disposition:* Specifies presentational information for the object. See http://www.w3.org/Protocols/rfc2616/rfc2616-sec19.html#sec19.5.1.
- *Content-Encoding:* Specifes what content codings have been applied to the object and thus what decoding mechanisms must be applied in order to obtain the media-type referenced by the Content-Type header field. See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.11.
- *Expires:* Gives the date/time after which the response is considered stale. See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.21.
- *x-amz-meta-:* x-amz-meta-: Any header starting with this prefix is considered user metadata. It will be stored with the object and returned when you retrieve the object. The total size of the HTTP request, not including the body, must be less than 4 KB.

Request Body

The data to be stored in the object is sent in the request body.

Response Headers

- *ETag:* The entity tag is an MD5 hash of the object that you can use to do conditional `GET` operatons using the If-Modified request tag with the `GET` request operation. You can also use this value to compare to your own calculated MD5 hash to ensure the object wasn't corrupted over the wire.

Access Control

You must have `WRITE` access to the bucket to add an object.

GET Object

You fetch objects from S3 using the `GET` operation. This operation returns the object directly from S3 using a client/server delivery mechanism. If you want to distribute big files to a large number of people, you may find BitTorrent delivery to be preferable since it uses less bandwidth. Please see the section on Using BitTorrent with S3 for details.

Retrieve the "Nelson" object and its metadata from the "quotes" bucket:

Sample Request

```
GET /quotes/Nelson HTTP/1.0
Authorization: AWS 15B4D3461F177624206A:nV/j6LGf9AcWQ2jId1qOGgekHPQ=
Date: Thu, 17 Nov 2005 08:22:38 GMT
```

Sample Response

```
HTTP/1.1 200 OK
x-amz-id-2: j5ULAWpFbJQJpukUsZ4tfXVOjVZExLtEyNTvY5feC+hHIegsN5p578JLTVpkFrpL
x-amz-request-id: BE39A20848A0D52B
Date: Thu, 17 Nov 2005 08:22:38 GMT
x-amz-meta-family: Muntz
Last-Modified: Thu, 17 Nov 2005 08:22:38 GMT
ETag: "828ef3fdfa96f00ad9f27c383fc9ac7f"
Content-Type: text/plain
Content-Length: 5
Connection: close
Server: AmazonS3

HA-HA
```

Request Headers

The GET request method on objects supports several standard HTTP request headers. These are briefly described below, for details see the HTTP spec, RFC 2616 (http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html).

- *If-Modified-Since:* Return the object only if it has been modified since the specified time, otherwise return a 304 (not modified).
- *If-Unmodified-Since:* Return the object only if it has not been modified since the specified time, otherwise return a 412 (precondition failed).
- *If-Match:* Return the object only if its entity tag (ETag) is the same as the one specified, otherwise return a 412 (precondition failed).
- *If-None-Match:* Return the object only if its entity tag (ETag) is different from the one specified, otherwise return a 304 (not modified).
- *Range:* Return only the bytes of the object in the specified range.

Response Headers

- *x-amz-meta-:* If you supplied user metadata when you PUT the object, that metadata is returned in one or more response headers prefixed with x-amz-meta- and with the suffix name that you provided on storage. This metadata is simply returned verbatim; it is not interpreted by S3.
- *Content-Type:* This is set to the same value you specified in the corresponding header when the data was PUT. The default content type is binary/octet-stream.
- *Content-Disposition:* This is set to the same value you specified in the corresponding header when the data was PUT. Except in the case of a request for a BitTorrent torrent file (see section on using BitTorrent with S3), if no Content-Disposition was specified at the time of PUT then this

header is not returned.

- *Content-Range:* This indicates the range of bytes returned in the event that you requested a subset of the object by setting the Range request header.

- *x-amz-missing-meta:* This is set to the number of metadata entries not returned in x-amz-meta headers. This can happen if you create metadata using an API like SOAP that supports more flexible metadata than the REST API. For example, using SOAP, you can create metadata whose values are not legal HTTP headers.

Access Control

The GET will succeed if you have been granted READ access to the object. If you make a request without an authorization header, then you can read the object if READ access has been granted to the anonymous user.

HEAD Object

The HEAD operation is used to retrieve information about a specific object, without actually fetching the object itself. This is useful if you're only interested in the object metadata, and don't want to waste bandwidth on the object data. A HEAD request has the same options as a GET operation on an object. The response is identical to the GET response, except that there is no response body. For details, please see GET Object.

Retrieve only the metadata for the "Nelson" object in the "quotes" bucket:

Sample Request

```
HEAD /quotes/Nelson HTTP/1.0
Host: s3.amazonaws.com
Authorization: AWS 15B4D3461F177624206A:EZBZUmGd1qqMRk7ffYkmCbrWExM=
Date: Thu, 17 Nov 2005 08:22:39 GMT
```

Sample Response

```
HTTP/1.1 200 OK
x-amz-id-2: KZ7XUBI18rqFH91yZmYpWSRPg0/aeqwJXVzNgnk9Pa9GcHUuN2cxfsKk7V3NSUKg
x-amz-request-id: F7B5DF3AB381F03F
Date: Thu, 17 Nov 2005 08:22:38 GMT
x-amz-meta-family: Muntz
Last-Modified: Thu, 17 Nov 2005 08:22:38 GMT
ETag: "828ef3fdfa96f00ad9f27c383fc9ac7f"
Content-Type: text/plain
Content-Length: 5
Connection: close
Server: AmazonS3
```

DELETE Object

The DELETE request operation removes the specified object from Amazon S3. Once you've deleted it, there is no going back: Amazon S3 does not support any form of undelete.

Delete the "Nelson" object from the "quotes" bucket:

Sample Request

```
DELETE /quotes/Nelson HTTP/1.0
Host: s3.amazonaws.com
Authorization: AWS 15B4D3461F177624206A:ddmbvSlc742HK94p46a5r1dvdLk=
Date: Thu, 17 Nov 2005 08:22:40 GMT
```

Sample Response

```
HTTP/1.1 204 No Content
x-amz-id-2: 4NJT5+xl9kKL1w8YnhfDTbMPvBEIbl8Ek/kuX55i+4FTSINVbcRDVnhi4TZGcj1y
x-amz-request-id: 7FA15BA5170D44B0
Date: Thu, 17 Nov 2005 08:22:38 GMT
Connection: close
Server: AmazonS3
```

Access Control

You can delete an object if, and only if, you have `WRITE` access to the bucket. It does not matter who owns the object, or what rights have been granted to the object.

Using the SOAP API

This section contains information specific to the Amazon S3 SOAP API. The following sections are included:

- Common SOAP API Elements
- The SOAP Error Response
- Authenticating SOAP Requests
- Setting Access Policy with SOAP

Operation on the Service

- ListAllMyBuckets

Operation on Buckets

- CreateBucket
- DeleteBucket
- ListBucket
- GetBucketAccessControlPolicy
- SetBucketAccessControlPolicy

SOAP Operation on Objects

- PutObjectInline
- PutObject
- GetObject
- GetObjectExtended
- DeleteObject
- GetObjectAccessControlPolicy
- SetObjectAccessControlPolicy

Common SOAP API Elements

You can interact with S3 using SOAP 1.1 over HTTP. The S3 WSDL, which describes the S3 API in a machine-readable way, is available at: http://s3.amazonaws.com/doc/2006-03-01/AmazonS3.wsdl. The S3 schema is available at http://s3.amazonaws.com/doc/2006-03-01/AmazonS3.xsd.

Most users will interact with S3 using a SOAP toolkit tailored for their language and devlopment environment. Different toolkits will expose the S3 API in different ways. Please refer to your specific toolkit documentation to understand how to use it. This section illustrates the S3 SOAP operations in a toolkit-independent way by exhibiting the XML requests and responses as they appear "on the wire."

SOAP Endpoint

You can send S3 SOAP messages to either a SSL secured or un-secured endpoint. Note that authenticated SOAP requests are only accepted over SSL. The available S3 SOAP endpoints are `http://s3.amazonaws.com/soap` and `https://s3.amazonaws.com/soap` (SSL).

Common Elements

You can include the following authorization-related elements with any SOAP request:

- *`AWSAccessKeyId:`* The AWS Access Key ID of the requestor.
- *`Timestamp:`* The current time on your system.
- *`Signature:`* The signature for the request.

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 S3 SOAP fault code is comprised of a standard SOAP 1.1 fault code (either "Server" or "Client") concatenated with the 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, which looks like:

```
<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, please see the Error Response.

Authenticating SOAP Requests

Every non-anonymous request must contain authentication information to establish the identify of the principal making the request. In SOAP, the authentication information is put into the following elements of the SOAP request:

- *`AWSAccessKeyId:`* Your AWS Access Key ID
- *`Timestamp:`* This must be a dateTime (http://www.w3.org/TR/xmlschema-2/#dateTime) in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as `2005-01-31T23:59:59.183Z`. 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 (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 "AmazonS3CreateBucket2005-01-31T23:59:59.183Z":

For example, in the following CreateBucket sample request, the signature element would contain the HMAC-SHA1 digest of the value "AmazonS3CreateBucket2005-01-31T23:59:59.183Z":

```
<CreateBucket xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Bucket>quotes</Bucket>
  <Acl>private</Acl>
  <AWSAccessKeyId>1D9FVRAYCP1VJS767E02</AWSAccessKeyId>
  <Timestamp>2005-01-31T23:59:59.183Z</Timestamp>
  <Signature>SZf1CHmQ/nrZbsrC13hCZS061yws</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 S3 overly specific time stamps. This can be accomplished by manually constructing `DateTime` objects with only millisecond precision.

For more information, please 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 the Using Access Control Lists topic. If no access control list is specified with these operations, the resource is created with a default access policy that gives the requestor FULL_CONTROL access (this is the case even if the request is a PutObjectInline or PutObject request for an object that already exists).

The following sample request 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

The following sample request writes data to an object and makes the object readable by anonymous principals:

Sample Request

```
<PutObjectInline xmlns="http://s3.amazonaws.com/doc/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>1D9FVRAYCP1VJS767E02</AWSAccessKeyId>
  <Timestamp>2005-01-31T23:59:59.183Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbq7RrtSFmw=</Signature>
</PutObjectInline>
```

Sample Response

```
<PutObjectInlineResponse xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <PutObjectInlineResponse>
    <ETag>&quot828ef3fdfa96f00ad9f27c383fc9ac7f&quot</ETag>
    <LastModified>2005-11-18T00:27:11.991Z</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. See the detailed explanation of these methods for more information.

Operations on the Service

- ListAllMyBuckets

ListAllMyBuckets

The `ListAllMyBuckets` operation returns a list of all buckets owned by the sender of the request.

Sample Request

```
<ListAllMyBuckets xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <AWSAccessKeyId>1D9FVRAYCP1VJS767E02</AWSAccessKeyId>
  <Timestamp>2005-01-31T23:59:59.999Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbq7RrtSFmw=</Signature>
</ListAllMyBuckets>
```

Sample Response

```
<ListAllMyBucketsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>bcaf1ffd86f41caff1a493dc2ad8c2c281e37522a640e161ca5fb16fd081034f</ID>
    <DisplayName>webfile</DisplayName>
  </Owner>
  <Buckets>
    <Bucket>
      <Name>quotes;/Name>
      <CreationDate>2006-02-03T16:45:09.000Z</CreationDate>
    </Bucket>
    <Bucket>
      <Name>samples</Name>
      <CreationDate>2006-02-03T16:41:58.000Z</CreationDate>
    </Bucket>
  </Buckets>
</ListAllMyBucketsResult>
```

Response Body

- *`Owner:`* This provides information that S3 uses to represent your identity for purposes of authentication and access control. ID is a unique and permanent identifier for the developer who made the request. DisplayName is a human-readable name representing the developer who made the request. It is not unique, and may change over time.

- *`Name:`* The name of a bucket.

- *`CreationDate:`* The time that the bucket was created.

Access Control

You must authenticate with a valid AWS Access Key ID. Anonymous requests are never allowed to list buckets, and you can only list buckets for which you are the owner.

Operations on Buckets

The following operations are can be performed on Buckets:

- CreateBucket
- DeleteBucket
- ListBucket
- GetBucketAccessControlPolicy
- SetBucketAccessControlPolicy

CreateBucket

The `CreateBucket` operation creates a bucket.

Create a bucket named "quotes":

Sample Request

```
<CreateBucket xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Bucket>quotes</Bucket>
  <AWSAccessKeyId>1D9FVRAYCP1VJS767E02</AWSAccessKeyId>
  <Timestamp>2005-01-31T23:59:59.183Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbq7RrtSFmw=</Signature>
</CreateBucket>
```

Sample Response

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

Elements

- *`Bucket:`* The name of the bucket you are trying to create. The bucket's name must be more than 3, but less than 255 bytes in length. It can contain letters, numbers, dashes, and underscores. The namespace for bucket is global. If one user successfully creates a bucket called "abc" other users will not be able to create a bucket with same name. This is somewhat like how domain names are global through the Internet.
- *`AccessControlList:`* The access control list for the new bucket. This element is optional. If not provided, the bucket is created with an access policy that give the requestor FULL_CONTROL access.

Access Control

You must authenticate with a valid AWS Access Key ID. Anonymous requests are never allowed to create buckets.

DeleteBucket

The `DeleteBucket` operation deletes a bucket. All objects in the bucket must be deleted before the bucket itself can be deleted.

Delete the "quotes" bucket:

Sample Request

```
<DeleteBucket xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Bucket>quotes</Bucket>
  <AWSAccessKeyId> 1D9FVRAYCP1VJS767E02</AWSAccessKeyId>
  <Timestamp>2005-01-31T23:59:59.183Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbq7RrtSFmw=</Signature>
</DeleteBucket>
```

Sample Response

```
<DeleteBucketResponse xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <DeleteBucketResponse>
    <Code>204</Code>
    <Description>No Content</Description>
  </DeleteBucketResponse>
</DeleteBucketResponse>
```

Elements

- *`Bucket:`* The name of the bucket you want to delete.

Access Control

Only the owner of a bucket is allowed to delete it, regardless the access control policy on the bucket.

ListBucket

The `ListBucket` operation returns information about some of the items in the bucket.

For a general introduction to the list operation, see the Listing Keys section.

List up to 40 keys in the "quotes" bucket that have the prefix "N" and occur lexographically after "Ned":

Sample Request

```
<ListBucket xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Bucket>quotes</Bucket>
  <Prefix>N</Prefix>
```

```
  <Marker>Ned</Marker>
  <MaxKeys>40</MaxKeys>
  <AWSAccessKeyId>1D9FVRAYCP1VJS767E02</AWSAccessKeyId>
  <Timestamp>2005-01-31T23:59:59.183Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbq7RrtSFmw=</Signature>
</ListBucket>
```

Sample Response

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>quotes</Name>
  <Prefix>N</Prefix>
  <Marker>Ned</Marker>
  <MaxKeys>40</MaxKeys>
  <IsTruncated>false;</IsTruncated>
  <Contents>
    <Key>Nelson</Key>
    <LastModified>2005-11-17T07:13:48Z</LastModified>
    <ETag>&amp;quot;828ef3fdfa96f00ad9f27c383fc9ac7f&amp;quot;</ETag>
    <Size>5</Size>
    <StorageClass>STANDARD</StorageClass>
    <Owner>
<ID>bcaf1ffd86f41caff1a493dc2ad8c2c281e37522a640e161ca5fb16fd081034f</ID>
      <DisplayName>webfile</DisplayName>
     </Owner>
  </Contents>
  <Contents>
    <Key>Neo</Key>
    <LastModified>2005-11-17T07:13:48Z</LastModified>
    <ETag>&amp;quot;828ef3fdfa96f00ad9f27c383fc9ac7f&amp;quot;</ETag>
    <Size>4</Size>
    <StorageClass>STANDARD</StorageClass>
     <Owner>
<ID>bcaf1ffd86f41caff1a493dc2ad8c2c281e37522a640e161ca5fb16fd081034f</ID>
      <DisplayName>webfile</DisplayName>
    </Owner>
 </Contents>
</ListBucketResult>
```

Elements

See the Common List Request Parameters section for comprehensive information about the list request parameters.

- *`Prefix:`* Limits the response to keys that begin with the indicated prefix. You can use prefixes to separate a bucket into different sets of keys in a way similar to how a file system uses folders. This is an optional argument.

- *`Marker:`* Indicates where in the bucket to begin listing. The list includes only keys that occur alphabetically after marker. This is convenient for pagination: To get the next page of results use the last key of the current page as the marker.The most keys you'd like to see in the response body. The server may return less than this number of keys, but will not return more. This is an optional argument.

- *`Delimiter:`* Causes keys that contain the same string between the prefix and the first occurrence of the delimiter to be rolled up into a single result element in the CommonPrefixes collection. These rolled-up keys are not returned elsewhere in the response.

- *`MaxKeys:`* This optional argument limits the number of results returned in response to your query. Amazon S3 will return at most this number of results, but possibly less. For the purpose of counting MaxKeys, a 'result' is either a key in the 'Contents' collection, or a delimited prefix in the

'PrefixRollup' collection.

Response Body

See the Common List Response Elements section for information about the list response.

Access Control

To list the keys of a bucket you need to have been granted `READ` access on the bucket.

GetBucketAccessControlPolicy

The `GetBucketAccessControlPolicy` operation fetches the access control policy for a bucket.

Retrieve the access control policy for the "quotes" bucket:

Sample Request

```
<GetBucketAccessControlPolicy xm-
lns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Bucket>quotes</Bucket>
  <AWSAccessKeyId>1D9FVRAYCP1VJS767E02</AWSAccessKeyId>
  <Timestamp>2005-01-31T23:59:59.183Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbq7RrtSFmw=</Signature>
</GetBucketAccessControlPolicy>
```

Sample Response

```
<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>
    <Grant>
      <Grantee xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/global/AllUsers<URI>
      </Grantee>
      <Permission>READ</Permission>
    </Grant>
  </AccessControlList>
<AccessControlPolicy>
```

Response Body

The response contains the access control policy for the bucket. See Using Amazon S3 for an explanation of this response.

Access Control

You must have READ_ACP rights to the bucket in order to retrieve the access control policy for a bucket.

SetBucketAccessControlPolicy

The SetBucketAccessControlPolicy operation sets the Access Control Policy for an existing bucket. If successful, the previous Access Control Policy for the bucket is entirely replaced with the specified Access Control Policy.

Give the specified user (usually the owner) FULL_CONTROL access to the "quotes" bucket.

Sample Request

```
<SetBucketAccessControlPolicy xm-
lns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Bucket>quotes</Bucket>
  <AccessControlList>
    <Grant>
      <Grantee xsi:type="CanonicalUser">
<ID>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666e9</ID>
        <DisplayName>chriscustomer</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
  </AccessControlList>
  <AWSAccessKeyId>1D9FVRAYCP1VJS767E02</AWSAccessKeyId>
  <Timestamp>2005-01-31T23:59:59.183Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbq7RrtSFmw=</Signature>
</SetBucketAccessControlPolicy >
```

Sample Response

```
<GetBucketAccessControlPolicyResponse xm-
lns="http://s3.amazonaws.com/doc/2006-03-01/">
  <GetBucketAccessControlPolicyResponse>
    <Code>200</Code>
    <Description>OK</Description>
  </GetBucketAccessControlPolicyResponse>
</GetBucketAccessControlPolicyResponse>
```

Access Control

You must have WRITE_ACP rights to the bucket in order to set the access control policy for a bucket.

Operations on Objects

The following operations can be performed on Objects:

- PutObjectInline
- PutObject
- GetObject
- GetObjectExtended
- DeleteObject
- GetObjectAccessControlPolicy
- SetObjectAccessControlPolicy

PutObjectInline

The `PutObjectInline` operation adds an object to a bucket. The data for the object is provided in the body of the SOAP message.

If the object already exists in the bucket, the new object overwrites the existing object. S3 orders all of the requests that it receives. It is possible that if you send two requests nearly simultaneously, we will receive them in a different order than they were sent. The last request received is the one which is stored in S3. Note that this means if multiple parties are simultaneously writing to the same object, they may all get a successful response even though only one of them wins in the end. This is because S3 is a distributed system and it may take a few seconds for one part of the system to realize that another part has received an object update. In this release of Amazon S3, there is no ability to lock an object for writing -- such functionality, if required, should be provided at the application layer.

Write some text and metadata into the "Nelson" object in the "quotes" bucket, give a user (usually the owner) `FULL_CONTROL` access to the object, and make the object readable by anonymous parties:

Sample Request

```
<PutObjectInline xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Bucket>quotes</Bucket>
  <Key>Nelson</Key>
  <Metadata>
    <Name>Content-Type</Name>
    <Value>text/plain</Value>
  </Metadata>
  <Metadata>
    <Name>family</Name>
    <Value>Muntz</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>1D9FVRAYCP1VJS767E02</AWSAccessKeyId>
  <Timestamp>2005-01-31T23:59:59.183Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbq7RrtSFmw=</Signature>
</PutObjectInline>
```

Sample Response

```
<PutObjectInlineResponse xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <PutObjectInlineResponse>
    <ETag>&quot828ef3fdfa96f00ad9f27c383fc9ac7f&quot</ETag>
    <LastModified>2005-11-18T00:27:11.991Z</lastModified>
  </PutObjectInlineResponse>
</PutObjectInlineResponse>
```

Elements

- *`Bucket:`* The bucket in which to add the object.
- *`Key:`* The key to assign to the object.
- *`Metadata:`* You can provide name-value metadata pairs in the metadata element. These will be stored with the object.
- *`Data:`* The base 64 encoded form of the data.
- *`ContentLength:`* The length of the data in bytes.
- *`AccessControlList:`* An Access Control List for the resource. This element is optional. If omitted, the requestor is given `FULL_CONTROL` access to the object. If the object already exists, the pre-existing access control policy is replaced.

Response

- *`ETag:`* The entity tag is an MD5 hash of the object that you can use to do conditional fetches of the object using `GetObjectExtended`. You can also use this value to compare to your own calculated MD5 hash to ensure the object wasn't corrupted over the wire.
- *`LastModified:`* The Amazon S3 timestamp for the saved object.

Access Control

You must have `WRITE` access to the bucket in order to put objects into the bucket.

PutObject

The `PutObject` operation adds an object to a bucket. The data for the object is attached as a DIME attachment.

If the object already exists in the bucket, the new object overwrites the existing object. S3 orders all of the requests that it receives. It is possible that if you send two requests nearly simultaneously, we will receive them in a different order than they were sent. The last request received is the one which is stored in S3. Note that this means if multiple parties are simultaneously writing to the same object, they may all get a successful response even though only one of them wins in the end. This is because S3 is a distributed system and it may take a few seconds for one part of the system to realize that another part has received an object update. In this release of Amazon S3, there is no ability to lock an object for writing -- such functionality, if required, should be provided at the application layer.

Put some data and metadata in the "Nelson" object of the "quotes" bucket, give a user (usually the owner) `FULL_CONTROL` access to the object, and make the object readable by anonymous parties. In this sample, the actual attachment is not shown:

Sample Request

```
<PutObject xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Bucket>quotes</Bucket>
  <Key>Nelson</Key>
  <Metadata>
    <Name>Content-Type</Name>
    <Value>text/plain</Value>
  </Metadata>
  <Metadata>
    <Name>family</Name>
    <Value>Muntz</Value>
  </Metadata>
  <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>1D9FVRAYCP1VJS767E02</AWSAccessKeyId>
  <Timestamp>2005-01-31T23:59:59.183Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbq7RrtSFmw=</Signature>
</PutObject>
```

Sample Response

```
<PutObjectResponse xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <PutObjectResponse>
    <ETag>&amp;quot;828ef3fdfa96f00ad9f27c383fc9ac7f&amp;quot;</ETag>
    <LastModified>1970-01-01T00:00:00.000Z</LastModified>
  </PutObjectResponse>
</PutObjectResponse>
```

Elements

- *Bucket:* The bucket in which to add the object.
- *Key:* The key to assign to the object.
- *Metadata:* You can provide name-value metadata pairs in the metadata element. These will be stored with the object.
- *ContentLength:* The length of the data in bytes.
- *AccessControlList:* An Access Control List for the resource. This element is optional. If omitted, the requestor is given FULL_CONTROL access to the object. If the object already exists, the pre-existing Access Control Policy is replaced.

Response

- *ETag:* The entity tag is an MD5 hash of the object that you can use to do conditional fetches of the object using GetObjectExtended. You can also use this value to compare to your own calculated MD5 hash to ensure the object wasn't corrupted over the wire.
- *LastModified:* The S3 timestamp for the saved object.

Access Control

You must have WRITE access to the bucket in order to put objects into the bucket.

GetObject

GetObject is the basic operation for retrieving an object stored in Amazon S3. If you want a few more bells and whistles, check out GetObjectExtended.

Get the "Nelson" object from the "quotes" bucket:

Sample Request

```
<GetObject xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Bucket>quotes</Bucket>
  <Key>Nelson</Key>
  <GetMetadata>true</GetMetadata>
  <GetData>true</GetData>
  <InlineData>true</InlineData>
  <AWSAccessKeyId>1D9FVRAYCP1VJS767E02</AWSAccessKeyId>
  <Timestamp>2005-01-31T23:59:59.183Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbq7RrtSFmw=</Signature>
</GetObject>
```

Sample Response

```
<GetObjectResponse xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <GetObjectResponse>
    <Status>
      <Code>200</Code>
```

```
      <Description>OK</Description>
    </Status>
    <Metadata>
      <Name>Content-Type</Name>
      <Value>text/plain</Value>
    </Metadata>
    <Metadata>
      <Name>family</Name>
      <Value>Muntz</Value>
    </Metadata>
    <Data>aGEtaGE=</Data>
    <LastModified>2005-11-18T01:25:07.994Z</LastModified>
    <ETag>&amp;quot;828ef3fdfa96f00ad9f27c383fc9ac7f&amp;quot;</ETag>
  </GetObjectResponse>
</GetObjectResponse>
```

Elements

- *Bucket:* The bucket from which to retrieve the object.
- *Key:* The key that identifies the object.
- *GetMetadata:* The metadata is returned with the object if this is true.
- *GetData:* The object data is returned if this is true.
- *InlineData:* If this is true, then the data is returned, base 64-encoded, as part of the SOAP body of the response. If false, then the data is returned as a SOAP attachment.

Returned Elements

- *Metadata:* The name-value paired metadata stored with the object.
- *Data:* If InlineData was true in the request, this contains the base 64 encoded object data.
- *LastModified:* The time that the object was stored in Amazon S3.
- *ETag:* The object's entity tag. This is a hash of the object that can be used to do conditional gets.

Access Control

You can read an object only if you have been granted `READ` access to the object.

GetObjectExtended

`GetObjectExtended` is exactly like GetObject, except that it supports the following additional elements that can be used to accomplish much of the same functionality provided by HTTP GET headers (http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html).

GetObjectExtended supports the following elements in addition to those supported by GetObject:

- *ByteRangeStart, ByteRangeEnd:* These elements specify that only a portion of the object data

should be retrieved. They follow the behavior of the HTTP byte ranges (http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.35).

- *IfModifiedSince:* Return the object only if the object's timestamp is later than the specified timestamp. (http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.25)

- *IfUnmodifiedSince:* Return the object only if the object's timestamp is earlier than or equal to the specified timestamp. (http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.28)

- *IfMatch:* Return the object only if its ETag matches the supplied tag(s). (http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.24)

- *IfNoneMatch:* Return the object only if its ETag does not match the supplied tag(s). (http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.26)

- *ReturnCompleteObjectOnConditionFailure:*ReturnCompleteObjectOnConditionFailure: If true, then if the request includes a range element and one or both of IfUnmodifiedSince/IfMatch elements, and the condition fails, return the entire object rather than a fault. This enables the If-Range functionality described here: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.27

DeleteObject

The `DeleteObject` operation removes the specified object from S3. Once you've deleted it, there is no going back: S3 does not support any form of undelete.

Delete the "Nelson" object from the "quotes" bucket:

Sample Request

```
<DeleteObject xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Bucket>quotes</Bucket>
  <Key>Nelson</Key>
  <AWSAccessKeyId> 1D9FVRAYCP1VJS767E02</AWSAccessKeyId>
  <Timestamp>2005-01-31T23:59:59.183Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbq7RrtSFmw=</Signature>
</DeleteObject>
```

Sample Response

```
<DeleteObjectResponse xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <DeleteObjectResponse>
    <Code>200</Code>
    <Description>OK</Description>
  </DeleteObjectResponse>
</DeleteObjectResponse>
```

Elements

- *Bucket:* The bucket that holds the object.

- *Key:* The key that identifies the object.

Access Control

You can delete an object if, and only if, you have WRITE access to the bucket. It does not matter who owns the object, or what rights have been granted to the object.

GetObjectAccessControlPolicy

The `GetObjectAccessControlPolicy` operation fetches the access control policy for an object.

Retrieve the access control policy for the "Nelson" object from the "quotes" bucket:

Sample Request

```
<GetObjectAccessControlPolicy xm-
lns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Bucket>quotes</Bucket>
  <Key>Nelson</Key>
  <AWSAccessKeyId>1D9FVRAYCP1VJS767E02</AWSAccessKeyId>
  <Timestamp>2005-01-31T23:59:59.183Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbq7RrtSFmw=</Signature>
</GetObjectAccessControlPolicy>
```

Sample Response

```
<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>
    <Grant>
      <Grantee xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/global/AllUsers<URI>
      </Grantee>
      <Permission>READ</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

Response Body

The response contains the access control policy for the bucket. See Using Amazon S3 for an explanation of this response.

Access Control

You must have `READ_ACP` rights to the object in order to retrieve the access control policy for an object.

SetObjectAccessControlPolicy

The `SetObjectAccessControlPolicy` operation sets the access control policy for an existing object. If successful, the previous access control policy for the object is entirely replaced with the specified access control policy.

Give the specified user (usually the owner) `FULL_CONTROL` access to the "Nelson" object from the "quotes" bucket:

Sample Request

```
<SetObjectAccessControlPolicy xm-
lns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Bucket>quotes</Bucket>
  <AccessControlList>
    <Grant>
      <Grantee xsi:type="CanonicalUser">
<ID>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666e9</ID>
        <DisplayName>chriscustomer</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
  </AccessControlList>
  <AWSAccessKeyId>1D9FVRAYCP1VJS767E02</AWSAccessKeyId>
  <Timestamp>2005-01-31T23:59:59Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbq7RrtSFmw=</Signature>
</SetObjectAccessControlPolicy>
```

Sample Response

```
<SetObjectAccessControlPolicyResponse xm-
lns="http://s3.amazonaws.com/doc/2006-03-01/">
  <SetObjectAccessControlPolicyResponse>
    <Code>200</Code>
    <Description>OK</Description>
  </SetObjectAccessControlPolicyResponse>
</SetObjectAccessControlPolicyResponse>
```

Access Control

You must have `WRITE_ACP` rights to the object in order to set the access control policy for a bucket.

Using BitTorrent™ with S3

BitTorrent™ is an open, peer-to-peer protocol for distributing files, invented by Bram Cohen. You can use the BitTorrent protocol to retrieve any publicly-accessible object in S3. This section describes why you might want to use BitTorrent to distribute your data out of S3 and how to do so.

Why BitTorrent?

S3 supports the BitTorrent protocol so that developers can save costs when distributing content at high scale. S3 is useful for simple, reliable storage of any data. The default distribution mechanism for S3 data is via client/server download. In client/server distribution, the entire object is transferred point-to-point from 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 downloaders 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 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 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.

How You are Charged for BitTorrent Delivery

There is no extra charge for use of BitTorrent with 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 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 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 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 may 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 S3

Any object in 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. S3's BitTorrent implementation has been tested to work with the official BitTorrent client, available at 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 S3 generated .torrent file, it should start downloading immediately from 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 may be as easy as opening this URL in your web browser.

There is no mechanism to fetch the .torrent for an S3 object using the SOAP API.

Retrieve the Torrent file for the "Nelson" object in the "quotes" bucket:

Sample Request

```
GET /quotes/Nelson?torrent HTTP/1.0
Date: Thu, 17 Nov 2005 08:22:38 GMT
```

Sample Response

```
HTTP/1.1 200 OK
x-amz-request-id: 7CD745EBB7AB5ED9
Date: Wed, 08 Mar 2006 17:19:29 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 S3 and BitTorrent

Every anonymously readable object stored in 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 the Access Control Policy topic.

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 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. S3 may 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 S3, or modifying your access control policy to prohibit anonymous reads. After doing so, 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, once a .torrent for your file has been published, this action may not be sufficient to stop public downloads of your object that happen exclusively using the BitTorrent peer to peer network.