Welcome to Amazon S3

Topics

- Who Should Read this Guide
- How to Give Us Feedback
- How This Guide Is Organized
- <u>Amazon S3 Resources</u>

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

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

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

Who Should Read this Guide

This guide has two audiences:

• Developers creating libraries to implement the Amazon S3 API

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

• Developers using libraries created to implement the Amazon S3 API

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

Required Knowledge and Skills

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

- XML (go to <u>W3 Schools XML Tutorial</u>)
- Basic understanding of web services (go to <u>W3 Schools Web Services</u> <u>Tutorial</u>))
- A programming language for consuming a web service and any related tools

You should also have read the *Amazon S3 Getting Started Guide*. For more information, go to *Amazon S3 Getting Started Guide*.

How to Give Us Feedback

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

Feedback	9
	Feedback

How This Guide Is Organized

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

Information	Relevant Sections
General information about Amazon S3	Introduction to Amazon S3
Conceptual information about Amazon S3	Introduction to Amazon S3
Information about making requests	Making Requests
Information about how to use the Amazon S3 REST operations	Common REST API Elements
Information about how to use the Amazon S3 SOAP operations	Common SOAP API Elements
Information about using DevPay with Amazon S3	<u>Using Amazon DevPay with</u> <u>Amazon S3</u>
Information about handling errors	Handling Errors
Information about BitTorrent	Using BitTorrent with Amazon S3
Typographic and symbol conventions	Document Conventions

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

Amazon S3 Resources

Following is a table that lists related resources that you'll find useful as you work with this service.

Resource	Description
<u>Amazon S3 Getting</u> <u>Started Guide</u>	The Getting Started Guide provides a quick tutorial of the service based on a simple use case. Examples and instructions for Java, Perl, PHP, C#, Python, and Ruby are included.
<u>Amazon S3 API</u> <u>Reference</u>	The API Reference describes Amazon S3 operations in detail.
Amazon S3Technical FAQ	The FAQ covers the top 20 questions developers have asked about this product.
<u>Amazon S3 Release</u> <u>Notes</u>	The Release Notes give a high-level overview of the current release. They specifically note any new features, corrections, and known issues.
AWS Developer Resource Center	A central starting point to find documentation, code samples, release notes, and other information to help you build innovative applications with AWS.
AWS Management Console	The console allows you to perform most of the functions of Amazon S3without programming.
Discussion Forums	A community-based forum for developers to discuss technical questions related to Amazon Web Services.
AWS Support Center	The home page for AWS Technical Support, including access to our Developer Forums, Technical FAQs, Service Status page, and Premium Support.
<u>AWS Premium</u> Support	The primary web page for information about AWS Premium Support, a one- on-one, fast-response support channel to help you build and run applications on AWS Infrastructure Services.
Amazon S3 product information	The primary web page for information about Amazon S3.
Contact Us	A central contact point for inquiries concerning AWS billing, account, events, abuse etc.
Conditions of Use	Detailed information about the copyright and trademark usage at Amazon.com and other topics.

What's New

This What's New is associated with the 2006-03-01 release of Amazon S3. This guide was last updated on November 11, 2009.

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

Change	Description	Date
AWS SDK for .NET	AWS now provides libraries, sample code, tutorials, and other resources for software developers who prefer to build applications using .NET language-specific APIs instead of REST or SOAP. These libraries provide basic functions (not included in the REST or SOAP APIs), such as request authentication, request retries, and error handling so that it's easier to get started. For more information about language-specific libraries and resources, see <u>AWS Library</u> <u>Support</u> , or go to <u>Working With Amazon S3</u> in the Amazon Simple Storage Service Getting Started Guide.	2009- 11-11
Technical documents reorganized	The API reference has been split out of the <i>Amazon S3 Developer Guide</i> . Now, on the documentation landing page, http://developer.amazonwebservices.com/connect/entry.jspa? externalID=123&categoryID=48 you can select the document you want to view. When viewing the documents online, the links in one document will take you, when appropriate, to one of the other guides.	2009- 9-16

Introduction to Amazon S3

Topics

•

Overview of Amazon S3

- Advantages to Amazon S3
- <u>Amazon S3 Concepts</u>
- Paying for Amazon S3
- <u>Related Amazon Web Services</u>

This introduction to Amazon S3 is intended to give you a detailed summary of this web service. After reading this section, you should have a good idea of what it offers and how it can fit in with your business.

Overview of Amazon S3

Amazon S3 is storage for the Internet. It is designed to make web-scale computing easier for developers.

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

Advantages to Amazon S3

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

• Create Buckets—Create and name a bucket that stores data

Buckets are the fundamental container in Amazon S3 for data storage.

• Store data in Buckets—Store an infinite amount of data in a bucket

Upload as many objects as you like into an Amazon S3 bucket. Each object can contain up to 5 GB of data. Each object is stored and retrieved using a unique developer-assigned key.

• Download data—Download your data or enable others to

Download your data any time you like or allow others to do the same.

• **Permissions**—Grant or deny access to others who want to upload or download data into your Amazon S3 bucket

Grant upload and download permissions to three types of users. Authentication mechanisms to ensure that data is kept secure from unauthorized access.

• **Standard interfaces**—Use standards-based REST and SOAP interfaces designed to work with any Internet-development toolkit.

Amazon S3 Concepts

Topics

- <u>Buckets</u>
- <u>Objects</u>
- <u>Keys</u>
- <u>Operations</u>
- <u>Amazon S3 Application Programming Interfaces (API)</u>
- <u>Amazon S3 Data Consistency Model</u>

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

Buckets

A bucket is 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.

For more information about buckets, see <u>Working with Amazon S3 Buckets</u>.

Objects

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

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 web service endpoint, bucket name, and key, as in http://doc.s3.amazonaws.com/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. Following are the most common operations you'll execute through the API.

Common Operations

- **Create a Bucket**—Create and name your own bucket in which to store your objects.
- Write an Object—Store data by creating or overwriting an object.

When you write an object, you specify a unique key in the namespace of your bucket. This is also a good time to specify any access control you want on the object.

• Read an Object—Read data back.

You can choose to download the data via HTTP or BitTorrent.

- **Deleting an Object**—Delete some of your data.
- Listing Keys—List the keys contained in one of your buckets.

You can filter the key list based on a prefix.

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

Amazon S3 Application Programming Interfaces (API)

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

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

The REST Interface

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

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

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

The SOAP Interface

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

Amazon S3 Data Consistency Model

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

Amazon S3 achieves high availability by replicating data across multiple servers within Amazon's data centers. After a "success" is returned, your data is safely stored. However, information about the changes might not immediately replicate across Amazon S3 and you might observe the following behaviors:

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



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

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

Related Amazon Web Services

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

• Amazon ElasticCompute Cloud—This web service provides virtual compute resources in the cloud.

For more information, go to <u>Amazon ElasticCompute Cloud</u>.

• Amazon Elastic MapReduce—This web service enables businesses, researchers, data analysts, and developers to easily and cost-effectively process vast amounts of data.

It utilizes a hosted Hadoop framework running on the web-scale infrastructure of Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3). For more information, go to <u>Amazon Elastic MapReduce</u>.

 Amazon Import/Export—This service enables you to mail a storage device, such as a RAID drive, to Amazon so that we can upload your (terabytes) of data onto Amazon S3. For more information, go to <u>AWS</u> <u>Import/Export Developer Guide</u>>.

Paying for Amazon S3

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

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

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

For information about paying for Amazon S3 storage, go to the <u>AWS Resource</u> <u>Center</u>.

Making Requests

Topics

- <u>AWS Language Support</u>
- <u>Request Endpoints</u>
- <u>Using the REST API</u>
- <u>Using the SOAP API</u>

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

AWS Language Support

AWS provides libraries, sample code, tutorials, and other resources for software developers who prefer to build applications using language-specific APIs instead of Amazon S3's SOAP and REST APIs. These libraries provide basic functions (not included in Amazon S3's SOAP and REST APIs), such as request authentication, request retries, and error handling so that it's easier to get started. For more information about language-specific libraries and resources, go to:

- <u>Java</u>
- <u>PHP</u>
- <u>Ruby</u>
- Windows and .NET

For libraries and sample code in all languages, go to <u>Sample Code & Libraries</u>.

Request Endpoints

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

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

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

Using the REST API

Topics

•

Common REST API Elements

- <u>Authenticating REST Requests</u>
- <u>REST Access Control Policy</u>
- <u>Virtual Hosting of Buckets</u>
- Request Redirection and the REST API
- Browser-Based Uploads Using POST

This section contains information specific to the Amazon S3 REST API. The examples in this guide use the newer virtual hosted-style method for accessing buckets instead of the path-style. Although the path-style is still supported for legacy applications, we recommend using the virtual-hosted style where applicable. For more information, see <u>Working with Amazon S3 Buckets</u>

For a list of REST endpoints, see <u>How to Select a Region for Your Buckets</u>.

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

```
DELETE /puppy.jpg HTTP/1.1
User-Agent: dotnet
Host: mybucket.s3.amazonaws.com
Date: Tue, 15 Jan 2008 21:20:27 +0000
x-amz-date: Tue, 15 Jan 2008 21:20:27 +0000
Authorization: AWS 0PN5J17HBGZHT7JJ3X82:k3nL7gH3+PadhTEVn5EXAMPLE
```

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

```
DELETE /mybucket/puppy.jpg HTTP/1.1
User-Agent: dotnet
Host: s3.amazonaws.com
Date: Tue, 15 Jan 2008 21:20:27 +0000
x-amz-date: Tue, 15 Jan 2008 21:20:27 +0000
Authorization: AWS 0PN5J17HBGZHT7JJ3X82:k3nL7gH3+PadhTEVn5EXAMPLE
```

Common REST API Elements

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

All REST requests use a common set of endpoints. For information, see <u>Request</u> <u>Endpoints</u>.

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

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

Sample Request

```
GET /Nelson HTTP/1.1
Host: quotes.s3.amazonaws.com
Date: Wed, 01 Mar 2006 12:00:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

Sample Response

```
HTTP/1.1 200 OK
x-amz-id-2: qBmKRcEWBBhH6XAqsKU/eg24V3jf/kWKN9dJip1L/FpbYr9FDy7wWFurfdQ(
x-amz-request-id: F2A8CCCA26B4B26D
Date: Wed, 01 Mar 2006 12:00:00 GMT
Last-Modified: Sun, 1 Jan 2006 12:00:00 GMT
ETag: "828ef3fdfa96f00ad9f27c383fc9ac7f"
Content-Type: text/plain
Content-Length: 5
Connection: close
Server: AmazonS3
```

ha-ha

Common Request Headers

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

Header Name	Description	Required
Content- Length	Length of the message (without the headers) according to RFC 2616. Condition: Required for PUTs and operations that load XML, such as logging and ACLs.	
Content-Type	The content type of the resource. Example: text/plain	No
Date	The current date and time according to the requester. Example: Wed, 01 Mar 2006 12:00:00 GMT	Yes
Host	Normally, 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 <u>Virtual Hosting of Buckets</u> . Condition: Required for HTTP 1.1 (most toolkits add this header	
	automatically); optional for HTTP/1.0 requests.	
Authorization	ⁿ The information required for request authentication. For more information, see <u>The Authentication Header</u> for details about the format.	
x-amz- security- token	The security tokens for operations that use Amazon DevPay. Each request that uses Amazon DevPay requires two x-amz-security-token headers: one for the product token and one for the user token.	Conditional
	Condition: Required for requests that use Amazon DevPay.	
	渗 Note	
	When Amazon S3 receives an	
	authenticated request, it compares	
	the computed signature with the	
	provided signature. Improperly formatted multi-value headers	
	used to calculate a signature can	
	cause authentication issues. To	
	ensure the signature is calculated properly, follow the instructions in	

the <u>Constructing the</u> <u>CanonicalizedResource Element</u> section.	
---	--

Authenticating REST Requests

Topics

•

The Authentication Header

- Request Canonicalization for Signing
- <u>Constructing the CanonicalizedResource Element</u>
- <u>Constructing the CanonicalizedAmzHeaders Element</u>
- <u>Positional versus Named HTTP Header StringToSign Elements</u>
- <u>Time Stamp Requirement</u>
- <u>Authentication Examples</u>
- <u>REST Request Signing Problems</u>
- Query String Request Authentication Alternative

Authentication is the process of proving your identity to the system. Identity is an important factor in Amazon S3 access control decisions. Requests are allowed or denied in part based on the identity of the requester. For example, the right to create buckets is reserved for registered developers and (by default) the right to create objects in a bucket is reserved for the owner of the bucket in question. As a developer, you'll be making requests that invoke these privileges so you'll need to prove your identity to the system by authenticating your requests. This section shows you how.



The content in this section does not apply to HTTP POST. For more information, see <u>Browser-Based Uploads Using POST</u>.

The Amazon S3 REST API uses a custom HTTP scheme based on a keyed-HMAC (Hash Message Authentication Code) for authentication. To authenticate a request, you first concatenate selected elements of the request to form a string. You then use your AWS Secret Access Key to calculate the HMAC of that string. Informally, we call this process "signing the request," and we call the output of the HMAC algorithm the "signature" because it simulates the security properties of a real signature. Finally, you add this signature as a parameter of the request, using the syntax described in this section. When the system receives an authenticated request, it fetches the AWS Secret Access Key that you claim to have, and uses it in the same way to compute a "signature" for the message it received. It then compares the signature it calculated against the signature presented by the requester. If the two signatures match, then the system concludes that the requester must have access to the AWS Secret Access Key, and therefore acts with the authority of the principal to whom the key was issued. If the two signatures do not match, the request is dropped and the system responds with an error message.

Example Authenticated Amazon S3 REST Request

GET /photos/puppy.jpg HTTP/1.1
Host: johnsmith.s3.amazonaws.com
Date: Mon, 26 Mar 2007 19:37:58 +0000

Authorization: AWS 0PN5J17HBGZHT7JJ3X82:frJIUN8DYpKDt0LCwo//yllqDzg=

The Authentication Header

The Amazon S3 REST API uses the standard HTTPAuthorization header to pass authentication information. (The name of the standard header is unfortunate, sinceit carries authentication information, not authorization).Under the Amazon S3 authentication scheme, the Authorization header has the following form.

Authorization: AWS AWSAccessKeyId:Signature

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

The *Signature* element is the RFC 2104HMAC-SHA1 of selected elements from the request, and so the*Signature* part of the Authorization header will vary from request to request. If the request signature calculated by the system matches the*Signature* included with the request, then the requester will have demonstrated possession to the AWSSecret Access Key. The request will then be processed under the identity, and with the authority, of the developer to whom the key was issued.

Following is pseudo-grammar that illustrates the construction of the *Authorization* request header (\nmeans the Unicode code point U+000A commonly called newline).

```
Authorization = "AWS" + " " + AWSAccessKeyId + ":" + Signature;
Signature = Base64( HMAC-SHA1( UTF-8-Encoding-Of( YourSecretAccessKeyID,
StringToSign = HTTP-Verb + "\n" +
Content-MD5 + "\n" +
Content-Type + "\n" +
Date + "\n" +
CanonicalizedAmzHeaders +
CanonicalizedAmzHeaders +
CanonicalizedResource;
```

```
<HTTP-Request-URI, from the protocol name up to the query string
[ sub-resource, if present. For example "?acl", "?location", "?lo
```

HMAC-SHA1 is an algorithm defined by RFC 2104 (go to <u>RFC 2104 - Keyed-Hashing for Message Authentication</u>). The algorithm takes as input two bytestrings: a key and a message. For Amazon S3 Request authentication, use your AWS Secret Access Key (*YourSecretAccessKeyID*) as the key, and the UTF-8 encoding of the *StringToSign* as the message. The output of HMAC-SHA1 is also a byte string, called the digest. The *Signature* request parameter is constructed by Base64 encoding this digest.

Request Canonicalization for Signing

Recall that when the system receives an authenticated request, it compares the computed request signature with the signature provided in the request. In order for the system-computed signature to match the developer-computed signature, the *StringToSign* for a request must be constructed by both parties in exactly the same way. We call the process of putting a request in an agreed-upon form for signing "canonicalization".

Constructing the CanonicalizedResource Element

CanonicalizedResource represents the Amazon S3 resource targeted by the request. Construct it for a REST request as follows:

Launch Process

$1 \mid S$	Start with	the	empty	string	("").
------------	------------	-----	-------	--------	-------

- 2 If the request specifies a bucket using the HTTP Host header (virtual hosted-style), append the bucket name preceded by a "/" (e.g., "/bucketname"). For path-style requests and requests that don't address a bucket, do nothing. For more information on virtual hosted-style requests, see <u>Virtual Hosting of Buckets</u>.
- 3 Append the path part of the un-decoded HTTP Request-URI, up-to but not including the query string.
- 4 If the request addresses a sub-resource, like ?location, ?acl, or ?torrent, append the sub-resource including question mark.

Elements of the CanonicalizedResource that come from the HTTP Request-URI should be signed literally as they appear in the HTTP request, including URL-Encoding metacharacters.

The *CanonicalizedResource* might be different than the HTTP Request-URI. In particular, if your request uses the HTTP Host header to specify a bucket, the bucket does appear in the HTTP Request-URI. However, the *CanonicalizedResource* continues to include the bucket. Query string parameters other than sub-resource flags (e.g., "?acl", "?location", "?logging", or "?torrent") will also appear in the Request-URI but are not included in *CanonicalizedResource*. For more information, see <u>Virtual Hosting of Buckets</u>.

Constructing the CanonicalizedAmzHeaders Element

To construct the CanonicalizedAmzHeaders part of *StringToSign*, select all HTTP request headers that start with 'x-amz-' (using a case-insensitive comparison) and use the following process.

CanonicalizedAmzHeaders Process

1	Convert each HTTP header name to lower-case. For example, 'X-Amz-Date' becomes 'x-amz-date'.
2	Sort the collection of headers lexicographically by header name.
3	Combine header fields with the same name into one "header-name:comma-separated-value-list" pair as prescribed by RFC 2616, section 4.2, without any white-space between values. For example, the two metadata headers 'x-amz-meta-username: fred' and 'x-amz-meta-username: barney' would be combined into the single header 'x-amz-meta-username: fred, barney'.
4	"Unfold" long headers that span multiple lines (as allowed by RFC 2616, section 4.2) by replacing the folding white-space (including new-line) by a single space.
5	Trim any white-space around the colon in the header. For example, the header 'x-amz-meta-username: fred, barney' would become 'x-amz-meta-username:fred, barney'
6	Finally, append a new-line (U+000A) to each canonicalized header in the resulting list. Construct the CanonicalizedResource element by concatenating all headers in this list into a single string.

Positional versus Named HTTP Header StringToSign Elements

The first few header elements of *StringToSign* (Content-Type, Date, and Content-MD5) are positional in nature. *StringToSign* does not include the names of these headers, only their values from the request. In contrast, the 'x-amz-' elements are named; Both the header names and the header values appear in *StringToSign*.

If a positional header called for in the definition of *StringToSign* is not present in your request, (Content-Type or Content-MD5, for example, are optional for PUT requests, and meaningless for GET requests), substitute the empty string ("") in for that position.

Time Stamp Requirement

A valid time stamp (using either the HTTP Date header or an x-amz-date alternative) is mandatory for authenticated requests. Furthermore, the client time-stamp included with an authenticated request must be within 15 minutes of the Amazon S3 system time when the request is received. If not, the request will fail with the *RequestTimeTooSkewed* error status code. The intention of these restrictions is to limit the possibility that intercepted requests could be replayed by an adversary. For stronger protection against eavesdropping, use the HTTPS transport for authenticated requests.

Some HTTP client libraries do not expose the ability to set the Date header for a request. If you have trouble including the value of the 'Date' header in the canonicalized headers, you can set the time-stamp for the request using an 'x-amz-date' header instead. The value of the x-amz-date header must be in one of the RFC 2616 formats (http://www.ietf.org/rfc/rfc2616.txt). When an x-amz-date header is present in a request, the system will ignore any Date header when computing the request signature. Therefore, if you include the x-amz-date header, use the empty string for the Date when constructing the *StringToSign*. See the next section for an example.

Authentication Examples

The examples in this section use the (non-working) credentials in the following table.

Parameter	Value
AWSAccessKeyId	0PN5J17HBGZHT7JJ3X82
AWSSecretAccessKey	uV3F3YluFJax1cknvbcGwgjvx4QpvB+leU8dUj2o

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

Example Object GET

This example gets an object from the johnsmith bucket.

Request	StringToSign	
GET /photos/puppy.jpg HTTP/1.1	GET\n	
Host: johnsmith.s3.amazonaws.com	\n	
Date: Tue, 27 Mar 2007 19:36:42 +0000	\n	
Authorization: AWS 0PN5J17HBGZHT7JJ3X82:	Tue, 27 Mar 2007 19:36:42 +000	
xXjDGYUmKxnwqr5KXNPGldn5LbA=	/johnsmith/photos/puppy.jpg	

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

Example Object PUT

This example puts an object into the johnsmith bucket.

Request StringToSign

<pre>PUT /photos/puppy.jpg HTTP/1.1 Content-Type: image/jpeg Content-Length: 94328 Host: johnsmith.s3.amazonaws.com Date: Tue, 27 Mar 2007 21:15:45 +0000 Authorization: AWS 0PN5J17HBGZHT7JJ3X82: hcicpDDvL9Ss06AkvxqmIWkmOuQ=</pre>	PUT\n \n image/jpeg\n Tue, 27 Mar 2007 21:15:45 +0000 /johnsmith/photos/puppy.jpg
---	---

Note the Content-Type header in the request and in the StringToSign. Also note that the Content-MD5 is left blank in the StringToSign since it is not present in the request.

Example List

This example lists the content of the johnsmith bucket.

Request	StringToSign
<pre>GET /?prefix=photos&max-keys=50▮=puppy HTTP/1.1 User-Agent: Mozilla/5.0 Host: johnsmith.s3.amazonaws.com Date: Tue, 27 Mar 2007 19:42:41 +0000 Authorization: AWS 0PN5J17HBGZHT7JJ3X82:jsRt/rhG+Vtp88HrYL706QhE4w4=</pre>	GET\n \n \n Tue, 27 Mar 2 /johnsmith/

Note the trailing slash on the CanonicalizedResource, and the absence of query string parameters.

Example Fetch

This example fetches the access control policy sub-resource for the 'johnsmith' bucket.

Request	StringToSign
<pre>GET /?acl HTTP/1.1 Host: johnsmith.s3.amazonaws.com Date: Tue, 27 Mar 2007 19:44:46 +0000 Authorization: AWS 0PN5J17HBGZHT7JJ3X82:thdUi9VAkzhkniLj96JIr0PGi0g=</pre>	GET\n \n \n Tue, 27 Mar 2 /johnsmith/?a

Notice how the sub-resource query string parameter is included in the CanonicalizedResource.

Example Delete

This example deletes an object from the 'johnsmith' bucket using the path-style and Date alternative.

DELETE /johnsmith/photos/puppy.jpg HTTP/1.1DELUser-Agent: dotnet\nHost: s3.amazonaws.com\nDate: Tue, 27 Mar 2007 21:20:27 +0000\n	ingToSign
	l

Note how we used the alternate 'x-amz-date' method of specifying the date (because our client library prevented us from setting the date, say). In this case the field for the actual 'Date' header is left blank in the StringToSign.

Example Upload

This example uploads an object to a CNAME style virtual hosted bucket with metadata.

Request	StringToSign
PUT /db-backup.dat.gz HTTP/1.1 User-Agent: curl/7.15.5 Host: static.johnsmith.net:8080 Date: Tue, 27 Mar 2007 21:06:08 +0000	PUT\n 4gJE4saaMU4Bq application/x Tue, 27 Mar 20
<pre>x-amz-acl: public-read content-type: application/x-download Content-MD5: 4gJE4saaMU4BqNR0kLY+lw== X-Amz-Meta-ReviewedBy: joe@johnsmith.net X-Amz-Meta-ReviewedBy: jane@johnsmith.net X-Amz-Meta-FileChecksum: 0x02661779 X-Amz-Meta-ChecksumAlgorithm: crc32 Content-Disposition: attachment; filename=database.dat Content-Encoding: gzip Content-Length: 5913339</pre>	<pre>x-amz-acl:pub x-amz-meta-che x-amz-meta-fi x-amz-meta-rev joe@johnsmith /static.johnsr</pre>
Authorization: AWS 0PN5J17HBGZHT7JJ3X82: C0Fl0tU8Ylb9KDTpZqYkZPX91iI=	

Notice how the 'x-amz-' headers are sorted, white-space trimmed, converted to lowercase, and multiple headers with the same name have been joined using a comma to separate values.

Note how only the Content-Type and Content-MD5 HTTP entity headers appear in the *StringToSign*. The other Content-* entity headers do not.

Again, note that the *CanonicalizedResource* includes the bucket name, but the HTTP Request-URI does not (the bucket is specified by the Host header).

Example List All My Buckets

Request	StringToSign
GET / HTTP/1.1	GET∖n

Host: s3.amazonaws.com Date: Wed, 28 Mar 2007 01:29:59 +0000 Authorization: AWS 0PN5J17HBGZHT7JJ3X82:Db+gepJSUbZKwpx1FR0DLtEYoZA=	\n \n Wed, /	28 Mar 2

Example Unicode Keys

Request	StringToSign	
<pre>GET /dictionary/fran%C3%A7ais/pr%c3%a9f%c3%a8re HTTP/1.1 Host: s3.amazonaws.com Date: Wed, 28 Mar 2007 01:49:49 +0000 Authorization: AWS 0PN5J17HBGZHT7JJ3X82: dxhSBHoI6eVSPcXJqEghlUzZMnY=</pre>	GET\n \n \n Wed, 28 Mar /dictionary,	



Note

The elements in *StringToSign* that were derived from the Request-URI are taken literally, including URL-Encoding and capitalization.

REST Request Signing Problems

When REST request authentication fails, the system responds to the request with an XML error document. The information contained in this error document is meant to help developers diagnose the problem. In particular, the *StringToSign* element of the *SignatureDoesNotMatch* error document tells you exactly what request canonicalization the system is using.

Some toolkits silently insert headers that you do not know about beforehand, such as adding the header Content-Type during a PUT. In most of these cases, the value of the inserted header remains constant, allowing you to discover the missing headers using tools such as Ethereal or tcpmon.

Query String Request Authentication Alternative

You can authenticate certain types of requests by passing the required information as query-string parameters instead of using the Authorization HTTP header. This is useful for enabling direct third-party browser access to your private Amazon S3 data, without proxying the request. The idea is to construct a "pre-signed" request and encode it as a URL that an end-user's browser can retrieve. Additionally, you can limit a pre-signed request by specifying an expiration time.

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

```
GET /photos/puppy.jpg
?AWSAccessKeyId=0PN5J17HBGZHT7JJ3X82&Expires=1141889120&Signature=vjbyP>
Host: johnsmith.s3.amazonaws.com
Date: Mon, 26 Mar 2007 19:37:58 +0000
```

The query string request authentication method doesn't require any special HTTP headers. Instead, the required authentication elements are specified as query string parameters:

Query String Parameter Name	Example Value	Description
AWSAccessKeyId	0PN5J17HBGZHT7JJ3X82	Your AWS Access Key Id. Specifies the AWS Secret Access Key used to sign the request, and (indirectly) the identity of the developer making the request.
Expires	1141889120	The time when the signature expires, specified as the number of seconds since the epoch (00:00:00 UTC on January 1, 1970). A request received after this time (according to the server), will be rejected.
Signature	vjbyPxybdZaNmGa%2ByT272YEAiv4%3D	The URL encoding of the Base64 encoding of the HMAC-SHA1 of StringToSign.

The query string request authentication method differs slightly from the ordinary

method but only in the format of the *Signature* request parameter and the *StringToSign* element. Following is pseudo-grammar that illustrates the query string request authentication method.

```
Signature = URL-Encode( Base64( HMAC-SHA1( YourSecretAccessKeyID, UTF-8-
StringToSign = HTTP-VERB + "\n" +
Content-MD5 + "\n" +
Content-Type + "\n" +
Expires + "\n" +
CanonicalizedAmzHeaders +
CanonicalizedResource;
```

YourSecretAccessKeyID is the AWS Secret Access Key ID Amazon assigns to you when you sign up to be an Amazon Web Service developer. Notice how the *Signature* is URL-Encoded to make it suitable for placement in the query-string. Also note that in *StringToSign*, the HTTP Date positional element has been replaced with *Expires*. The *CanonicalizedAmzHeaders* and *CanonicalizedResource* are the same.

Example Query String Request Authentication

Request	StringToSign
<pre>GET /photos/puppy.jpg?AWSAccessKeyId=0PN5J17HBGZHT7JJ3X82&</pre>	GET\n
Signature=rucSbH0yNEcP9oM2XNlouVI3BH4%3D&	\n
Expires=1175139620 HTTP/1.1	1175139620
Host: johnsmith.s3.amazonaws.com	/johnsmit

We assume that when a browser makes the GET request, it won't provide a Content-MD5 or a Content-Type header, nor will it set any x-amz- headers, so those parts of the *StringToSign* are left blank.

REST Access Control Policy

Topics

•

Existing Buckets or Objects

<u>Canned Access Policies</u>

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

Existing Buckets or Objects

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

Example

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

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

```
GET /Neo?acl HTTP/1.1
Host: quotes.s3.amazonaws.com
Date: Wed, 01 Mar 2006 12:00:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
HTTP/1.1 200 OK
<AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    </owner>
        <ID>314133b66967d86f031c7249d1d9a80249109428335cd0ef1cdc487b4566cb1t
        <DisplayName>s3-nickname</DisplayName>
        </owner>
        <AccessControlList>
        <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xs:</pre>
```

```
<ID>314133b66967d86f031c7249d1d9a80249109428335cd0ef1cdc487b4566
<DisplayName>s3-nickname</DisplayName>
</Grantee>
<Permission>FULL_CONTROL</Permission>
</Grant>
</AccessControlList>
</AccessControlPolicy>
```

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

<ID>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be(

<DisplayName>chriscustomer</DisplayName>

</Grantee>

<Permission>FULL_CONTROL</Permission>

</Grant>

</AccessControlList>

</AccessControlPolicy>

Canned Access Policies

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

Following are canned ACLs that are supported for REST.

• **private**—Owner gets FULL_CONTROL.

No one else has access rights (default).

• **public-read**—Owner gets FULL_CONTROL and the anonymous principal is granted READ access.

If this policy is used on an object, it can be read from a browser with no authentication.

• **public-read-write**—Owner gets FULL_CONTROL, the anonymous principal is granted READ and WRITE access.

This can be a useful policy to apply to a bucket, but is generally not recommended.

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

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

Sample Request

```
PUT /Neo HTTP/1.1
x-amz-acl: public-read
Content-Length: 4
Host: quotes.s3.amazonaws.com
Date: Wed, 01 Mar 2006 12:00:00 GMT
Content-Type: text/plain
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
woah
```

Sample Response

```
HTTP/1.1 200 OK
x-amz-id-2: LriYPLdmOdAiIfgSm/F1YsViT1LW94/xUQxMsF7xiEbla0wiIOIxl+zbwZ1(
x-amz-request-id: 0A49CE4060975EAC
Date: Wed, 01 Mar 2006 12:00:00 GMT
ETag: "aba878a8"
Content-Length: 0
Connection: close
Server: AmazonS3
```

Virtual Hosting of Buckets

Topics

•

HTTP Host Header Bucket Specification

- Examples
- <u>Customizing Amazon S3 URLs with CNAMEs</u>
- <u>Limitations</u>
- Backward Compatibility

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

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

HTTP Host Header Bucket Specification

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

- If the Host header is omitted or its value is 's3.amazonaws.com', the bucket for the request will be the first slash-delimited component of the Request-URI, and the key for the request will be the rest of the Request-URI. This is the ordinary method, as illustrated by the first and second example in the following table. Note that omitting the Host header is only legal for HTTP 1.0 requests.
- Otherwise, if the value of the Host header ends in '.s3.amazonaws.com', then the bucket name is the leading component of the Host header's value up to '.s3.amazonaws.com'. The key for the request is the Request-URI. This interpretation exposes buckets as sub-domains of s3.amazonaws.com, and is illustrated by the third and fourth example in the following table.
- Otherwise, the bucket for the request will be the lower-cased value of the Host header and the key for the request is the Request-URI. This interpretation is useful when you have registered the same DNS name as your bucket name, and have configured that name to be a CNAME alias for Amazon S3. The procedure for registering domain names and configuring DNS is outside the scope of this document, but the result is illustrated by the final example in the following table.

Examples

This section provides example URLs and requests.

Example Path Style Method

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

Following is the example URL.

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

Following is the example request.

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

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

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

EU bucket names must be DNS compatible and therefore do not support the path style method. Non-EU bucket names do not have to be DNS compatible and therefore can support the path style method. Non-EU buckets can be named, http://s3.amazonaws.com/[bucket-name]/[*key*], for example, http://s3.amazonaws.com/images.johnsmith.net/mydog.jpg.

For more information about DNS compatible names, see <u>Limitations</u>. For more information about keys, see <u>Keys</u>.

Example Virtual Hosted Style Method

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

key name.

Following is the example URL.

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

Following is the example request.

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

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

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

Example CNAME Method

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

Following is the example URL.

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

Following is the example request.

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

Customizing Amazon S3 URLs with CNAMEs

Depending on your needs, you might not want "s3.amazonaws.com" to appear on your web site or service. For example, if you host your web site's images on Amazon S3, you might prefer http://images.johnsmith.net/ as opposed to http://johnsmith-images.s3.amazonaws.com/.

Any bucket with a DNS compatible name may be referenced as follows: http://[BucketName].s3.amazonaws.com/[*Filename*], for example, http://images.johnsmith.net.s3.amazonaws.com/mydog.jpg.Using CNAME you can map images.johnsmith.net to an Amazon S3 host name so the previous URL could become: http://images.johnsmith.net/mydog.jpg.

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

images.johnsmith.net CNAME images.johnsmith.net.s3

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



Note

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

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

1. Select a host name that belongs to a domain you control.

This example uses the images subdomain of the johnsmith.net domain.

2. Create a bucket that matches the host name.

In this example, the host and bucket names are images.johnsmith.net.



Your bucket name must exactly match the host name.

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

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

🚺 Important

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



Note

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

Limitations

Because DNS names are case insensitive, only lower-case buckets are addressable using the virtual hosting method. For more information, see <u>Bucket</u> <u>Restrictions and Limitations</u>.

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

Backward Compatibility

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

Request Redirection and the REST API

Topics

•

Redirects and HTTP User-Agents

- Redirects and 100-Continue
- <u>Redirect Example</u>

This section describes how to handle HTTP redirects using REST. For general information about Amazon S3 redirects, see <u>Request Redirection and the REST</u> <u>API</u>.

Redirects and HTTP User-Agents

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

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

Launch Process

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

HTTP user-agents that strictly conform to RFC2616 might require explicit confirmation before following a redirect when the HTTP request method is not GET or HEAD. It is generally safe to follow redirects generated by Amazon S3 automatically, as the system will only issue redirects to hosts within the amazonaws.com domain and the effect of the redirected request will be the same as the original request.

Redirects and 100-Continue

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



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

Redirect Example

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

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

```
PUT /nelson.txt HTTP/1.1
Host: quotes.s3.amazonaws.com
Date: Mon, 15 Oct 2007 22:18:46 +0000
Content-Length: 6
Expect: 100-continue
```

Amazon S3 returns the following:

```
HTTP/1.1 307 Temporary Redirect
Location: http://quotes.s3-4c25d83b.amazonaws.com/nelson.txt?rk=8d47490k
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Mon, 15 Oct 2007 22:18:46 GMT
Server: AmazonS3
<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>TemporaryRedirect</Code>
  <Message>Please re-send this request to the
  specified temporary endpoint. Continue to use the
  original request endpoint for future requests.
  </Message>
  <Endpoint>quotes.s3-4c25d83b.amazonaws.com</Endpoint>
  <Bucket>quotes</Bucket>
</Error>
```

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

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

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

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

HTTP/1.1 100 Continue

The client sends the request body.

ha ha∖n

Amazon S3 returns the final response.

HTTP/1.1 200 OK Date: Mon, 15 Oct 2007 22:18:48 GMT ETag: "a2c8d6b872054293afd41061e93bc289" Content-Length: 0 Server: AmazonS3

Browser-Based Uploads Using POST

Topics

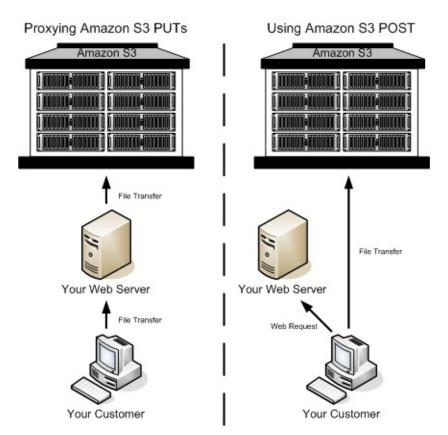
•

HTML Forms

- <u>Upload Examples</u>
- POST with Adobe Flash

Amazon S3 supports POST, which allows your users to upload content directly to Amazon S3. POST is designed to simplify uploads, reduce upload latency, and save you money on applications where users upload data to store in Amazon S3.

The following figure shows an upload using Amazon S3 POST.



Uploading Using POST

1	The user opens a web browser and accesses your web page.
2	Your web page contains an HTTP form that contains all the information necessary for the user to upload content to Amazon S3.
3	The user uploads content directly to Amazon S3.



Query string authentication is not supported for POST.

HTML Forms

Topics

- HTML Form Encoding
- <u>HTML Form Declaration</u>
- HTML Form Fields
- <u>Policy Construction</u>
- <u>Constructing a Signature</u>
- <u>Redirection</u>

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

To allow users to upload content to Amazon S3 using their browsers, you use HTML forms. HTML Forms consist of a form declaration and form fields. The form declaration contains high level information about the request. The form fields contain detailed information about the request as well as the policy that is used to authenticate it and make sure that it meets conditions that you specify.



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

This section describes how to use HTML forms.

HTML Form Encoding

The form and policy must be UTF-8 encoded. You can apply UTF-8 encoding to the form by specifying it in the HTML heading or as a request header.



The HTML form declaration does not accept query string authentication parameters. For information about query string authentication, see <u>Query String</u> <u>Authentication</u>.

Following is an example of UTF-8 encoding in the HTML heading.

```
<html>
<head>
...
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8",
...
</head>
<body>
```

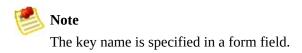
Following is an example of UTF-8 encoding in a request header.



HTML Form Declaration

The form declaration has three components: the action, the method, and the enclosure type. If any of these values is improperly set, the request fails.

The action specifies the URL that processes the request, which must be set to the URL of the bucket. For example, if the name of your bucket is "johnsmith", the URL is "http://johnsmith.s3.amazonaws.com/".



The method must be POST.

The enclosure type (enctype) must be specified and must be set to multipart/form-data (go to <u>RFC 1867</u>) for both file uploads and text area uploads.

Example

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

```
<form action="http://johnsmith.s3.amazonaws.com/" method="post"</pre>
```

```
enctype="multipart/form-data">
```

HTML Form Fields

Following is a table that describes a list of fields that can be used within a form.



The variable \${filename} is automatically replaced with the name of the file provided by the user and is recognized by all form fields. If the browser or client provides a full or partial path to the file, only the text following the last slash (/) or backslash (\) will be used (e.g., "C:\Program Files\directory1\file.txt" will be interpreted as "file.txt"). If no file or filename is provided, the variable is replaced with an empty string.

Element Name	Description	Required
AWSAccessKeyId	The AWS Access Key ID of the owner of the bucket who grants an Anonymous user access for a request that satisfies the set of constraints in the Policy. This is required if a policy document is included with the request.	Conditional
acl	 Specifies an Amazon S3 access control list. Options include private, public-read, public-read-write, authenticated-read. The default setting is private. If an invalid access control list is specified, an error is generated. For more information on ACLs, see <u>Access Control Lists</u>. 	No
Cache-Control, Content- Type, Content- Disposition, Content- Encoding, Expires	REST-specific headers. For more information, see <u>RESTObjectPUT</u> .	No
key	The name of the uploaded key. To use the filename provided by the user, use the \${filename} variable. For example, if the user Betty uploads the file the file lolcatz.jpg and you specify /user/betty/\${filename}, the file is stored as /user/betty/lolcatz.jpg. For more information, see <u>Keys</u> .	Yes
policy	Security Policy describing what is permitted in the request. Requests without a security policy are considered anonymous and only work on publicly writable buckets.	Yes

success_action_redirect,	The LIDI to a high the align tie we diverse drug on an encouncil	N
redirect	The URL to which the client is redirected upon successful upload.	No
	If success_action_redirect is not specified, Amazon S3	
	returns the empty document type specified in the	
	success_action_status field.	
	If Amazon S3 cannot interpret the URL, it acts as if the field is not present.	
	-	
	If the upload fails, Amazon S3 displays an error and does not redirect the user to a URL.	
	For more information, see <u>Redirection</u> .	
	The second se	
	Note Note	
	The redirect field name	
	is deprecated and	
	support for the redirect	
	field name will be	
	removed in the future.	
success_action_status	The status code returned to the client upon successful upload if success_action_redirect is not specified.	No
	Accepts the values 200, 201, or 204 (default).	
	If the value is set to 200 or 204, Amazon S3 returns an empty document with a 200 or 204 status code.	
	If the value is set to 201, Amazon S3 returns an XML	
	document with a 201 status code. For information on the content of the XML document, go to <u>RESTObjectPUT</u> .	
	If the value is not set or if it is set to an invalid value,	
	Amazon S3 returns an empty document with a 204 status code.	
	-	
	📂 Note	
	Some versions of the	
	Adobe Flash player do	
	not properly handle	
	HTTP responses with an	
	empty body. To support	

	uploads through Adobe Flash, we recommend setting <i>success_action_status</i> to 201.	
signature	The HMAC signature constructed using the secret key of the provided AWSAccessKeyId.For more information, see <u>Policy Construction</u> and <u>Authentication and Access Control</u>.	Conditional
x-amz-security-token	 Amazon DevPay security token. Each request that uses Amazon DevPay requires two x-amz-security-token form fields: one for the product token and one for the user token. For more information, see <u>Using Amazon DevPay with Amazon S3</u>. 	No
Other field names prefixed with x-amz-meta-	User-specified metadata. Amazon S3 does not validate or use this data. For more information, see <u>RESTObjectPUT</u> .	No
file	File or text content.The file or content must be the last field in the form.You cannot upload more than one file at a time.	Yes

Policy Construction

Topics

- Expiration
- <u>Conditions</u>
- Condition Matching
- Character Escaping

The policy is a UTF-8 and Base64 encoded JSON document that specifies conditions which the request must meet and is used to authenticate the content. Depending on how you design your policy documents, you can use them perupload, per-user, for all uploads, or according to other designs that meet your needs.



Note

Although the policy document is optional, we highly recommend it over making a bucket publicly writable.

Following is an example of a policy document.

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

The policy document contains the expiration and conditions.

Expiration

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

Conditions

The conditions in the policy document are used to validate the contents of the uploaded object. Each form field that you specify in the form (except AWSAccessKeyId, signature, file, policy, and field names that have an x-ignore- prefix) must be included in the list of conditions.



If you have multiple fields with the same name, the values must be separated by commas. For example, if you have two fields named "x-amz-meta-tag" and the first one has a value of "Ninja" and second has a value of "Stallman", you would set the policy document to Ninja, Stallman.

All variables within the form are expanded prior to validating the policy. Therefore, all condition matching should be against the expanded fields. For example, if you set the key field to user/betty/\${filename}, your policy might be ["starts-with", "\$key", "user/betty/"]. Do not enter ["startswith", "\$key", "user/betty/\${filename}"]. For more information, see <u>Condition Matching</u>.

Policy document conditions are described in the following table.

Element Name	Description
acl	Specifies conditions the ACL must meet.
	Supports exact matching and <i>starts-with</i> .
content-length-range	Specifies the minimum and maximum allowable size for the uploaded content.
	Supports range matching.
Cache-Control,	REST-specific headers.
Content-Type, Content- Disposition, Content- Encoding, Expires	Supports exact matching and <i>starts-with</i> .
key	The name of the uploaded key.
	Supports exact matching and <i>starts-with</i> .

success_action_redirect, redirect	The URL to which the client is redirected upon successful upload. Supports exact matching and <i>starts-with</i> .	
success_action_status	The status code returned to the client upon successful upload if success_action_redirect is not specified.	
	oupporto erace materinig.	
x-amz-security-tokenAmazon DevPay security token.Each request that uses Amazon DevPay requires two x-amz-security- token form fields: one for the product token and one for the user token. result, the values must be separated by commas. For example, if the user token is <i>eW91dHViZQ</i> == and the product token is <i>b0hnNVNKWVJIQTA</i> =, you the policy entry to: { "x-amz-security-token": "eW91dHViZQ==, b0hnNVNKWVJIQTA=" }.		
	For more information about Amazon DevPay, see <u>Using Amazon DevPay</u> <u>with Amazon S3</u> .	
Other field names prefixed with x-amz-	User-specified metadata.	
meta-	Supports exact matching and <i>starts-with</i> .	



If your toolkit adds additional fields (e.g., Flash adds filename), you must add them to the policy document. If you can control this functionality, prefix x-ignore- to the field so Amazon S3 ignores the feature and it won't affect future versions of this feature.

Condition Matching

Following is a table that describes condition matching types. Although you must specify one condition for each form field that you specify in the form, you can create more complex matching criteria by specifying multiple conditions for a form field.

Condition	Description	
Exact Matches	Exact matches verify that fields match specific values. This example indicates that the ACL must be set to public-read:	
	<pre>{"acl": "public-read" }</pre>	
	This example is an alternate way to indicate that the ACL must be set to public-read:	

	["eq", "\$acl", "public-read"]
Starts With	If the value must start with a certain value, use starts-with. This example indicates that the key must start with user/betty:
	["starts-with", "\$key", "user/betty/"]
Matching Any Content	To configure the policy to allow any content within a field, use starts-with with an empty value. This example allows any success_action_redirect:
	["starts-with", "\$success_action_redirect", ""]
Specifying Ranges	For fields that accept ranges, separate the upper and lower ranges with a comma. This example allows a file size from 1 to 10 megabytes:
	["content-length-range", 1048579, 10485760]

Character Escaping

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

Escape Sequence	Description
//	Backslash
\\$	Dollar symbol
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\mathbf{v}	Vertical tab
\uxxxx	All Unicode characters

Constructing a Signature

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

For information about constructing the policy, see <u>Policy Construction</u>. For general information about authentication, see <u>Authentication and Access</u> <u>Control</u>.

Redirection

This section describes how to handle redirects.

General Redirection

On completion of the POST, the user is redirected to the location that you specified in the success_action_redirect field. If Amazon S3 cannot interpret the URL, it ignores the success_action_redirect field.

If success_action_redirect is not specified, Amazon S3 returns the empty document type specified in the success_action_status field.

If the POST fails, Amazon S3 displays an error and does not provide a redirect.

Pre-Upload Redirection

If your bucket was created using <CreateBucketConfiguration>, your end-users might require a redirect. If this occurs, some browsers might handle the redirect incorrectly. This is relatively rare, but is most likely to occur right after a bucket is created.

Upload Examples

Topics

- <u>File Upload</u><u>Text Area Upload</u>

File Upload

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

Policy and Form Construction

Following is a policy that supports uploads to Amazon S3 for the johnsmith bucket.

```
{ "expiration": "2007-12-01T12:00:00.000Z",
  "conditions": [
    {"bucket": "johnsmith"},
    ["starts-with", "$key", "user/eric/"],
    {"acl": "public-read"},
    {"success_action_redirect": "http://johnsmith.s3.amazonaws.com/succe
    ["starts-with", "$Content-Type", "image/"],
    {"x-amz-meta-uuid": "14365123651274"},
    ["starts-with", "$x-amz-meta-tag", ""]
  ]
}
```

This policy requires the following:

- The upload must occur before 12:00 GMT on 2007-12-01
- The content must be uploaded to the johnsmith bucket
- The key must start with "user/eric/"

- The ACL is set to public-read
- The success_action_redirect is set to http://johnsmith.s3.amazonaws.com/successful_upload.html
- The object is an image file
- The x-amz-meta-uuid tag must be set to 14365123651274
- The x-amz-meta-tag can contain any value

Following is a Base64 encoded version of this policy.

eyAiZXhwaXJhdGlvbiI6ICIyMDA3LTEyLTAxVDEy0jAw0jAwLjAwMFoiLAogICJjb25kaXRp

The secret key ID is uV3F3YluFJax1cknvbcGwgjvx4QpvB+leU8dUj2o, so 0RavWzkygo6QX9caELEqKi9kDbU= is the signature for the preceding Policy document.

Following is a form that supports a POST to the johnsmith.net bucket using this policy.

```
<html>
<head>
...
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8",
...
</head>
<body>
...
<form action="http://johnsmith.s3.amazonaws.com/" method="post" enctype",
Key to upload: <input type="input" name="key" value="user/eric/" /><
```

```
<input type="hidden" name="acl" value="public-read" />
<input type="hidden" name="success_action_redirect" value="http://jc
Content-Type: <input type="input" name="Content-Type" value="image/j
<input type="hidden" name="x-amz-meta-uuid" value="14365123651274" /
Tags for File: <input type="input" name="x-amz-meta-tag" value="" />
<input type="hidden" name="AWSAccessKeyId" value="15B4D3461F17762426
<input type="hidden" name="Policy" value="POLICY" />
<input type="hidden" name="Signature" value="SIGNATURE" />
File: <input type="file" name="file" /> <br />
<!-- The elements after this will be ignored -->
<input type="submit" name="submit" value="Upload to Amazon S3" />
</form>
...
</html>
```

Sample Request

This request assumes that the image uploaded is 117,108 bytes; the image data is not included.

```
POST / HTTP/1.1
Host: johnsmith.s3.amazonaws.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.10)
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
```

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Keep-Alive: 300

Connection: keep-alive

Content-Type: multipart/form-data; boundary=9431149156168

Content-Length: 2661134

--9431149156168

Content-Disposition: form-data; name="key"

user/eric/MyPicture.jpg

--9431149156168

Content-Disposition: form-data; name="acl"

public-read

--9431149156168

Content-Disposition: form-data; name="success_action_redirect"

http://johnsmith.s3.amazonaws.com/successful_upload.html

--9431149156168

Content-Disposition: form-data; name="Content-Type"

image/jpeg

--9431149156168

Content-Disposition: form-data; name="x-amz-meta-uuid"

14365123651274

--9431149156168

Content-Disposition: form-data; name="x-amz-meta-tag"

Some,Tag,For,Picture

--9431149156168

Content-Disposition: form-data; name="AWSAccessKeyId"

15B4D3461F177624206A

--9431149156168

Content-Disposition: form-data; name="Policy"

eyAiZXhwaXJhdGlvbiI6ICIyMDA3LTEyLTAxVDEy0jAw0jAwLjAwMFoiLAogICJjb25kaXRr --9431149156168

Content-Disposition: form-data; name="Signature"

0RavWzkygo6QX9caELEqKi9kDbU=

--9431149156168

Content-Disposition: form-data; name="file"; filename="MyFilename.jpg"

Content-Type: image/jpeg

...file content...

--9431149156168

Content-Disposition: form-data; name="submit"

Upload to Amazon S3

--9431149156168--

Sample Response

HTTP/1.1 303 Redirect
x-amz-request-id: 1AEE782442F35865
x-amz-id-2: cxzFLJRatFHy+NGtaDFRR8YvI9BHmgLxjvJzNiGGICARZ/mVXHj7T+qQKhdp
Content-Type: application/xml
Date: Wed, 14 Nov 2007 21:21:33 GMT
Connection: close
Location: http://johnsmith.s3.amazonaws.com/successful_upload.html?bucke
Server: AmazonS3

Text Area Upload

Topics

- Policy and Form Construction
- Sample Request
- <u>Sample Response</u>

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

Policy and Form Construction

Following is a policy that supports text area uploads to Amazon S3 for the johnsmith bucket.

```
{ "expiration": "2007-12-01T12:00:00.000Z",
    "conditions": [
        {"bucket": "johnsmith"},
        ["starts-with", "$key", "user/eric/"],
        {"acl": "public-read"},
        {"acl": "public-read"},
        {"success_action_redirect": "http://johnsmith.s3.amazonaws.com/new_puper
        ["eq", "$Content-Type", "text/html"],
        {"x-amz-meta-uuid": "14365123651274"},
        ["starts-with", "$x-amz-meta-tag", ""]
    ]
}
```

This policy requires the following:

- The upload must occur before 12:00 GMT on 2007-12-01
- The content must be uploaded to the johnsmith bucket
- The key must start with "user/eric/"
- The ACL is set to public-read
- The success_action_redirect is set to http://johnsmith.s3.amazonaws.com/new_post.html
- The object is HTML text
- The x-amz-meta-uuid tag must be set to 14365123651274
- The x-amz-meta-tag can contain any value

Following is a Base64 encoded version of this policy.

eyAiZXhwaXJhdGlvbiI6ICIyMDA3LTEyLTAxVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXR pb25zIjogWwogICAgeyJidWNrZXQiOiAiam9obnNtaXRoIn0sCiAgICBbInN0YXJ0cy13aXF LAogICAgeyJhY2wiOiAicHVibGljLXJlYWQifSwKICAgIHsic3VjY2Vzc19hY3Rpb25fcmVF C5zMy5hbWF6b25hd3MuY29tL25ld19wb3N0Lmh0bWwifSwKICAgIFsiZXEiLCAiJENvbnRlk CAgIHsieC1hbXotbWV0YS11dWlkIjogIjE0MzY1MTIzNjUxMjc0In0sCiAgICBbInN0YXJ0c IsICIiXQogIF0KfQo=

The secret key ID is uV3F3YluFJax1cknvbcGwgjvx4QpvB+leU8dUj2o, so qA7FWXKq6VvU68lI9KdveT1cWgE= is the signature for the preceding Policy document.

Following is a form that supports a POST to the johnsmith.net bucket using this policy.

<html> <head>

```
. . .
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" /
    . . .
  </head>
  <body>
  . . .
  <form action="http://johnsmith.s3.amazonaws.com/" method="post" enctyp</pre>
    Key to upload: <input type="input" name="key" value="user/eric/" /><
    <input type="hidden" name="acl" value="public-read" />
    <input type="hidden" name="success action redirect" value="http://jc
    <input type="hidden" name="Content-Type" value="text/html" />
    <input type="hidden" name="x-amz-meta-uuid" value="14365123651274" ,
    Tags for File: <input type="input" name="x-amz-meta-tag" value="" />
    <input type="hidden" name="AWSAccessKeyId" value="15B4D3461F1776242(</pre>
    <input type="hidden" name="Policy" value="POLICY" />
    <input type="hidden" name="Signature" value="SIGNATURE" />
    Entry: <textarea name="file" cols="60" rows="10">
Your blog post goes here.
    </textarea><br />
    <!-- The elements after this will be ignored -->
    <input type="submit" name="submit" value="Upload to Amazon S3" />
  </form>
  . . .
</html>
```

Sample Request

This request assumes that the image uploaded is 117,108 bytes; the image data is not included.

```
POST / HTTP/1.1
Host: johnsmith.s3.amazonaws.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.10)
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Content-Type: multipart/form-data; boundary=178521717625888
Content-Length: 5299
--178521717625888
Content-Disposition: form-data; name="key"
user/eric/NewEntry.html
--178521717625888
Content-Disposition: form-data; name="acl"
public-read
```

--178521717625888

Content-Disposition: form-data; name="success_action_redirect"

http://johnsmith.s3.amazonaws.com/new_post.html

--178521717625888

Content-Disposition: form-data; name="Content-Type"

text/html

--178521717625888

Content-Disposition: form-data; name="x-amz-meta-uuid"

14365123651274

--178521717625888

Content-Disposition: form-data; name="x-amz-meta-tag"

Interesting Post

--178521717625888

Content-Disposition: form-data; name="AWSAccessKeyId"

15B4D3461F177624206A

--178521717625888

Content-Disposition: form-data; name="Policy"

eyAiZXhwaXJhdGlvbiI6ICIyMDA3LTEyLTAxVDEy0jAw0jAwLjAwMFoiLAogICJjb25kaXR

--178521717625888

```
Content-Disposition: form-data; name="Signature"

qA7FWXKq6VvU68lI9KdveTlcWgE=

--178521717625888

Content-Disposition: form-data; name="file"

...content goes here...

--178521717625888

Content-Disposition: form-data; name="submit"

Upload to Amazon S3

--178521717625888--
```

Sample Response

```
HTTP/1.1 303 Redirect
x-amz-request-id: 1AEE782442F35865
x-amz-id-2: cxzFLJRatFHy+NGtaDFRR8YvI9BHmgLxjvJzNiGGICARZ/mVXHj7T+qQKhdp
Content-Type: application/xml
Date: Wed, 14 Nov 2007 21:21:33 GMT
Connection: close
Location: http://johnsmith.s3.amazonaws.com/new_post.html?bucket=johnsm:
Server: AmazonS3
```

POST with Adobe Flash

This section describes how to use POST with Adobe Flash.

Adobe Flash Player Security

By default, the Adobe Flash Player security model prohibits Adobe Flash Players from making network connections to servers outside the domain that serves the SWF file.

To override the default, you must upload a public-readable crossdomain.xml file to the bucket that will accept POST uploads. Following is a sample crossdomain.xml file.

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM
"http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy.dtd">
<cross-domain-policy>
<allow-access-from domain="*" secure="false" />
</cross-domain-policy>
```



Note

For more information about the Adobe Flash security model, go to the <u>Adobe</u> <u>web site</u>.

Adding the crossdomain.xml file to your bucket allows any Adobe Flash Player to connect to the crossdomain.xml file within your bucket. However, it does not grant access to the actual Amazon S3 bucket.

Other Adobe Flash Considerations

The FileReference API in Adobe Flash adds the *Filename* form field to the POST request. When building Adobe Flash applications that upload to Amazon S3 using the FileReference API, include the following condition in your policy:

```
['starts-with', '$Filename', '']
```

Some versions of the Adobe Flash Player do not properly handle HTTP responses that have an empty body. To configure POST to return a response that does not have an empty body, set *success_action_status* to 201. When set, Amazon S3 returns an XML document with a 201 status code. For more information, go to <u>RESTObjectPUT</u>.

Using the SOAP API

Topics

•

Common SOAP API Elements

- Authenticating SOAP Requests
- Setting Access Policy with SOAP

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

Common SOAP API Elements

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

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

Common Elements

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

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

For information about endpoints, see <u>Request Endpoints</u>.

Authenticating SOAP Requests

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

- AWSAccessKeyId: Your AWS Access Key ID
- *Timestamp:* This must be a dateTime (go to http://www.w3.org/TR/xmlschema-2/#dateTime) in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as 2006-01-01T12:00:00.000Z. Authorization will fail if this timestamp is more than 15 minutes away from the clock on Amazon S3 servers.
- *Signature*: The RFC 2104 HMAC-SHA1 digest (go to <u>http://www.ietf.org/rfc/rfc2104.txt</u>) 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 "AmazonS3CreateBucket2006-01-01T12:00:00.000Z":

For example, in the following CreateBucket sample request, the signature element would contain the HMAC-SHA1 digest of the value "AmazonS3CreateBucket2006-01-01T12:00:00.000Z":

Example

```
<CreateBucket xmlns="http://doc.s3.amazonaws.com/2006-03-01">
<Bucket>quotes</Bucket>
<Acl>private</Acl>
<AWSAccessKeyId>1D9FVRAYCP1VJEXAMPLE=</AWSAccessKeyId>
<Timestamp>2006-01-01T12:00:00.000Z</Timestamp>
<Signature>Iuyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
</CreateBucket>
```



Note

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



🚺 Important

Due to different interpretations regarding how extra time precision should be dropped, .NET users should take care not to send Amazon S3 overly specific time stamps. This can be accomplished by manually constructing DateTime objects with only millisecond precision.

For more information, see the sample <u>.NET SOAP libraries</u> 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 <u>Authentication and Access Control</u>. If no access control list is specified with these operations, the resource is created with a default access policy that gives the requester FULL_CONTROL access (this is the case even if the request is a PutObjectInline or PutObject request for an object that already exists).

Following is a request that writes data to an object, makes the object readable by anonymous principals, and gives the specified user FULL_CONTROL rights to the bucket (Most developers will want to give themselves FULL_CONTROL access to their own bucket).

Example

Following is a request that writes data to an object and makes the object readable by anonymous principals.

Sample Request

```
<PutObjectInline xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <Bucket>guotes</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>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be(
        <DisplayName>chriscustomer</DisplayName>
      </Grantee>
      <Permission>FULL CONTROL</Permission>
    </Grant>
    <Grant>
      <Grantee xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/global/AllUsers<URI>
```

```
</Grantee>

<Permission>READ</Permission>

</Grant>

</AccessControlList>

<AWSAccessKeyId>1D9FVRAYCP1VJEXAMPLE=</AWSAccessKeyId>

<Timestamp>2006-03-01T12:00:00.183Z</Timestamp>

<Signature>Iuyz3d3P0aTou39dzbqaEXAMPLE=</Signature>

</Put0bjectInline>
```

Sample Response

```
<PutObjectInlineResponse xmlns="http://s3.amazonaws.com/doc/2006-03-01">
    <PutObjectInlineResponse>
        <ETag>&quot828ef3fdfa96f00ad9f27c383fc9ac7f&quot</ETag>
        <LastModified>2006-01-01T12:00:00.000Z</LastModified>
        </PutObjectInlineResponse>
</PutObjectInlineResponse>
```

The access control policy can be read or set for an existing bucket or object using the GetBucketAccessControlPolicy, GetObjectAccessControlPolicy, SetBucketAccessControlPolicy, and SetObjectAccessControlPolicy methods. For more information, see the detailed explanation of these methods.

Working with Amazon S3 Components

Topics

- Working with Amazon S3 Buckets
- <u>Working with Amazon S3 Objects</u>

This section describes buckets and objects:

Note The <u>Authentication and Access Control</u> section describes access control in detail.

Working with Amazon S3 Buckets

Topics

•

Bucket Restrictions and Limitations

- Bucket Configuration Options
- Requester Pays Buckets
- Buckets and Access Control
- Billing and Reporting of Buckets
- Bucket Configuration Errors

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

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

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

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

that you own the bucket or have permission to access it.

Bucket Configuration Errors

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

- <u>MalformedXML</u>
- <u>MissingRequestBodyError</u>

Bucket Restrictions and Limitations

A bucket is owned by the AWS account (identified by AWS Access Key ID) that created it. Each AWS account can own up to 100 buckets at a time. Bucket ownership is not transferable. However, if a bucket is empty, it can be deleted and its name can be reused.



If you are using Amazon DevPay, each of your customers can have up to 100 buckets for each Amazon DevPay product they use. For more information, see <u>Using Amazon DevPay with Amazon S3</u>.

To comply with Amazon S3 requirements, bucket names:

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

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

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



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

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

If you create a bucket using <CreateBucketConfiguration>, applications that access your bucket must be able to handle 307 redirects. For more information, see <u>Request Redirection and the REST API</u>.

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

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

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

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



Note

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

Bucket Configuration Options

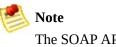
When creating buckets, you can take advantage of additional Amazon S3 features by attaching the <CreateBucketConfiguration> XML body to a PUT Bucket request. Currently, you can select a location constraint. For more information, see <u>How to Select a Region for Your Buckets</u>.

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

- You cannot make a request to a bucket created with <CreateBucketConfiguration> using a path-style request; you must use the virtual hosted-style request. For more information, see <u>Virtual Hosting of Buckets</u>.
- You must follow additional bucket naming restrictions. For more information, see <u>Bucket Restrictions and Limitations</u>.

How to Select a Region for Your Buckets

You can choose a geographical region where Amazon S3 will store the buckets you create. For example, if you reside in Europe, you might like to specify that the buckets you create (and therefore the objects stored in them) reside in Europe. You specify a region using the *LocationConstraint* bucket parameter. If you do not specify a region, Amazon S3 hosts your buckets on servers in the US region. The other region you can constrain a bucket to is EU (Europe).



The SOAP API does not support geographical constraints.

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

Specifying a Bucket's Region

1 In a bucket creation request, set the *LocationContraint* parameter to a specific region, for example, *CreateBucketConfiguration.LocationConstraint=EU*.

Bucket Access

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

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

You cannot use the path-style request:

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

If you use the path-style request, you receive a permanent redirect.

Redirection

Amazon supports two types of redirects: temporary and permanent.

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

Permanent redirects redirect users from the path-style request to the virtual hosted-style request format for buckets created using <CreateBucketConfiguration>. Users will be provided with the correct URL, but will not be forwarded to the correct location.

Requester Pays Buckets

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

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



Important

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

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

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

Requester Pays buckets do not support the following.

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

bucket.

Setting the requestPayment Bucket Configuration

The bucket owner and only the bucket owner can set the *RequestPaymentConfiguration.payer* configuration value of a bucket to BucketOwner, the default, or Requester. Setting the *requestPayment* resource is optional. If you don't, the bucket, by default, is a non-Requester Pays bucket.

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

To set requestPayment

• Use a PUT request to set the *Payer* value to Requester on a specified bucket.

PUT ?requestPayment HTTP/1.1 Host: [BucketName].s3.amazonaws.com Content-Length: 173 Date: Wed, 01 Mar 2009 12:00:00 GMT Authorization: AWS [signature] <RequestPaymentConfiguration xmlns="http://s3.amazonaws.com/doc/200 <Payer>Requester</Payer> </RequestPaymentConfiguration>

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

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

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

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



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

Retrieving requestPayment Configuration

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

To return the requestPayment resource

• Use a GET request to obtain the *requestPayment* resource, as shown in the following request.

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

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

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

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

Downloading Objects in Requester Pays Buckets

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

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

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



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

Make sure to include *x*-*amz*-*request*-*payer* and its value in your signature calculation. For more information, see <u>Constructing the</u> <u>CanonicalizedAmzHeaders Element</u>.

To download objects from a Requester Pays bucket

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

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

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

Amazon S3 can return Access Denied errors for requests trying to get objects from Requester Pays buckets. For more information, go to <u>ErrorCodeList</u>.

DevPay and Requester Pays

You can use Amazon DevPay to sell the content stored in your Requester Pays bucket. For more information, go to "Using Amazon S3 Requester Pays with DevPay," in the <u>Using Amazon S3 Requester Pays with DevPay</u>.

Charge Details

The charge for successful Requester Pays requests is straight forward: the requester pays for data transfer and the request; the bucket owner pays for the data storage. However, the bucket owner is charged for the request if:

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

Buckets and Access Control

Each bucket has an associated access control policy. This policy governs the creation, deletion and enumeration of objects within the bucket. For more information, see <u>Authentication and Access Control</u>.

Billing and Reporting of Buckets

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

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

Working with Amazon S3 Objects

Topics

•

<u>Keys</u>

- <u>Metadata</u>
- <u>Getting Objects</u>
- <u>Copying Amazon S3 Objects</u>

Amazon S3 is designed to store objects. Objects are stored in buckets and consist of a value, a key, metadata, and an access control policy.

The object value is the content that you are storing. The object value can be any sequence of bytes, but must be smaller than five gigabytes. There is no fixed limit to the number of objects you can store in Amazon S3.

The key is the handle that you assign to an object that allows you retrieve it later.

Metadata is a set of key-value pairs with which you can store information regarding the object.

The access control policy controls access to the object.

Keys

Topics

•

Listing Keys

- <u>Common List Request Parameters</u>
- <u>Common List Response Elements</u>
- Iterating Through Multi-Page Results
- Listing Keys Hierarchically using Prefix and Delimiter

The key is the handle that you assign to an object that allows you retrieve it later. A key is a sequence of Unicode characters whose UTF-8 encoding is at most 1024 bytes long. Each object in a bucket must have a unique key.

Keys can be listed by prefix. By choosing a common prefix for the names of related keys and marking these keys with a special character that delimits hierarchy, you can use the list operation to select and browse keys hierarchically. This is similar to how files are stored in directories within a file system. For more information, see Listing Keys.

Keys often have a suffix that describes the type of data in the object. For example, ".jpg" indicates that an object is an image. Although Amazon S3 supports key suffixes, they are not required.

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.

For API independent information about composing a list request, see <u>Common</u> <u>List Request Parameters</u>.

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. For more information, see <u>Common List Response Elements</u>.

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.

For instructions on how to correctly handle large list result sets, see <u>Iterating</u> <u>Through Multi-Page Results</u>.

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 file system. For example, to extend the dictionary bucket to contain more than just English words, you might form keys by prefixing each word with its language and a delimiter, 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 toplevel list of available languages without having to iterate through all the lexicographically intervening keys.

For more information on this aspect of listing, see <u>Listing Keys Hierarchically</u> <u>using Prefix and Delimiter</u>.

List Implementation Efficiency

List performance is not substantially affected by the total number of keys in your bucket, nor by the presence or absence of the prefix, marker, maxkeys, or delimiter arguments.

Common List Request Parameters

Following is a table that describes common list request parameters that are used by both SOAP and REST.

Parameter	Description
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. For more information, see the NextMarker response element. 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. For more information, see these API-specific sections:

- <u>SOAPListBucket</u>
- <u>RESTBucketGET</u>

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.

Example

```
<?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>2006-01-01T12:00:00.000Z</LastModified>
    <ETag>"celacdafcc879d7eee54cf4e97334078"</ETag>
    <Size>1234</Size>
    <0wner>
      <ID>214153b66967d86f031c7249d1d9a80249109428335cd08f1cdc487b4566ct
      <DisplayName>John Smith</DisplayName>
    </0wner>
    <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:

Element	Description
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.

Element	Description
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. For more information, see <u>Access Control</u> .
StorageClass	Always has the value STANDARD.

The CommonPrefixes element might 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 Prefix request parameter.

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. Following is a pseudo-code procedure that 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.Prefixe
        // 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 slash (/), that doesn't occur in any of your anticipated key names. Next, construct your key names by concatenating all containing levels of the hierarchy, separating each level with the delimiter.

For example, if you were storing information about cities, you might naturally organize them by continent, then by country, then by province or state. Since these names don't usually contain punctuation, you might select slash (/) as the delimiter. The following example uses a slash (/) delimiter.

- Europe/France/Aquitaine/Bordeaux
- North America/Canada/Quebec/Montreal
- North America/USA/California/San Francisco
- North America/USA/Washington/Seattle

and so on.

If you stored data for every city in the world in this manner, it would become awkward to manage a flat key namespace. 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.

Metadata

Topics

•

<u>Metadata Size</u>

• Metadata Interoperability

Each Amazon S3 object has a set of key-value pairs with which it is associated. There are two kinds of metadata: system metadata, and user metadata.

System metadata is used and is sometimes processed by Amazon S3. System metadata behavior depends on which API (REST or SOAP) you are using.

User metadata entries are specified by you. Amazon S3 does not interpret this metadata—it simply stores it and passes it back when you ask for it. Metadata keys and values can be any length, but must conform to US-ASCII when using REST and UTF-8 when using SOAP or browser-based uploads through POST.



For more information about metadata encodings, go to sections 2 and 4.2 of <u>http://www.ietf.org/rfc/rfc2616.txt</u>.

Metadata Size

For both REST and SOAP requests to Amazon S3, user metadata size is limited to 2k bytes for the total length of all values and keys.

Metadata Interoperability

In REST, user metadata keys must begin with "x-amz-meta-" to distinguish them as custom HTTP headers. When this metadata is retrieved via SOAP, the x-amzmeta- prefix is removed. Similarly, metadata stored via SOAP will have x-amzmeta- added as a prefix when it is retrieved via REST or HTTP, except the Content-Type header.

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

Getting Objects

Topics

•

Standard Downloads

• Chunked and Resumable Downloads

You get objects from Amazon S3 using the GET operation. This operation returns the object directly from Amazon S3.

Standard Downloads

Following is an example of a REST GET request.

```
GET /Nelson HTTP/1.1
Host: quotes.s3.amazonaws.com
Date: Wed, 01 Mar 2006 12:00:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

It returns the following response.

```
HTTP/1.1 200 OK
x-amz-id-2: j5ULAWpFbJQJpukUsZ4tfXV0jVZExLtEyNTvY5feC+hHIegsN5p578JLTVpF
x-amz-request-id: BE39A20848A0D52B
Date: Wed, 01 Mar 2006 12:00:00 GMT
x-amz-meta-family: Muntz
Last-Modified: Sun, 1 Jan 2006 12:00:00 GMT
ETag: "828ef3fdfa96f00ad9f27c383fc9ac7f"
Content-Type: text/plain
Content-Length: 5
Connection: close
Server: AmazonS3
HA-HA
```

Chunked and Resumable Downloads

To provide GET flexibility, Amazon S3 supports chunked and resumable downloads.

This allows you to download part of an object stored in Amazon S3 so you can break large downloads into smaller chunks or design your applications to recover from failed downloads.

You can select a method from the following:

- For information about using resumable downloads with the REST API, go to <u>RESTObjectGET</u>.
- For information about using resumable downloads with the SOAP API, go to <u>SOAPResumableDownloads</u>.

Copying Amazon S3 Objects

The copy operation enables you to copy objects within Amazon S3. Using the copy operation, you can:

- Create additional copies of objects
- Rename objects by copying them and deleting the original ones
- Move objects across Amazon S3 locations (e.g., US and EU)
- Update object metadata by copying original objects to new ones that contain new metadata



Copying objects across locations incurs bandwidth charges.

For more information about copy requests, see <u>RESTObjectCOPY</u> for REST and <u>SOAPCopyObject</u> for SOAP.

Example

This example describes how to copy an object using REST.

This example copies the flotsam object from the pacific bucket to the jetsam object of the atlantic bucket, preserving its metadata.

```
PUT /jetsam HTTP/1.1
Host: atlantic.s3.amazonaws.com
x-amz-copy-source: /pacific/flotsam
Authorization: AWS 15B4D3461F177624206A:ENoSbxYByFA0UGLZUqJN5EUnLDg=
Date: Wed, 20 Feb 2008 22:12:21 +0000
```

The signature was generated from the following information.

```
PUT\r\n
\r\n
\r\n
Wed, 20 Feb 2008 22:12:21 +0000\r\n
x-amz-copy-source:/pacific/flotsam\r\n
/atlantic/jetsam
```

Amazon S3 returns the following response which specifies the etag of the object and when it was last modified.

Related Resources

Access Control Lists

Authentication and Access Control

Topics

•

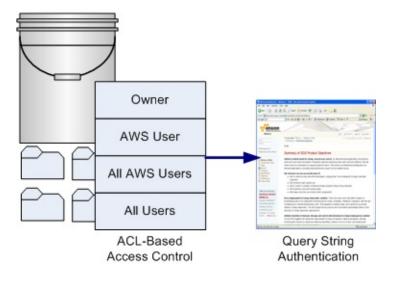
Authentication

- <u>Access Control Lists</u>
- **Query String Authentication**

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

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

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



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

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



Note

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

Authentication

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

- Access Key ID (a 20-character, alphanumeric string). For example: 022QF06E7MXBSH9DHM02
- Secret Access Key (a 40-character string). For example: kWcrlUX5JEDGM/LtmEENI/aVmYvHNif5zB+d9+ct



Important

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

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

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

Viewing Your Credentials

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

To view your AWS access identifiers

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

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

Using HMAC-SHA1 Signatures

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

Request Elements

- AWS Access Key Id—Your AWS account is identified by your Access Key ID, which AWS uses to look up your Secret Access Key.
- **Signature**—Each request must contain a valid request signature, or the request is rejected.

A request signature is calculated using your Secret Access Key, which is a shared secret known only to you and AWS.

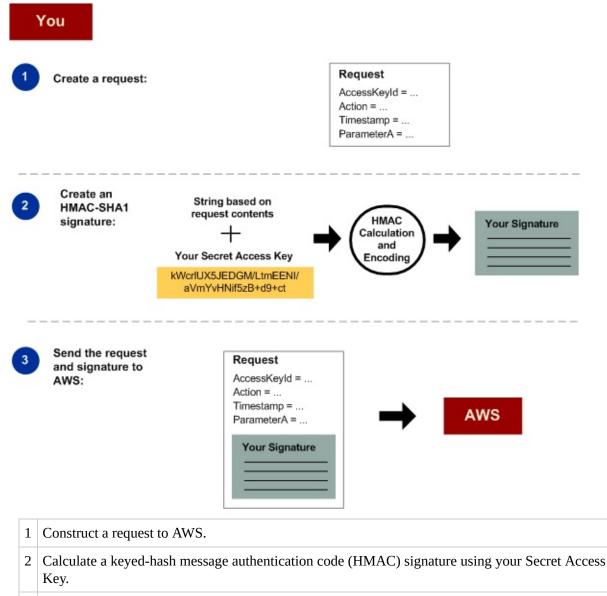
• **Time stamp**—Each request must contain the date and time the request was created, represented as a string in UTC.

The format of the value of this parameter is API-specific.

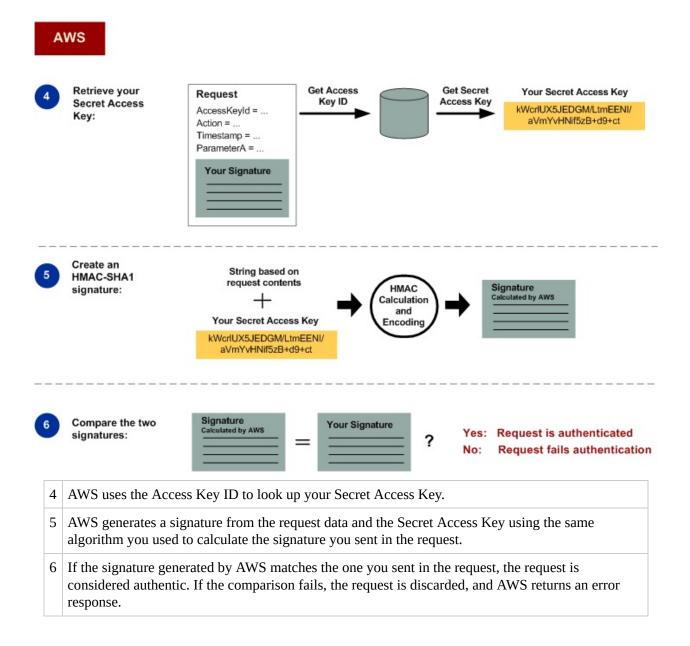
• **Date**—Each request must contain the time stamp of the request.

Depending on the API you're using, you can provide an expiration date and time for the request instead of or in addition to the time stamp. See the authentication topic for the particular API to determine what the API requires.

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



³ Include the signature and your Access Key ID in the request, and then send the request to AWS. AWS performs the next three steps.



Detailed Authentication Information

For detailed information about REST and SOAP authentication, see <u>Authenticating REST Requests</u> and <u>Authenticating SOAP Requests</u>.

Using Base64 Encoding

HMAC request signatures must be Base64 encoded. Base64 encoding converts the signature into a simple ASCII string that can be attached to the request. Two characters, plus (+) and forward slash (/), cannot be used directly and must be encoded if used in a URI. For example, if the authentication code includes a plus (+) sign, encode it as %2B; in the request. Encode a forward slash as %2F;.

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

Access Control Lists

Topics

- <u>Grantees</u>
- <u>Permissions</u>
- Using ACLs

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

An ACL is a list of grants. A grant consists of one grantee and one permission. ACLs only grant permissions; they do not deny them.



Note

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



Important

We highly recommend that you do not grant the anonymous group write access to your buckets as you will have no control over the objects others can store and their associated charges. For more information, see <u>Grantees</u> and <u>Permissions</u>

Grantees

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

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

Owner

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



Note

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

User by E-mail

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

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

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

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

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

User by Canonical Representation

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



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

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

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

The ID string specifies the CanonicalUser ID and must exactly match the ID of

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

AWS User Group

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

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

AllUsers Group

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

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

```
<Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type=
<<URI>http://acs.amazonaws.com/groups/global/AllUsers<URI>
</Grantee>
```

Permissions

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

Elements

• **READ**—when applied to a bucket, grants permission to list the bucket.

When applied to an object, this grants permission to read the object data and/or metadata.

• **WRITE**—when applied to a bucket, grants permission to create, overwrite, and delete any object in the bucket.

This permission is not supported for objects.

• **READ_ACP**—grants permission to read the ACL for the applicable bucket or object.

The owner of a bucket or object always has this permission implicitly.

• **WRITE_ACP**—gives permission to overwrite the ACP for the applicable bucket or object.

The owner of a bucket or object always has this permission implicitly.

Granting this permission is equivalent to granting FULL_CONTROL because the grant recipient can make any changes to the ACP.

• **FULL_CONTROL**—provides READ, WRITE, READ_ACP, and WRITE_ACP permissions.

It does not convey additional rights and is provided only for convenience.

Using ACLs

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

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



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

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

```
<AccessControlPolicy>
<0wner>
<ID>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666e9
<DisplayName>chriscustomer</DisplayName>
</0wner>
<AccessControlList>
<Grant>
<Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xs:
<ID>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be9
<DisplayName>chriscustomer</DisplayName>
</Grantee>
<Permission>FULL_CONTROL</Permission>
</Grant>
<Grant>
<Grant>
<Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xs:
```

```
<ID>79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47€
        <DisplayName>Frank</DisplayName>
      </Grantee>
      <Permission>WRITE</Permission>
    </Grant>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xs:</pre>
        <ID>79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47<
        <DisplayName>Frank</DisplayName>
      </Grantee>
      <Permission>READ ACP</Permission>
    </Grant>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xs:</pre>
        <ID>e019164ebb0724ff67188e243eae9ccbebdde523717cc312255d9a82498(
        <DisplayName>Jose</DisplayName>
      </Grantee>
      <Permission>WRITE</Permission>
    </Grant>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xs:</pre>
         <ID>e019164ebb0724ff67188e243eae9ccbebdde523717cc312255d9a8249{
        <DisplayName>Jose</DisplayName>
    </Grantee>
       <Permission>READ ACP</Permission>
    </Grant>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xs:</pre>
        <URI>http://acs.amazonaws.com/groups/global/AllUsers</URI>
      </Grantee>
      <Permission>READ</Permission>
    </Grant>
 </AccessControlList>
</AccessControlPolicy>
```



Note

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

Query String Authentication

Query string authentication is useful for giving HTTP or browser access to resources that would normally require authentication.

When using query string authentication, you create a query, specify an expiration time for the query, sign it with your signature, place the data in an HTTP request, and distribute the request to a user or embed the request in a web page.

Query string authentication requests require an expiration date. You can specify any future expiration time in epoch or UNIX time (number of seconds since January 1, 1970). For example, a query URL is similar to the following example.

http://quotes.s3.amazonaws.com/nelson?AWSAccessKeyId=44CF9590006BF252F70

Request Routing

Topics

•

Request Redirection and the REST API

DNS Considerations

Programs that make requests against buckets created using the <CreateBucketConfiguration> API must support redirects. Additionally, some clients that do not respect DNS TTLs might encounter issues.

This section describes routing and DNS issues to consider when designing your service or application for use with Amazon S3.

Request Redirection and the REST API

Overview

Amazon S3 uses the Domain Name System (DNS) to route requests to facilities that can process them. This system works very effectively. However, temporary routing errors can occur.

If a request arrives at the wrong Amazon S3 location, Amazon S3 responds with a temporary redirect that tells the requester to resend the request to a new endpoint.

If a request is incorrectly formed, Amazon S3 uses permanent redirects to provide direction on how to perform the request correctly.



Important

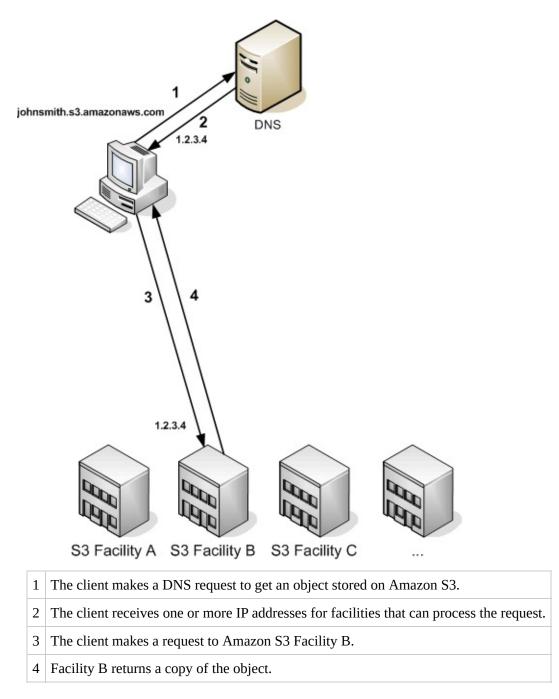
Every Amazon S3 program must be designed to handle redirect responses. The only exception is for programs that work exclusively with buckets that were created without <CreateBucketConfiguration>. For more information on location constraints, see

How to Select a Region for Your Buckets.

DNS Routing

DNS routing routes requests to appropriate Amazon S3 facilities.

The following figure shows an example of DNS routing.



Temporary Request Redirection

A temporary redirect is a type of error response that signals to the requester that he should resend his request to a different endpoint.

Due to the distributed nature of Amazon S3, requests can be temporarily routed to the wrong facility. This is most likely to occur immediately after buckets are created or deleted. For example, if you create a new bucket and immediately make a request to the bucket, you will receive a temporary redirect. After information about the bucket propagates through DNS, redirects will be rare.

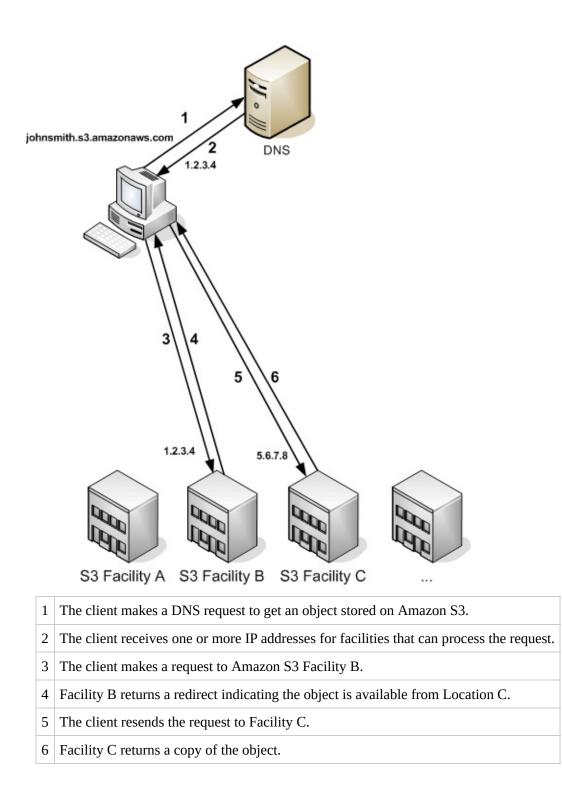
Temporary redirects contain a URI to the correct facility which you can use to immediately resend the request.



Important

Do not reuse an endpoint provided by a previous redirect response. It might appear to work (even for long periods of time), but might provide unpredictable results and will eventually fail without notice.

The following figure shows an example of a temporary redirect.



Permanent Request Redirection

A permanent redirect indicates that your request addressed a resource inappropriately. For example, permanent redirects occur if you use a path-style request to access a bucket that was created using <CreateBucketConfiguration>. For more information, see <u>Using</u> <u>CreateBucketConfiguration</u>.

To help you find these errors during development, this type of redirect does not contain a Location HTTP header that allows you to automatically follow the request to the correct location. Consult the resulting XML error document for help using the correct Amazon S3 endpoint.

Example REST API Redirect

```
HTTP/1.1 307 Temporary Redirect
Location: http://johnsmith.s3-gztb4pa9sq.amazonaws.com/photos/puppy.jpg?
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Fri, 12 Oct 2007 01:12:56 GMT
Server: AmazonS3
<?xml version="1.0" encoding="UTF-8"?>
<Error>
        <Code>TemporaryRedirect</Code>
        <Message>Please re-send this request to the specified temporary endpo:
        Continue to use the original request endpoint for future requests.</me
        <Endpoint>johnsmith.s3-gztb4pa9sq.amazonaws.com</Endpoint>
</Error>
```

Example SOAP API Redirect

</soapenv:Fault> </soapenv:Body>

DNS Considerations

One of the design requirements of Amazon S3 is extremely high availability. One of the ways we meet this requirement is by updating the IP addresses associated with the Amazon S3 endpoint in DNS as needed. These changes are automatically reflected in short-lived clients, but not in some long-lived clients. Long-lived clients will need to take special action to re-resolve the Amazon S3 endpoint periodically to benefit from these changes. For more information about virtual machines (VMs). refer to the following:

- For Java, Sun's JVM caches DNS lookups forever by default; go to the "InetAddress Caching" section of <u>the InetAddress documentation</u> for information on how to change this behavior.
- For PHP, the persistent PHP VM that runs in the most popular deployment configurations caches DNS lookups until the VM is restarted. Go to <u>the getHostByName PHP docs</u>.

Performance Optimization

Topics

•

TCP Window Scaling

• **TCP Selective Acknowledgement**

Amazon S3 provides new features that support high performance networking. These include TCP window scaling and selective acknowledgements.

Note

For more information on high performance tuning, go to <u>http://www.psc.edu/networking/projects/tcptune/</u>.

TCP Window Scaling

TCP window scaling allows you to improve network throughput performance between your operating system and application layer and Amazon S3 by supporting window sizes larger than 64 KB. At the start of the TCP session, a client advertises its supported receive window WSCALE factor, and Amazon S3 responds with its supported receive window WSCALE factor for the upstream direction.

Although TCP window scaling can improve performance, it can be challenging to set correctly. Make sure to adjust settings at both the application and kernel level. For more information about TCP window scaling, refer to your operating system's documentation and go to <u>RFC 1323</u>.

TCP Selective Acknowledgement

TCP selective acknowledgement is designed to increase recovery time after a large number of packet losses. TCP selective acknowledgement is supported by most newer operating systems, but might have to be enabled. For more information about TCP selective acknowledgements, refer to the documentation that accompanied your operating system and go to <u>RFC 2018</u>.

Using BitTorrent with Amazon S3

Topics

•

How You are Charged for BitTorrent Delivery

- Using BitTorrent to Retrieve Objects Stored in Amazon S3
- Publishing Content Using Amazon S3 and BitTorrent

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

Amazon S3 supports the BitTorrent protocol so that developers can save costs when distributing content at high scale. Amazon S3 is useful for simple, reliable storage of any data. The default distribution mechanism for Amazon S3 data is via client/server download. In client/server distribution, the entire object is transferred point-to-point from Amazon S3 to every authorized user who requests that object. While client/server delivery is appropriate for a wide variety of use cases, it is not optimal for everybody. Specifically, the costs of client/server distribution increase linearly as the number of users downloading objects increases. This can make it expensive to distribute popular objects.

BitTorrent addresses this problem by recruiting the very clients that are downloading the object as distributors themselves: Each client downloads some pieces of the object from Amazon S3 and some from other clients, while simultaneously uploading pieces of the same object to other interested "peers." The benefit for publishers is that for large, popular files the amount of data actually supplied by Amazon S3 can be substantially lower than what it would have been serving the same clients via client/server download. Less data transferred means lower costs for the publisher of the object.

How You are Charged for BitTorrent Delivery

There is no extra charge for use of BitTorrent with Amazon S3. Data transfer via the BitTorrent protocol is metered at the same rate as client/server delivery. To be precise, whenever a downloading BitTorrent client requests a "piece" of an object from the Amazon S3 "seeder," charges accrue just as if an anonymous request for that piece had been made using the REST or SOAP protocol. These charges will appear on your Amazon S3 bill and usage reports in the same way. The difference is that if a lot of clients are requesting the same object simultaneously via BitTorrent, then the amount of data Amazon S3 must serve to satisfy those clients will be lower than with client/server delivery. This is because the BitTorrent clients are simultaneously uploading and downloading amongst themselves.

The data transfer savings achieved from use of BitTorrent can vary widely depending on how popular your object is. Less popular objects require heavier use of the "seeder" to serve clients, and thus the difference between BitTorrent distribution costs and client/server distribution costs might be small for such objects. In particular, if only one client is ever downloading a particular object at a time, the cost of BitTorrent delivery will be the same as direct download.

Using BitTorrent to Retrieve Objects Stored in Amazon S3

Any object in Amazon S3 that can be read anonymously can also be downloaded via BitTorrent. Doing so requires use of a BitTorrent client application. Amazon does not distribute a BitTorrent client application, but there are many free clients available. The Amazon S3BitTorrent implementation has been tested to work with the official BitTorrent client (go to http://www.bittorrent.com/).

The starting point for a BitTorrent download is a .torrent file. This small file describes for BitTorrent clients both the data to be downloaded and where to get started finding that data. A .torrent file is a small fraction of the size of the actual object to be downloaded. Once you feed your BitTorrent client application an Amazon S3 generated .torrent file, it should start downloading immediately from Amazon S3 *and* from any "peer" BitTorrent clients.

Retrieving a .torrent file for any publicly available object is easy. Simply add a "?torrent" query string parameter at the end of the REST GET request for the object. No authentication is required. Once you have a BitTorrent client installed, downloading an object using BitTorrent download might be as easy as opening this URL in your web browser.

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

Example

This example retrieves the Torrent file for the "Nelson" object in the "quotes" bucket.

Sample Request

```
GET /quotes/Nelson?torrent HTTP/1.0
Date: Wed, 01 Mar 2006 12:00:00 GMT
```

Sample Response

HTTP/1.1 200 OK x-amz-request-id: 7CD745EBB7AB5ED9 Date: Wed, 01 Mar 2006 12:00:00 GMT Content-Disposition: attachment; filename=Nelson.torrent; Content-Type: application/x-bittorrent Content-Length: 537 Server: AmazonS3

<body: a Bencoded dictionary as defined by the BitTorrent specification>

Publishing Content Using Amazon S3 and BitTorrent

Every anonymously readable object stored in Amazon S3 is automatically available for download using BitTorrent. The process for changing the ACL on an object to allow anonymous READ operations is described in <u>Authentication and Access Control</u>.

You can direct your clients to your BitTorrent accessible objects by giving them the .torrent file directly or by publishing a link to the ?torrent URL of your object. One important thing to note is that the .torrent file describing an Amazon S3 object is generated on-demand, the first time it is requested (via the REST ?torrent resource). Generating the .torrent for an object takes time proportional to the size of that object. For large objects, this time can be significant. Therefore, before publishing a ?torrent link, we suggest making the first request for it yourself. Amazon S3 might take several minutes to respond to this first request, as it generates the .torrent file. Unless you update the object in question, subsequent requests for the .torrent will be fast. Following this procedure before distributing a ?torrent link will ensure a smooth BitTorrent downloading experience for your customers.

To stop distributing a file using BitTorrent, simply remove anonymous access to it. This can be accomplished by either deleting the file from Amazon S3, or modifying your access control policy to prohibit anonymous reads. After doing so, Amazon S3 will no longer act as a "seeder" in the BitTorrent network for your file, and will no longer serve the .torrent file via the ?torrent REST API. However, after a .torrent for your file is published, this action might not stop public downloads of your object that happen exclusively using the BitTorrent peer to peer network.

Using Amazon DevPay with Amazon S3

Topics

•

Amazon S3 Customer Data Isolation

- <u>Amazon DevPay Token Mechanism</u>
- <u>Amazon S3 and Amazon DevPay Authentication</u>
- <u>Amazon S3 Bucket Limitation</u>
- <u>Amazon S3 and Amazon DevPay Process</u>
- Additional Information

Amazon DevPay enables you to charge customers for using your Amazon S3 product through Amazon's authentication and billing infrastructure. You can charge any amount for your product including usage charges (storage, transactions, and bandwidth), monthly fixed charges, and a one-time charge.

Once a month, Amazon bills your customers for you. AWS then deducts the fixed Amazon DevPay transaction fee and pays you the difference. AWS then separately charges you for the Amazon S3 usage costs incurred by your customers and the percentage-based Amazon DevPay fee.

If your customers do not pay their bills, AWS turns off access to Amazon S3 (and your product). AWS handles all payment processing.

Amazon S3 Customer Data Isolation

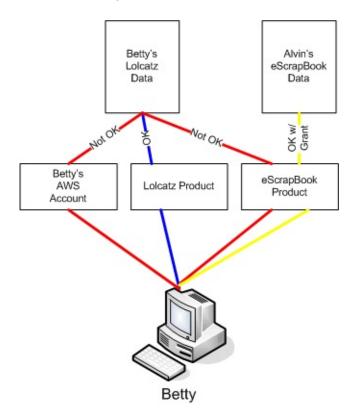
Amazon DevPay requests store and access data on behalf of the users of your product. The resources created by your application are owned by your users; unless you modify the ACL, you cannot read or modify the user's data.

Data stored by your product is isolated from other Amazon DevPay products and general Amazon S3 access. Customers that *store* data in Amazon S3 through your product can only *access* that data through your product. The data cannot be accessed through other Amazon DevPay products or through a personal AWS account.

Two users of a product can only access each other's data if your application explicitly grants access through the ACL.

Example

The following figure illustrates allowed, disallowed, and conditional (discretionary) data access.



Betty's access is limited as follows:

- She can access Lolcatz data through the Lolcatz product. If she attempts to access her Lolcatz data through another product or a personal AWS account, her requests will be denied.
- She can access Alvin's eScrapBook data through the eScrapBook product if access is explicitly granted.

Amazon DevPay Token Mechanism

To enable you to make requests on behalf of your customers and ensure that your customers are billed for use of your application, your application must send two tokens with each request: the product token and the user token.

The product token identifies your product; you must have one product token for each Amazon DevPay product that you provide. The user token identifies a user in relationship to your product; you must have a user token for each user/product combination. For example, if you provide two products and a user subscribes to each, you must obtain a separate user token for each product.

For information on obtaining product and user tokens, refer to the *Amazon DevPay Developer Guide*.

Amazon S3 and Amazon DevPay Authentication

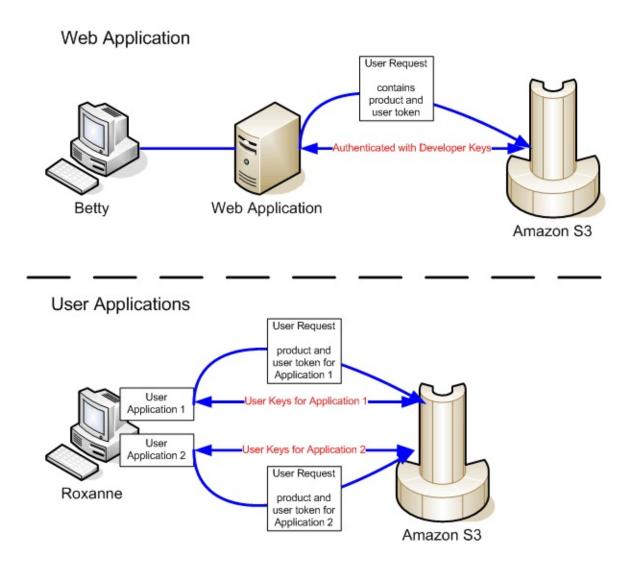
Although the token mechanism uniquely identifies a customer and product, it does not provide authentication.

Normally, your applications communicate directly with Amazon S3 using your Access Key ID and Secret Access Key. For Amazon DevPay, Amazon S3 authentication works a little differently.

If your Amazon DevPay product is a web application, you securely store the Secret Access Key on your servers and use the user token to specify the customer for which requests are being made.

However, if your Amazon S3 application is installed on your customers' computers, your application must obtain an Access Key ID and a Secret Access Key for each installation and must use those credentials when communicating with Amazon S3.

The following figure shows the differences between authentication for web applications and user applications.



Amazon S3 Bucket Limitation

Each of your customers can have up to 100 buckets for each Amazon DevPay product that you sell. For example, if a customer uses three of your products, the customer can have up to 300 buckets (100 * 3) plus any buckets outside of your Amazon DevPay products (i.e., buckets in Amazon DevPay products from other developers and the customer's personal AWS account).

Amazon S3 and Amazon DevPay Process

Following is a high-level overview of the Amazon DevPay process.

Launch Process

1	A customer signs up for your product through Amazon.
2	The customer receives an activation key.
3	The customer enters the activation key into your application.
4	Your application communicates with Amazon and obtains the user's token. If your application is installed on the user's computer, it also obtains an Access Key ID and Secret Access Key on behalf of the customer.
5	Your application provides the customer's token and the application product token when making Amazon S3 requests on behalf of the customer. If your application is installed on the customer's computer, it authenticates with the customer's credentials.
6	Amazon uses the customer's token and your product token to determine who to bill for the Amazon S3 usage.
7	Once a month, Amazon processes usage data and bills your customers according to the terms you defined.
8	AWS deducts the fixed Amazon DevPay transaction fee and pays you the difference. AWS then separately charges you for the Amazon S3 usage costs incurred by your customers and the percentage-based Amazon DevPay fee.

Additional Information

For information about using, setting up, and integrating with Amazon DevPay, refer to the *Amazon DevPay Developer Guide*.

Handling Errors

Topics

- The REST Error Response
- The SOAP Error Response
- Amazon S3 Error Best Practices

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

The REST Error Response

If a REST request results in an error, the HTTP reply has:

- An XML error document as the response body
- Content-Type: application/xml
- An appropriate 3xx, 4xx, or 5xx HTTP status code

Following is an example of a REST Error Response.

```
<?xml version="1.0" encoding="UTF-8"?>
<Error>
<Code>NoSuchKey</Code>
<Message>The resource you requested does not exist</Message>
<Resource>/mybucket/myfoto.jpg</Resource>
<RequestId>4442587FB7D0A2F9</RequestId>
</Error>
```

For more information about Amazon S3 errors, go to ErrorCodeList.

Response Headers

Following are response headers returned by all operations:

- *x*-*amz*-*r*equest-*i*d: A unique ID assigned to each request by the system. In the unlikely event that you have problems with Amazon S3, Amazon can use this to help troubleshoot the problem.
- *x*-*amz*-*id*-*2*: A special token that will help us to troubleshoot problems.

Error Response

Topics

•

Error Code

- Error Message
- <u>Further Details</u>

When an Amazon S3 request is in error, the client receives an error response. The exact format of the error response is API specific: For example, the REST error response differs from the SOAP error response. However, all error responses have common elements.

Error Code

The error code is a string that uniquely identifies an error condition. It is meant to be read and understood by programs that detect and handle errors by type. Many error codes are common across SOAP and REST APIs, but some are APIspecific. For example, NoSuchKey is universal, but UnexpectedContent can occur only in response to an invalid REST request. In all cases, SOAP fault codes carry a prefix as indicated in the table of error codes, so that a NoSuchKey error is actually returned in SOAP as Client.NoSuchKey.

Error Message

The error message contains a generic description of the error condition in English. It is intended for a human audience. Simple programs display the message directly to the end user if they encounter an error condition they don't know how or don't care to handle. Sophisticated programs with more exhaustive error handling and proper internationalization are more likely to ignore the error message.

Further Details

Many error responses contain additional structured data meant to be read and understood by a developer diagnosing programming errors. For example, if you send a Content-MD5 header with a REST PUT request that doesn't match the digest calculated on the server, you receive a BadDigest error. The error response also includes as detail elements the digest we calculated, and the digest you told us to expect. During development, you can use this information to diagnose the error. In production, a well-behaved program might include this information in its error log.

The SOAP Error Response

In SOAP, an error result is returned to the client as a SOAP fault, with the HTTP response code 500. If you do not receive a SOAP fault, then your request was successful. The Amazon S3 SOAP fault code is comprised of a standard SOAP 1.1 fault code (either "Server" or "Client") concatenated with the Amazon S3-specific error code. For example: "Server.InternalError" or "Client.NoSuchBucket". The SOAP fault string element contains a generic, human readable error message in English. Finally, the SOAP fault detail element contains miscellaneous information relevant to the error.

For example, if you attempt to delete the object "Fred", which does not exist, the body of the SOAP response contains a "NoSuchKey" SOAP fault.

Example

```
<soapenv:Body>
<soapenv:Fault>
<Faultcode>soapenv:Client.NoSuchKey</Faultcode>
<Faultstring>The specified key does not exist.</Faultstring>
<Detail>
<Key>Fred</Key>
</Detail>
</soapenv:Fault>
</soapenv:Body>
```

For more information about the errors, go to ErrorCodeList.

Amazon S3 Error Best Practices

When designing an application for use with Amazon S3, it is important to handle Amazon S3 errors appropriately. This section describes issues to consider when designing your application.

Retry InternalErrors

Internal errors are errors that occur within the Amazon S3 environment.

Requests that receive an InternalError response might not have processed. For example, if a PUT request returns InternalError, a subsequent GET might retrieve the old value or the updated value.

If Amazon S3 returns an InternalError response, retry the request.

Tune Application for Repeated SlowDown errors

As with any distributed system, S3 has protection mechanisms which detect intentional or unintentional resource over-consumption and react accordingly. SlowDown errors can occur when a high request rate triggers one of these mechanisms. Reducing your request rate will decrease or eliminate errors of this type. Generally speaking, most users will not experience these errors regularly; however, if you would like more information or are experiencing high or unexpected SlowDown errors, please post to our Amazon S3 developer forum http://developer.amazonwebservices.com/connect/forum.jspa?forumID=24 or sign up for AWS Premium Support http://aws.amazon.com/premiumsupport/.

Isolate Errors

Amazon S3 provides a set of error codes that are used by both the SOAP and REST API. The SOAP API returns standard Amazon S3 error codes. The REST API is designed to look like a standard HTTP server and interact with existing HTTP clients (e.g., browsers, HTTP client libraries, proxies, caches, and so on). To ensure the HTTP clients handle errors properly, we map each Amazon S3 error to an HTTP status code.

HTTP status codes are less expressive than Amazon S3 error codes and contain less information about the error. For example, the NoSuchKey and NoSuchBucket Amazon S3 errors both map to the HTTP 404 Not Found status code.

Although the HTTP status codes contain less information about the error, clients that understand HTTP, but not the Amazon S3 API, will usually handle the error correctly.

Therefore, when handling errors or reporting Amazon S3 errors to end users, use the Amazon S3 error code instead of the HTTP status code as it contains the most information about the error. Additionally, when debugging your application, you should also consult the human readable <Details> element of the XML error response.

Server Access Logging

Topics

•

Server Access Logging Configuration API

- <u>Delivery of Server Access Logs</u>
- <u>Server Access Log Format</u>
- <u>Setting Up Server Access Logging</u>



Important

This section describes Beta functionality that is subject to change in future releases. Please provide feedback on this functionality in the <u>Amazon S3</u> <u>Developer Forum</u>.

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

By default, server access logs are not collected for a bucket. To learn how to enable server access logging, see <u>Server Access Logging Configuration API</u>.

Once logging is enabled for a bucket, available log records are periodically aggregated into log files and delivered to you via an Amazon S3 bucket of your choosing. For a detailed description of this process, see <u>Delivery of Server</u> <u>Access Logs</u>.

For information on how to interpret the contents of log files, see <u>Server Access</u> <u>Log Format</u>.

To walk through the process of enabling logging for your bucket, see <u>Setting Up</u> <u>Server Access Logging</u>.



There is no extra charge for enabling the server access logging feature on an

Amazon S3 bucket, however any log files the system delivers to you will accrue the usual charges for storage (you can delete the log files at any time). No data transfer charges will be assessed for log file delivery, but access to the delivered log files is charged for data transfer in the usual way.

Server Access Logging Configuration API



Important

This section describes Beta functionality that is subject to change in future releases. Please provide feedback on this functionality in the

Amazon S3 Developer Forum.

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

The BucketLoggingStatus element has the following structure.

Example

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

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

LoggingEnabled

The presence of this element indicates that server access logging is enabled

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

TargetBucket

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



The source and the target buckets must be in the same location. For more information about bucket location constraints, see <u>How to Select a Region</u> <u>for Your Buckets</u>

• TargetPrefix

This element lets you specify a prefix for the keys that the delivered log files will be stored under. For information on how the key name for log files is constructed, see <u>Delivery of Server Access Logs</u>.

• TargetGrants

The bucket owner is automatically granted FULL_CONTROL to all logs delivered to the bucket. This optional element enables you grant access to others. Any specified TargetGrants are added to the default ACL. For more information about ACLs, see <u>Access Control Lists</u>.

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

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

```
GET ?logging HTTP/1.1
```

```
Host: mybucket.s3.amazonaws.com
Date: Wed, 01 Mar 2006 12:00:00 GMT
Authorization: AWS your_aws_access_key_ID: your_sIgnature_Here
HTTP/1.1 200 OK
Date: Wed, 01 Mar 2006 12:00:00 GMT
Connection: close
Server: AmazonS3
<?xml version="1.0" encoding="UTF-8"?>
<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <LoggingEnabled>
    <TargetBucket>mybucketlogs</TargetBucket>
    <TargetPrefix>mybucket-access log-/</TargetPrefix>
        <TargetGrants>
            <Grant>
                <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-ins</pre>
                     <EmailAddress>user@company.com</EmailAddress>
                </Grantee>
                <Permission>READ</Permission>
            </Grant>
        </TargetGrants>
  </LoggingEnabled>
</BucketLoggingStatus>
```

In SOAP, you can work with BucketLoggingStatus resource using the <u>SOAPSetBucketLoggingStatus</u> and <u>SOAPGetBucketLoggingStatus</u> operations.

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

BucketLoggingStatus Changes Take Effect Over Time

Changes to the logging status for a bucket are visible in the configuration API immediately, but they take time to actually affect the delivery of log files. For example, if you enable logging for a bucket, some requests made in the following hour might be logged, while others might not. Or, if you change the target bucket for logging from bucket A to bucket B, some logs for the next hour might continue to be delivered to bucket A, while others might be delivered to the new target bucket B. In all cases, the new settings will eventually take effect without any further action on your part.

Delivery of Server Access Logs



This section describes Beta functionality that is subject to change in future releases. Please provide feedback on this functionality in the

Amazon S3 Developer Forum.

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

🎒 No	te
------	----

The source and the target buckets must be in the same location. For more information about bucket location constraints, see <u>How to Select a Region for</u> <u>Your Buckets</u>.

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

TargetPrefixYYYY-mm-DD-HH-MM-SS-UniqueString

In the key, YYYY, mm, DD, HH, MM and SS are the digits of the year, month, day, hour, minute, and seconds (respectively) when the log file was delivered.

A log file delivered at a specific time can contain records written at any point before that time. There is no way to know whether all log records for a certain time interval have been delivered or not.

The TargetPrefix component of the key is a string provided by the bucket owner using the logging configuration API. For more information, see <u>Server Access</u> <u>Logging Configuration API</u>.

The UniqueString component of the key carries no meaning and should be

ignored by log processing software.

The system does not delete old log files. If you do not want server logs to accumulate, you must delete them yourself. To do so, use the List operation with the prefix parameter to locate old logs to delete. For more information, see Listing Keys.

Access Control Interaction

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

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

Best Effort Server Log Delivery

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

However, the server logging feature is offered on a best-effort basis. The completeness and timeliness of server logging is not guaranteed. The log record for a particular request might be delivered long after the request was actually processed, or it might not be delivered at all. The purpose of server logs is to give the bucket owner an idea of the nature of traffic against his or her bucket. It is not meant to be a complete accounting of all requests.

Usage Report Consistency

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

Server Access Log Format



This section describes Beta functionality that is subject to change in future releases. Please provide feedback on this functionality in the

Amazon S3 Developer Forum.

The log files consist of a sequence of new-line delimited log records. Log records appear in no particular order. Each log record represents one request and consists of space delimited fields described in the following table.

Field Name	Example Entry	Notes
Bucket Owner	314159b66967d86f031c7249d1d9a8024 9109428335cd0ef1cdc487b4566cb1b	The canonical us of the source buc
Bucket	mybucket	The name of the request was proc the system receiv request and cann bucket, the reque in any server acc
Time	[04/Aug/2006:22:34:02 +0000]	The time at whic received. The for strftime() term [%d/%B/%Y:%H:%
Remote IP	72.21.206.5	The apparent Int the requester. Int and firewalls mig actual address of making the reque
Requester	314159b66967d86f031c7249d1d9a80 249109428335cd0ef1cdc487b4566cb1b	The canonical us requester, or the "Anonymous" fo requests. This ide one used for acce purposes.
Request		The request ID is

ID	3E57427F33A59F07	generated by An uniquely identify
Operation	SOAP.CreateBucket	Either SOAP.ope REST.HTTP_meth
	or	
	REST.PUT.OBJECT	
Key	/photos/2006/08/puppy.jpg	The "key" part of encoded, or "-" does not take a
Request- URI	"GET /mybucket/photos/2006/08/	The Request-UF request message
	puppy.jpg?x-foo=bar"	
HTTP status	200	The numeric HT the response.
Error Code	NoSuchBucket	The Amazon Sa if no error occur
Bytes Sent	2662992	The number of nexcluding HTTI overhead, or "-"
Object Size	3462992	The total size of question.
Total Time	70	The number of r request was in f server's perspec
		measured from request is receiv the last byte of t Measurements r client's perspect due to network
Turn- Around	10	The number of r Amazon S3 spen

Time		request. This val from the time the request was rece the first byte of t sent.
Referrer	"http://www.amazon.com/webservices"	The value of the header, if present agents (e.g. brow this header to the linking or embed making a request
User- Agent	"curl/7.15.1"	The value of the header.

Any field can be set to "-" to indicate that the data was unknown or unavailable, or that the field was not applicable to this request.

Custom Access Log Information

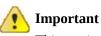
You can include custom information to be stored in the access log record for a request by adding a custom query-string parameter to the URL for the request. Amazon S3 will ignore query-string parameters that begin with "x-", but will include those parameters in the access log record for the request, as part of the Request-URI field of the log record. For example, a GET request for "s3.amazonaws.com/mybucket/photos/2006/08/puppy.jpg?x-user=johndoe" will work the same as the same request for

"s3.amazonaws.com/mybucket/photos/2006/08/puppy.jpg", except that the "x-user=johndoe" string will be included in the Request-URI field for the associated log record. This functionality is available in the REST interface only.

Extensible Server Access Log Format

From time to time, we might extend the access log record format by adding new fields to the end of each line. Code that parses server access logs must be written to handle trailing fields that it does not understand.

Setting Up Server Access Logging



This section describes Beta functionality that is subject to change in future releases. Please provide feedback on this functionality in the

Amazon S3 Developer Forum.

The Amazon S3 server access logging feature lets you generate access log files for buckets that you own. These log files are delivered to you by writing them into a (possibly different) bucket that you own. Once delivered, the access logs are ordinary objects that you can read, list or delete at your convenience.

These instructions assume that you want to enable server access logging on one of your pre-existing buckets, and that you want to have those logs delivered into a new bucket you will create just for logging. We suppose that the bucket you want to log access to is called 'mybucket' and the new bucket you will create to hold your access logs is called 'mylogs'. This makes 'mybucket' the source bucket for logging and 'mylogs' the target bucket for logging. Whenever you see 'mybucket' or 'mylogs' in the example, replace them with the name of your bucket that you want to log, and the bucket you want to store your access logs, respectively.

This tutorial makes use of s3curl (go to <u>s3curl.pl sample program</u>) to work with the Amazon S3 REST API. Make sure you use the most recent version of s3curl, as it has been updated to support this tutorial. After invoking s3curl, always check for a 200 OK HTTP response. If you get some other response code, refer to the XML error response which likely contains information about what went wrong.

Preparing the Target Bucket

To prepare the target bucket

1. First, decide if you want your logs delivered to an existing bucket, or if you want to create a new bucket just for access log files. Following is a command that creates a new target bucket for logging. Notice the canned ACL argument that grants the system permission to write log files to this bucket.



The source and the target buckets must be in the same location. For more information about bucket location constraints, see <u>How to Select a Region</u> <u>for Your Buckets</u>

\$./s3curl.pl --id your_aws_access_key_id --key your_aws_secret_access_key --

2. If you just created a new bucket for logging, skip to the next section. Otherwise, to have your access logs files delivered to an existing bucket, you must modify the access control policy of that bucket by hand. Fetch the ?acl sub-resource of the target bucket and save it to a local file:

```
$ ./s3curl.pl --id YOUR_AWS_ACCESS_KEY_ID --Key YOUR_AWS_SECRET_ACCESS_KEY --
```

3. Now open the local copy of the logging resource in your favorite text editor and insert a new <Grant> element to the <AccessControlList> section that gives the log delivery group WRITE and READ_ACP permission to your bucket.



```
<URI>http://acs.amazonaws.com/groups/s3/LogDelivery</URI>
</Grantee>
</Grant>
</Grant>
<Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<URI>http://acs.amazonaws.com/groups/s3/LogDelivery</URI>
</Grantee>
<Permission>READ_ACP</Permission>
</Grant>
```

4. Finally, apply the modified access control policy by writing it back to Amazon S3.

\$./s3curl.pl --id your_aws_access_key_id --key your_aws_secret_access_key --

Enabling Server Access Logging on the Source Bucket

Now that the target bucket can accept log files, we'll update the ?logging subresource of the source bucket to turn on server access logging. Remember that you must be the bucket owner to read or write this resource.

Fetch the *?logging* sub-resource for modification using the command shown in the following example.

Example

```
$ ./s3curl.pl --id your_aws_access_key_id --key your_aws_secret_access_key -- -s
```

Open mybucket.logging in your favorite text editor and uncomment the <LoggingSettings> section. Replace the contents of the <TargetBucket> and <TargetPrefix> with 'mylogs' and 'mybucket-access_log-' respectively.

Additionally, to grant users access to log files within the bucket, you can specify one or more users in the <TargetGrants> section, You can specify users through their e-mail address (EmailAddress) or canonical user ID (CanonicalUser). Permissions include READ, WRITE, and FULL_CONTROL. The result should be similar to the following.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">
<LoggingEnabled>
<TargetBucket>mylogs</TargetBucket>
<TargetPrefix>mybucket-access_log-/</TargetPrefix>
<TargetGrants>
```





Note For general information about authentication, see <u>Authentication and Access</u> <u>Control</u>.

Now apply your modifications by writing the document back to the ?logging sub-resource in Amazon S3.

Example

```
$ ./s3curl.pl --id your_aws_access_key_id --key your_aws_secret_access_key --put |
```

You can confirm your changes by fetching the *?logging* sub-resource and comparing it to what you just wrote.

Server access logging should now be enabled. Make a few requests against the source bucket now, and your access logs should begin to be delivered to the target bucket within the next few hours.

Disabling Server Logging for a Bucket

Fetch, modify, and apply the ?logging sub resource in the same way as described in the preceding procedure, except use your text editor to remove the <LoggingEnabled> element.



Log changes do not take effect immediately; logs will be delivered for a while after disabling logging.

Glossary

100-continue

A method that enables a client to see if a server can accept a request before actually sending it. For large PUTs, this can save both time and bandwidth charges.

account

AWS account associated with a particular developer.

authentication

The process of proving your identity to the system.

bucket

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

canned access policy

A standard access control policy that you can apply to a bucket or object. Options include: private, public-read, public-read-write, authenticated-read.

canonicalization

The process of converting data into a standard format that will be recognized by a service such as Amazon S3.

consistency model

The method through which Amazon S3 achieves high availability, which involves replicating data across multiple servers within Amazon's data centers. After a "success" is returned, your data is safely stored. However, information about the changes might not immediately replicate across Amazon S3.

key

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

metadata

The metadata is a set of name-value pairs that describe the object. These include default metadata such as the date last modified and standard HTTP metadata such as Content-Type. The developer can also specify custom metadata at the time the Object is stored.

object

The fundamental entities stored in Amazon S3. Objects consist of object data and metadata. The data portion is opaque to Amazon S3.

service endpoint

The host and port with which you are trying to communicate within the destination URL. For virtual hosted-style requests, this is mybucket.s3.amazonaws.com. For path-style requests, this is s3.amazonaws.com

Document Conventions

This section lists the common typographical and symbol use conventions for AWS technical publications.

Typographical Conventions

This section describes common typographical use conventions.

Convention	Description/Example		
Call-outs	A call-out is a number in the body text to give you a visual reference. The reference point is further discussion elsewhere.		
	You can use this resource regularly.		
Code in text	Inline code samples (including XML) and commands are identified with a special font.		
	You can use the command java -version.		
Code blocks	Blocks of sample code are set apart from the body and marked accordingly.		
	<pre># ls -l /var/www/html/index.html -rw-rw-r 1 root root 1872 Jun 21 09:33 /var/www/html/inde # date Wed Jun 21 09:33:42 EDT 2006</pre>		
Emphasis	Unusual or important words and phrases are marked with a special font.		
	You <i>must</i> sign up for an account before you can use the service.		
Internal cross references	References to a section in the same document are marked.		
	For more information, see <u>Document Conventions</u> .		
Logical	A special font is used for expressions that are important to identify, but are not code.		
values, constants,	If the value is null, the returned response will be false.		
and regular expressions, abstracta			
Product and	Named AWS products and features are identified on first use.		
feature names	Create an Amazon Machine Image (AMI).		
Operations	In-text references to operations.		
	Use the GetHITResponse operation.		
Parameters	In-text references to parameters.		
	The operation accepts the parameter <i>AccountID</i> .		

Response elements	In-text references to responses. A container for one CollectionParent and one or more CollectionItems.		
Technical publication references	References to other AWS publications. If the reference is hyperlinked, it is also underscored For detailed conceptual information, refer to the <i>Amazon Mechanical Turk Developer Guide</i>		
User entered values	A special font marks text that the user types. At the password prompt, type MyPassword.		
User interface controls and labels	Denotes named items on the UI for easy identification. On the File menu, click Properties .		
Variables	 When you see this style, you must change the value of the content when you copy the text of sample to a command line. % ec2-register <<i>your-s3-bucket</i>>/image.manifest See also the following symbol convention. 		

Symbol Conventions

This section describes the common use of symbols.

Convention	Symbol	Description/Example
Mutually exclusive parameters	(Parentheses and vertical bars)	Within a code description, bar separators denote options from which one must be chosen. % data = hdfread (start stride edge)
Optional parameters XML variable text	[square brackets]	Within a code description, square brackets denote completely optional commands or parameters. % sed [-n, -quiet] Use square brackets in XML examples to differentiate them from tags. <customerid>[ID]</customerid>
Variables	<arrow brackets=""></arrow>	Within a code sample, arrow brackets denote a variable that must be replaced with a valid value. % ec2-register <your-s3-bucket>/image.manifest</your-s3-bucket>

Index

Symbols

100-continue, Redirects and 100-Continue

A

access control, Authentication and Access Control access logs, **Delivery of Server Access Logs** access policy **REST, <u>REST Access Control Policy</u>** SOAP, Setting Access Policy with SOAP Adobe Flash, Browser-Based Uploads Using POST Amazon DevPay, Using Amazon DevPay with Amazon S3 API, Amazon S3 Application Programming Interfaces (API) REST, The REST Interface, Using the REST API SOAP, The SOAP Interface, Using the SOAP API audience, Who Should Read this Guide authentication, Authentication and Access Control debugging, **REST Request Signing Problems REST**, Authenticating REST Requests SOAP, Authenticating SOAP Requests authentication header, The Authentication Header

B

billing, Paying for Amazon S3
BitTorrent, Using BitTorrent with Amazon S3

charges, How You are Charged for BitTorrent Delivery
publishing, Publishing Content Using Amazon S3 and BitTorrent
retrieving objects, Using BitTorrent to Retrieve Objects Stored in Amazon S3

browser uploads, Browser-Based Uploads Using POST
buckets, Buckets, Working with Amazon S3 Buckets

access control, Buckets and Access Control
billing, Billing and Reporting of Buckets

configuration, <u>Bucket Configuration Options</u> location selection, <u>How to Select a Region for Your Buckets</u> restrictions, <u>Bucket Restrictions and Limitations</u> virtual hosting, <u>Virtual Hosting of Buckets</u>

С

CanonicalizedAmzHeaders element, Constructing the CanonicalizedAmzHeaders Element changes, What's New charges, Paying for Amazon S3 chunked downloads, Chunked and Resumable Downloads components, Amazon S3 Concepts concepts API, Amazon S3 Application Programming Interfaces (API) buckets, Buckets components, Amazon S3 Concepts keys, Keys objects, **Objects** operations, **Operations REST API**, The REST Interface SOAP API, The SOAP Interface configuring logging, Server Access Logging Configuration API consistency model, Amazon S3 Data Consistency Model copying objects, overview, Copying Amazon S3 Objects costs, Paying for Amazon S3

D

data model, <u>Amazon S3 Data Consistency Model</u> delimiter, <u>Common List Request Parameters</u>, <u>Listing Keys Hierarchically using</u> <u>Prefix and Delimiter</u> DevPay, <u>Using Amazon DevPay with Amazon S3</u> DNS, <u>DNS Considerations</u> DNS routing, <u>DNS Routing</u>, <u>Temporary Request Redirection</u>, <u>Permanent</u> <u>Request Redirection</u> downloads, chunked and resumable, <u>Chunked and Resumable Downloads</u>

elements

REST, <u>Common REST API Elements</u> SOAP, <u>Common SOAP API Elements</u> errors, <u>Error Code</u> details, <u>Further Details</u> isolation, <u>Isolate Errors</u> messages, <u>Error Message</u> response, <u>Error Response</u> REST response, <u>The REST Error Response</u> SlowDown, <u>Tune Application for Repeated SlowDown errors</u> SOAP response, <u>The SOAP Error Response</u>

F

features, <u>Advantages to Amazon S3</u> file size, maximum, <u>Advantages to Amazon S3</u> Flash, Adobe, <u>Browser-Based Uploads Using POST</u>

G

glossary, <u>Glossary</u> guide organization, <u>How This Guide Is Organized</u>

Η

HTTP user agents, Redirects and HTTP User-Agents

I

introduction, <u>Introduction to Amazon S3</u> IsTruncated, <u>Common List Response Elements</u>

J

Java, <u>DNS Considerations</u> JVM cache, <u>DNS Considerations</u>

Ε

keys, <u>Keys</u>

listing, <u>Listing Keys</u> listing hierarchically, <u>Listing Keys Hierarchically using Prefix and</u> <u>Delimiter</u> multi-page results, <u>Iterating Through Multi-Page Results</u> request parameters, <u>Common List Request Parameters</u> responses, <u>Common List Response Elements</u> using, <u>Keys</u>

L

listing keys, hierarchically, <u>Listing Keys Hierarchically using Prefix and</u> <u>Delimiter</u>

location constraints, <u>How to Select a Region for Your Buckets</u> logs, <u>Server Access Logging</u>

best effort delivery, <u>Best Effort Server Log Delivery</u> changing settings, <u>BucketLoggingStatus Changes Take Effect Over Time</u> configuration, <u>Server Access Logging Configuration API</u> delivery, <u>Delivery of Server Access Logs</u> format, <u>Server Access Log Format</u> setting up, <u>Setting Up Server Access Logging</u>

Μ

marker, <u>Common List Request Parameters</u> MaxKeys, <u>Common List Request Parameters</u> metadata, using, <u>Metadata</u> model, <u>Amazon S3 Data Consistency Model</u>

Ν

NextMarker, Common List Response Elements

0

object size, maximum, Advantages to Amazon S3

K

objects, <u>Objects</u> copying, <u>Copying Amazon S3 Objects</u> getting, <u>Getting Objects</u> using, <u>Working with Amazon S3 Objects</u> operations, <u>Operations</u> organization of guide, <u>How This Guide Is Organized</u> overview, <u>Overview of Amazon S3</u>

Р

pagination, <u>Iterating Through Multi-Page Results</u> paying, <u>Paying for Amazon S3</u> performance optimization, <u>Performance Optimization</u> PHP virtual machine, <u>DNS Considerations</u> POST, <u>Browser-Based Uploads Using POST</u> prefix, <u>Common List Request Parameters</u>, <u>Listing Keys Hierarchically using</u> <u>Prefix and Delimiter</u>

R

redirection, Request Redirection and the REST API permanent, Permanent Request Redirection request, Request Redirection and the REST API temporary, Temporary Request Redirection referrer, Server Access Log Format region, Common SOAP API Elements, How to Select a Region for Your Buckets request redirection, Request Redirection and the REST API access policy, Request Redirection and the REST API request routing, Request Routing resources, related, Amazon S3 Resources REST access policy, REST Access Control Policy API, Using the REST API authentication, Authenticating REST Requests examples, Authentication Examples header, <u>The Authentication Header</u> debugging authentication, **REST Request Signing Problems** elements, Common REST API Elements

POST, <u>Browser-Based Uploads Using POST</u> StringToSign, <u>Positional versus Named HTTP Header StringToSign</u> <u>Elements</u> time stamp, <u>Time Stamp Requirement</u> restrictions, <u>Bucket Restrictions and Limitations</u> resumable downloads, <u>Chunked and Resumable Downloads</u> routing, <u>Request Routing</u> DNS, <u>DNS Routing</u>

S

server access logs, <u>Server Access Logging</u>, <u>Delivery of Server Access Logs</u> SetObjectAccessControlPolicy SOAP, <u>Using BitTorrent with Amazon S3</u> size, object, <u>Advantages to Amazon S3</u> SOAP access policy, <u>Setting Access Policy with SOAP</u> API, <u>Using the SOAP API</u> authentication, <u>Authenticating SOAP Requests</u> elements, <u>Common SOAP API Elements</u> error response, <u>The SOAP Error Response</u> storage limit, <u>Advantages to Amazon S3</u> StringToSign, <u>Positional versus Named HTTP Header StringToSign Elements</u> system metadata, <u>Metadata</u>

Т

TCP optimization, <u>Performance Optimization</u> time stamp, <u>Time Stamp Requirement</u> TTLs, clients, <u>DNS Considerations</u>

U

uploads, browser, <u>Browser-Based Uploads Using POST</u> user metadata, <u>Metadata</u>

V

virtual hosted buckets, <u>Virtual Hosting of Buckets</u> virtual machines, <u>DNS Considerations</u>