




# EMC ATMOS Programmer's Guide

Version 1.3.0

Revision A

**EMC CORPORATION**  
*Corporate Headquarters:*  
Hopkinton, MA 01748-9103  
1-508-435-1000  
[www.EMC.com](http://www.EMC.com)



© 2010 EMC Corporation. All rights reserved.

EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

The information in this publication is provided “as is.” EMC Corporation makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose.

Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.

EMC is a registered trademark and Atmos is a trademark of EMC Corporation. All other trademarks used herein are the property of their respective owners.

# Contents

<b>Preface</b> .....	<b>7</b>
Documentation .....	7
Typographic Conventions .....	8

---

## PART I: INTRODUCTION

<b>1</b>	<b>Getting Started</b> .....	<b>10</b>
	Using the Object Interface .....	10
	Example: Creating an Object .....	10
	System Metadata .....	12
	Example: Getting System Metadata .....	13
	Non-listable User Metadata .....	13
	Example: Creating an Object with Non-listable User Metadata .....	14
	Example: Setting (Non-listable) User Metadata .....	14
	Listable User Metadata .....	15
	Example: Creating an Object with Listable Metadata Tags .....	15
	Example: Setting Listable Metadata Tags .....	16
	Using the Namespace Interface .....	16
	Example: Creating a Directory .....	17
	Example: Creating a File in a Directory .....	17
	Example: Listing a Directory .....	18
	Example: Reading a File .....	18
	Example: Reading Part of a File .....	19
	Example: Updating a file .....	20

---

## PART II: REST API

<b>2</b>	<b>Common REST Headers</b> .....	<b>22</b>
	Standard HTTP Headers .....	22
	Custom Headers .....	24
<b>3</b>	<b>REST API Reference</b> .....	<b>29</b>
	Specifying Objects/Files in REST Commands .....	29
	REST Commands .....	30
	Creating an Object .....	32

Deleting an Object .....	34
Deleting User Metadata .....	35
Getting an ACL .....	36
Getting Listable Tags .....	37
Getting Object Info .....	38
Getting Service Information .....	42
Getting System Metadata .....	43
Getting User Metadata .....	45
Listing Objects .....	46
Listing User Metadata Tags .....	52
Listing Versions .....	53
Reading an Object .....	54
Setting an ACL .....	60
Setting User Metadata .....	61
Updating an Object .....	62
Versioning an Object .....	64

---

## PART III: SOAP API

<b>4</b>	<b>SOAP WSDL Schema Structure .....</b>	<b>66</b>
	Common Elements .....	66
	ACL .....	67
	Grant .....	67
	Grantee .....	68
	Permission .....	68
	DirectoryList .....	69
	ExtentType .....	69
	MetadataTags .....	70
	TagEntry .....	70
	ObjectEntry .....	70
	SystemMetadataList .....	71
	SystemMetadataEntry .....	71
	UserMetadataList .....	71
	UserMetadataEntry .....	72
	UID .....	72
<b>5</b>	<b>SOAP API Reference .....</b>	<b>73</b>
	Creating an Object .....	74
	Deleting an Object .....	77
	Deleting User Metadata .....	78
	Getting an ACL .....	79

Getting Listable Tags .....	81
Getting System Metadata .....	82
Getting User Metadata .....	85
Listing Objects .....	87
Listing User Metadata Tags .....	90
Listing Versions .....	92
Reading an Object .....	93
Setting an ACL .....	101
Setting User Metadata .....	103
Updating an Object .....	105
Versioning an Object .....	108

---

## **PART IV: ADVANCED TOPICS**

### **6 Reserved Namespace for Extended Attributes ..... 110**

Linux Extended Attributes .....	110
Atmos Extended Attributes .....	111
Capability .....	112
Expiration of Objects .....	112
Layout Storage Object .....	113
MDS (Metadata Service) .....	114
Number of Links .....	114
Object ID .....	114
Queues .....	114
Reference Count .....	115
Retention of Objects .....	115
Statistics .....	116
Log Tracing .....	116
updateNum .....	116
Version .....	116

### **7 Security ..... 118**

Managing Authentication .....	118
REST Authentication: Algorithm for Securing REST Messages with Signatures .....	119
SOAP Authentication .....	121
Access Control Lists .....	122
REST ACLs .....	123
SOAP ACLs .....	123
Shareable URLs .....	124
URL Syntax .....	124
Example .....	124

	Calculating the Signature .....	125
<b>8</b>	<b>Error Messages and Status Codes .....</b>	<b>126</b>
	REST Information .....	126
	SOAP Information .....	126
	Error Codes .....	127
	HTTP Success Codes .....	129

# Preface

This document explains how to use EMC® Atmos™ Web-service APIs to create objects and manage their related metadata.

---

## Documentation

The following documents comprise the Atmos documentation set. You can find them at the EMC Powerlink Web site ([powerlink.emc.com](http://powerlink.emc.com)). After logging in, select Support > Technical Documentation and Advisories, then the links to software or hardware documentation for Atmos.

---

<b>Document</b>	<b>Description</b>	<b>Intended Audience</b>
<i>EMC Atmos Release Notes</i>	Presents summary information about changes in a given release.	Anyone preparing for a new Atmos release.
<i>EMC Atmos Conceptual Overview</i>	Presents a high-level overview of Atmos.	All Atmos users.
<i>EMC Atmos Installation and Upgrade Guide</i>	Explains how to install, configure, and upgrade Atmos on VMware and physical hardware as well as several optional RPMs.	User who install and configure Atmos.
<i>EMC Atmos System Administrator's Guide</i>	Describes how to use the system-management GUI to access and manage Atmos server software.	Atmos system managers. Also background for anyone who installs Atmos software.
<i>EMC Atmos Programmer's Guide</i>	Explains how to use the Atmos Web-service APIs to create objects and manage their related metadata.	Developers who build or deploy applications using the APIs.
<i>EMC Atmos System Management API Guide</i>	Describes how to use the Atmos System Management API to programmatically perform a subset of system management tasks.	Developers who build or deploy applications using the APIs.
<i>EMC Atmos Security Configuration Guide</i>	Provides an overview of settings available to ensure the secure operation of Atmos.	Atmos system managers.

---

<b>Document</b>	<b>Description</b>	<b>Intended Audience</b>
<i>EMC Atmos Non-EMC Software License Agreements</i>	Presents legal notices for non-EMC software that is available through Atmos.	—
<i>EMC Atmos Hardware Guide</i>	Describes the hardware which is the foundation of the Atmos platform.  This document has a different numbering scheme than the other, software manuals, because it has a different release schedule.	Anyone who installs and maintains Atmos hardware.

## Typographic Conventions

The following conventions are used in this guide:

<b>Conventions</b>	<b>Meaning</b>
<a href="#">Blue text</a>	Hyperlinked cross reference or URL.
<code>FixedWidth</code>	Commands, filenames, and code examples
<b><code>FixedWidthBold</code></b>	Emphasis within code examples
<i><code>FixedWidthItalics</code></i>	Variable names in text



**PART I:**

**INTRODUCTION**

# 1 Getting Started

EMC<sup>®</sup> Atmos<sup>™</sup> is an object-storage system with enormous scalability and extensibility. It uses metadata-driven policies to manage data placement and data services. The system offers multiple file- and object-access mechanisms; for example, file access is available using NFS, CIFS, and installable file systems for selected operating systems.

This guide describes the programmatic interfaces to create, read, update, and delete objects, and to manage object metadata. The object interface and local file system support metadata operations including version creation and tagging objects with user-defined metadata (to form user-defined collections of objects). APIs are available for REST and SOAP Web services.

Application developers use an API to create and manipulate objects and metadata. Applications can associate metadata with objects they store in the repository. Metadata can be used to trigger policies (defined by the system administrator) that meet customer goals for performance, data protection, content delivery, archiving, etc.

This chapter presents several sample Web-service requests and responses, using the REST Web service. The REST Web service uses HTTP methods like `POST`, `DELETE`, `PUT`, and `GET` to create, delete, update, and read objects and metadata. The chapter also contains some general background information about metadata.

The Web services APIs support both an *object interface* and a file-system-like *namespace interface*. The namespace allows users to create directories and assign names to files that they create.

For complete command descriptions, including sample requests and responses for each command, see [Chapter 3, “REST API Reference” on page 29](#) and [Chapter 5, “SOAP API Reference” on page 73](#).

---

## Using the Object Interface

### Example: Creating an Object

Consider the following simple example of creating an object. (Throughout this chapter, the line numbers are not part of the examples. They are included to clarify the discussion.)

```
1      POST /rest/objects HTTP/1.1
2      accept: */*
3      x-emc-useracl: john=FULL_CONTROL,mary=READ
4      date: Wed, 18 Feb 2009 16:03:52 GMT
5      content-type: application/octet-stream
6      x-emc-date: Wed, 18 Feb 2009 16:03:52 GMT
7      x-emc-groupacl: other=NONE
8      host: 168.159.116.96
9      content-length: 211
10     x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
11     x-emc-signature: KpT+3InilW+CS6YwJEAwyWvIlIs=
```

Each line of the example is explained briefly in the table following the example, along with a reference to other sections to obtain more detail.

<b>Line No.</b>	<b>Description</b>	<b>See ...</b>
1	Identifies the command and, where needed, the object being acted on. In this line, <code>/rest/objects</code> indicates an object is being referenced; alternately, you can specify a filename.	<a href="#">“Specifying Objects/Files in REST Commands” on page 29</a>
2	Standard HTTP header.	N/A
3	Sets access rights to the object for the specified user ID(s) (UIDs). In this case, Mary is granted read access and John is granted full access (read and write).	<a href="#">Chapter 2, “Common REST Headers” on page 22</a>
4	Specifies the date in UTC format, as defined in RFC 2616, section 3.3.1. Dates are used to (1) check whether a request is valid within the Web server's validity time window, and (2) compute signatures.	<a href="#">Chapter 2, “Common REST Headers” on page 22</a>
5	Specifies the type of object being stored.	<a href="#">Chapter 2, “Common REST Headers” on page 22</a>
6	Specifies the date in UTC format, as defined in RFC 2616, section 3.3.1. Dates are used to (1) check whether a request is valid within the Web server's validity time window, and (2) compute signatures.	<a href="#">Chapter 2, “Common REST Headers” on page 22</a>
7	Sets access rights to the object ID for the user group.	<a href="#">Chapter 2, “Common REST Headers” on page 22</a>
8	Standard HTTP header specifying the server that is the recipient of this request.	N/A
9	Standard HTTP header specifying the length of the request/response body, in bytes.	<a href="#">Chapter 2, “Common REST Headers” on page 22</a>
10	Specifies the UID of an application that is consuming the REST API and the ID of the subtenant to which that UID belongs. The format is subtenant-ID/application-ID.	<a href="#">Chapter 2, “Common REST Headers” on page 22</a>
11	Specifies the signature, which is a means for the system to authenticate the UID making the request.	<a href="#">Chapter 2, “Common REST Headers” on page 22</a>  <a href="#">“Managing Authentication” on page 118</a>

## System Metadata

System metadata is one of the two types of metadata. (The other type is user metadata; see [“Non-listable User Metadata” on page 13](#) and [“Listable User Metadata” on page 15](#).)

System metadata is generated automatically and updated by the system based on a predefined schema. The following table lists the system metadata that you can request from the Web service.

Name	Description	Example
atime	Last access time	2007-10-29T18:19:57Z
ctime	Last user-data modification time	2007-10-29T18:19:56Z
gid	Group ID	Apache
itime	Inception (create) time	2007-10-29T18:19:57Z
mtime	Last metadata modification time	2007-10-29T18:19:57Z
nlink	Number of hard links to a file. This is an internal, file-system reference count, generally not relevant to a user application	0
objectid	Object ID	4924264aa10573d404924281caf51f049242d810edc8
objname	Object name (filename or directory name), for objects created in the namespace. This is blank if the object does not have a name.	paris (for the directory photos/2008/paris/)  sunset.jpg (for the file photos/2008/paris/sunset.jpg)
policyname	Name of the policy under which the object is stored. If no policy was explicitly triggered for this object, either during creation or after the fact, the default policy is applied to it.	default
size	Object size in bytes	2971
type	String representing the data type, either “regular” (for objects or files) or “directory”	"regular" — OR — "directory"
uid	User ID (the owner)	user1

## Example: Getting System Metadata

### Request

The following example shows the request input for the `GetSystemMetadata` operation. The suffix `?metadata/system` is appended to the end of the object ID. In this case, all system metadata is returned; it also is possible to request only particular types of system metadata (see [“Getting System Metadata” on page 43](#) for REST or [“Getting System Metadata” on page 82](#) for SOAP).

```
GET /rest/objects/499ad542a1a8bc200499ad5a6b05580499c3168560a4?metadata/system HTTP/1.1
accept: */*
date: Thu, 05 Jun 2008 16:38:23 GMT
content-type: application/octet-stream
x-emc-date: Thu, 05 Jun 2008 16:38:23 GMT
host: 10.5.115.118
x-emc-uid: 6039ac182f194e15b9261d73ce044939/user1
x-emc-signature: 00NL3HHhyUXiRDTdWK52xqhQxTE=
```

### Response

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:36:18 GMT
Server: Apache
x-emc-meta: atime=2009-02-18T16:27:24Z, mtime=2009-02-18T16:03:52Z,
ctime=2009-02-18T16:27:24Z, itime=2009-02-18T16:03:52Z, type=regular, uid=user1,
gid=apache, objectid=499ad542a1a8bc200499ad5a6b05580499c3168560a4, objname=,
size=211, nlink=0, policyname=default
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

## Non-listable User Metadata

*User metadata* is a collection of text name-value pairs, which are not validated by the system. A user can store up to 127 user-metadata pairs. User metadata allows application developers the flexibility to create custom tags for data specific to the application they are developing. The Web service specifies user metadata using two types of tags: non-listable (described below) and listable (see [“Listable User Metadata” on page 15](#)).

*Non-listable* metadata tags (also just called metadata tags) are a way of classifying an object. Often, metadata tags are used to trigger policies; e.g., a tag-value pair of `Customer=Executive` could trigger a different policy than a tag-value pair of `Customer=Sales`. Talk to your system administrator to find out which metadata tag names trigger which policies in your system. (For an overview of policies, see *EMC Atmos Conceptual Overview*. For procedures to create policies, see *EMC Atmos Administrator's Guide*.)

There is no restriction on user metadata name size in Atmos, but user metadata values are restricted to 1 KB.

## Example: Creating an Object with Non-listable User Metadata

This example builds on the one in [“Example: Creating an Object” on page 10](#), with the addition of a line (in boldface) that define non-listable metadata tags:

```
1      POST /rest/objects HTTP/1.1
2      x-emc-meta: part1=buy
3      accept: */*
4      x-emc-useracl: john=FULL_CONTROL,mary=READ
5      date: Wed, 18 Feb 2009 16:03:52 GMT
6      content-type: application/octet-stream
7      x-emc-date: Wed, 18 Feb 2009 16:03:52 GMT
8      x-emc-groupacl: other=NONE
9      host: 168.159.116.96
10     content-length: 211
11     x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
12     x-emc-signature: KpT+3Ini1W+CS6YwJEAWYVvIlIs=
```

Line No.	Description	See ...
2	Sets the value of non-listable metadata tags. This might be used to trigger a policy that is implemented for this system; e.g., a policy might treat the data for a certain type of customer in a specific way.  In this case, the object being created is given a metadata tag named <code>part1</code> , with a value of <code>buy</code> .	<a href="#">Chapter 2, “Common REST Headers” on page 22</a>  <a href="#">“Non-listable User Metadata” on page 13</a>

## Example: Setting (Non-listable) User Metadata

In the following example, the `x-emc-meta` header specifies a non-listable user metadata tag for an already existing object (specified by the object ID on the first line):

```
POST /rest/objects/499ad542a1a8bc200499ad5a6b05580499c3168560a4?metadata/user HTTP/1.1
x-emc-meta: part1=order
accept: */*
date: Wed, 18 Feb 2009 16:27:24 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:27:24 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: OLI2TcDNWQ29gZv+ONr1ufCKA9M=
```

## Listable User Metadata

As mentioned above, there are two types of user metadata: non-listable (see [“Non-listable User Metadata” on page 13](#)) and listable (described below).

*Listable* metadata tags are metadata tags by which an object is indexed (and can be retrieved). For example, a user who wants to assign tags that classify the photos he took on vacation might create tags called beach, hotel, restaurant, and so on. This tagging can be done as part of the operations to create or update objects or set user metadata. In the file system, listable tags show up as directories containing symbolic links to the actual objects.

The same object can be tagged with multiple names; this is how tags differ from containers, as an object can belong to only one container. Note that these tags are private to the user who creates them; tags created by one user cannot be seen by another user.

There can be a hierarchy of listable metadata tags. For example, a listable metadata tag specification of `/vacation/2008/paris` creates a hierarchy of directories in the file system: `paris` is a subdirectory of `2008`, and `2008` is a subdirectory of `vacation`. The symbolic link to the object is under `paris`.

Users can use this functionality for easy indexing, searching, and retrieval of objects. For example, a user might have 2008 vacation pictures in two listable directories, `vacation/2008/paris` and `vacation/2008/china`. Then, he can easily retrieve a list of all pictures from his 2008 China vacation by issuing a `ListObjects` operation and specifying `vacation/2008/china` as the input parameter.

As another example, suppose we have a tag pair of `location=boston` for a new object, and we make the `location` tag listable. Then, if we perform a `ListObjects` operation with the tag argument specified as `location`, the object is returned in the response. If we remove `location` as a listable tag for the object, when we do a `ListObjects` request with the tag argument specified as `location`, the object is not returned.

As with non-listable user metadata, there is no restriction on listable user metadata name size in Atmos, but user metadata values are restricted to 1 KB.

## Example: Creating an Object with Listable Metadata Tags

This example builds on the one in [“Example: Creating an Object with Non-listable User Metadata” on page 14](#), with the addition of a line (in boldface) that define listable metadata tags:

```
1      POST /rest/objects HTTP/1.1
2      x-emc-listable-meta: part4/part7/part8=quick
3      x-emc-meta: part1=buy
4      accept: */*
5      x-emc-useracl: john=FULL_CONTROL,mary=READ
6      date: Wed, 18 Feb 2009 16:03:52 GMT
7      content-type: application/octet-stream
8      x-emc-date: Wed, 18 Feb 2009 16:03:52 GMT
9      x-emc-groupacl: other=NONE
10     host: 168.159.116.96
11     content-length: 211
12     x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
13     x-emc-signature: KpT+3Ini1W+CS6YwJEAwyVvIlIs=
```

Line No.	Description	See ...
2	Sets the value of listable metadata tags.  In this example, in the file system, <code>part8</code> is a subdirectory of <code>part7</code> , and <code>part7</code> is a subdirectory of <code>part4</code> . A symbolic link to the object, named <code>quick</code> , is under <code>part8</code> .	<a href="#">Chapter 2, “Common REST Headers” on page 22</a>  <a href="#">“Listable User Metadata” on page 15</a>

## Example: Setting Listable Metadata Tags

This example builds on the one in [“Example: Setting \(Non-listable\) User Metadata” on page 14](#), with the addition of a line (in boldface) that defines a listable user-metadata tag:

```
POST /rest/objects/499ad542a1a8bc200499ad5a6b05580499c3168560a4?metadata/user
HTTP/1.1
x-emc-listable-meta: part3=fast
x-emc-meta: part1=order
accept: */*
date: Wed, 18 Feb 2009 16:27:24 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:27:24 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: OLI2TcDNWQ29gZv+ONr1ufCKA9M=
```

## Using the Namespace Interface

Atmos Web services allow you to assign a filename to an object when creating the object. This enables clients to use their own name when referring to an object (filename), rather than an object ID that Atmos assigns to the object.

**NOTE:** *Directly under the / directory, you can create only directories, not files.* While creating a file, if you refer to a directory that does not exist, it is created automatically. In the example above, if the `photos` directory does not exist, it is created for you.

For information about constructing commands to specify files instead of objects, see [“Specifying Objects/Files in REST Commands” on page 29](#).

A simple example is shown below. Examples of using the namespace interface for each command are in [Chapter 3, “REST API Reference” on page 29](#) and [Chapter 5, “SOAP API Reference” on page 73](#).

## Example: Creating a Directory

To create a directory, you must use the namespace interface. When specifying the object's name, a trailing slash (/) identifies it as a directory (e.g., `mydirectory/`).

### Request

```
POST /rest/namespace/mydirectory/ HTTP/1.1
accept: */*
date: Mon, 03 Aug 2009 13:30:13 GMT
content-type: application/octet-stream
x-emc-date: Mon, 03 Aug 2009 13:30:13 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: Qfq/rwMcQh74P18W4JkyTJiPZW4=
```

### Response

```
HTTP/1.1 201 Created
Date: Mon, 03 Aug 2009 13:30:13 GMT
Server: Apache
location: /rest/objects/4a3fd8dfa2a8482004a3fd9315cf4704a76e665d80be
x-emc-delta: 0
x-emc-policy:
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

## Example: Creating a File in a Directory

To create a file in a directory, include the parent directory's name in the namespace request (below, `mydirectory/samplefile`).

If any intermediate directories do not exist, they are created automatically.

### Request

```
POST /rest/namespace/mydirectory/samplefile HTTP/1.1
accept: */*
date: Mon, 03 Aug 2009 13:32:34 GMT
content-type: application/octet-stream
x-emc-date: Mon, 03 Aug 2009 13:32:34 GMT
host: 168.159.116.96
content-length: 27
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: onk0Z8dvgqxKk6wDh1DznKrZqfM=
content-md5: i0WfxUgeVagRWCSGes+hUQ==
content for this file
```

### Response

```
HTTP/1.1 201 Created
Date: Mon, 03 Aug 2009 13:32:34 GMT
Server: Apache
location: /rest/objects/4a3fd8dfa2a8482004a3fd9315cf4704a76e6f2f1072
x-emc-delta: 27
x-emc-policy: default
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

## Example: Listing a Directory

A [Reading an Object](#) call on a directory returns a list of the directory's children (files and subdirectories).

### Request

```
GET /rest/namespace/mydirectory HTTP/1.1
accept: */*
date: Mon, 03 Aug 2009 13:33:38 GMT
content-type: application/octet-stream
x-emc-date: Mon, 03 Aug 2009 13:33:38 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: 6owPphyncgDRLkpZ8okLerzabzM=
```

### Response

```
HTTP/1.1 200 OK
Date: Mon, 03 Aug 2009 13:33:38 GMT
Server: Apache
x-emc-policy: _int
Content-Length: 327
x-emc-groupacl: other=NONE
x-emc-useracl: user1=FULL_CONTROL
x-emc-meta: atime=2009-08-03T13:30:13Z, mtime=2009-08-03T13:32:34Z,
ctime=2009-08-03T13:32:34Z, itime=2009-08-03T13:30:13Z, type=directory, uid=user1,
gid=apache, objectid=4a3fd8dfa2a8482004a3fd9315cf4704a76e665d80be,
objname=mydirectory, size=4096, nlink=1, policyname=default
Connection: close
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<ListDirectoryResponse xmlns='http://www.emc.com/cos/'>
  <DirectoryList>
    <DirectoryEntry>
      <ObjectID>4a3fd8dfa2a8482004a3fd9315cf4704a76e6f2f1072</ObjectID>
      <FileType>regular</FileType>
      <Filename>samplefile</Filename>
    </DirectoryEntry>
  </DirectoryList>
</ListDirectoryResponse>
```

## Example: Reading a File

To read a file, use the [Reading an Object](#) method, specifying the name of the file.

### Request

```
GET /rest/namespace/mydirectory/samplefile HTTP/1.1
accept: */*
date: Mon, 03 Aug 2009 13:34:38 GMT
content-type: application/octet-stream
x-emc-date: Mon, 03 Aug 2009 13:34:38 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: Tg2VUWnBQ9daW5OZafBoltBc7Vw=
```

**Response**

```
HTTP/1.1 200 OK
Date: Mon, 03 Aug 2009 13:34:38 GMT
Server: Apache
x-emc-policy: default
Content-Length: 27
x-emc-groupacl: other=NONE
x-emc-useracl: user1=FULL_CONTROL
x-emc-meta: atime=2009-08-03T13:32:35Z, mtime=2009-08-03T13:32:35Z,
ctime=2009-08-03T13:32:35Z, itime=2009-08-03T13:32:34Z, type=regular, uid=user1,
gid=apache, objectid=4a3fd8dfa2a8482004a3fd9315cf4704a76e6f2f1072,
objname=samplefile, size=27, nlink=1, policyname=default
Connection: close
Content-Type: application/octet-stream

content for this file
```

## Example: Reading Part of a File

To read part of a file, use the [Reading an Object](#) method with the Range request header. In the example below, the request is for 11 bytes (bytes 5-15).

**Request**

```
GET /rest/namespace/mydirectory/samplefile HTTP/1.1
accept: */*
date: Mon, 03 Aug 2009 13:35:11 GMT
content-type: application/octet-stream
x-emc-date: Mon, 03 Aug 2009 13:35:11 GMT
range: Bytes=5-15
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: vv7reSLatse4u7Wxo07FPSjJCpY=
```

**Response**

```
HTTP/1.1 206 Partial Content
Date: Mon, 03 Aug 2009 13:35:11 GMT
Server: Apache
x-emc-policy: default
Content-Length: 11
Content-Range: bytes 5-15/27
x-emc-groupacl: other=NONE
x-emc-useracl: user1=FULL_CONTROL
x-emc-meta: atime=2009-08-03T13:32:35Z, mtime=2009-08-03T13:32:35Z,
ctime=2009-08-03T13:32:35Z, itime=2009-08-03T13:32:34Z, type=regular, uid=user1,
gid=apache, objectid=4a3fd8dfa2a8482004a3fd9315cf4704a76e6f2f1072,
objname=samplefile, size=27, nlink=1, policyname=default
Connection: close
Content-Type: application/octet-stream

content for
```

## Example: Updating a file

### Request

```
PUT /rest/namespace/mydirectory/samplefile HTTP/1.1
accept: */*
date: Mon, 03 Aug 2009 13:36:41 GMT
content-type: application/octet-stream
x-emc-date: Mon, 03 Aug 2009 13:36:41 GMT
host: 168.159.116.96
content-length: 18
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: 0KL4MpDj/hI8ZGRnEOL2+1MdA5k=
content-md5: jv6qMXW6eTuumbd40ynJvQ==
```

different content

### Response

```
HTTP/1.1 200 OK
Date: Mon, 03 Aug 2009 13:36:41 GMT
Server: Apache
x-emc-delta: -9
x-emc-policy: default
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```



**PART II:**

**REST API**

# 2 Common REST Headers

The tables in this chapter show the common headers passed between the client and the server. Most object-service operations share common headers. For a list of the headers related to request and response transmissions, refer to each operation.

A request can have up to 100 HTTP headers, each up to 8kb.

There are separate tables for [“Standard HTTP Headers” on page 22](#) and [“Custom Headers” on page 24](#).

---

## Standard HTTP Headers

Header	Description
Content-Length:	The length of the request/response body, in bytes.
Content-Type:	Optional. Used to get and set the content type of the object. The default is <code>application/octet-stream</code> . Any value can be entered here, but only valid HTTP content types are understood when data is retrieved; e.g., by a browser.
Date : <i>date_in_UTC_format</i>	<p>(Optional: Date and/or <code>x-emc-date</code> must be in the request.)</p> <p>Date in UTC format, as defined in RFC 2616, section 3.3.1; e.g., Thu, 31 Jan 2008 19:37:28 GMT. Many HTTP clients set this header automatically.</p> <p>This date is used to check whether a request is valid within the Web server's validity time window. For this purpose, the timestamp in the <code>x-emc-date</code> header takes priority over this header. The Web server first checks for the <code>x-emc-date</code> header and uses its timestamp. If the <code>x-emc-date</code> header is not present, the Web server checks for the <code>Date</code> header and uses its timestamp.</p> <p>This date also is used for signature computation; see <a href="#">“REST Authentication: Algorithm for Securing REST Messages with Signatures” on page 119</a>.</p>

Header	Description
Expect	<p>Optional. May be used with the 100-continue expectation:</p> <p>Expect: 100-continue</p> <p>Sending this request header tells the server that the client will wait for a 100 Continue response before sending the request body. This is useful if you want the server to validate all request headers (including the signature), before the client sends data. This header may be used with POST and PUT methods, especially to create and update objects.</p>
Range: Bytes=begin_offset-end_offset	<p>When <i>updating</i> an object, you can update either the entire object or a single range of an object. To update a single range, use the Range header.</p> <p>For <i>reading</i> an object, byte ranges are implemented per the HTTP 1.1 specification. You may request the entire object, a single range of an object, or multiple ranges of an object. When multiple ranges are requested, a multipart message is returned. The multipart media type is “multipart/byteranges.”</p> <p>Range header formats:</p> <ul style="list-style-type: none"> <li>▶ <b>Format:</b> bytes=first-last  <b>Example:</b> range: bytes=10-20  <b>Description:</b> From first byte index to last byte index, inclusive.  <b>Use:</b> Reading or updating an object</li> <li>▶ <b>Format:</b> bytes=first-  <b>Example:</b> range: bytes=10-  <b>Description:</b> From first byte index until the end of the object (e.g., object size - 1).  <b>Use:</b> Reading an object</li> <li>▶ <b>Format:</b> bytes=-length  <b>Example:</b> range: bytes=-30  <b>Description:</b> The last length bytes.  <b>Use:</b> Reading an object</li> </ul> <p>For details, see the HTTP 1.1 specification (<a href="http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.35">http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.35</a>).</p>

---

## Custom Headers

---

Header	Description
<code>x-emc-date : date_in_UTC_format</code>	<p>(Optional: Date and/or <code>x-emc-date</code> must be in the request.)</p> <p>Date in UTC format, as defined in RFC 2616, section 3.3.1; e.g., Thu, 31 Jan 2008 19:37:28 GMT. This is set by the user.</p> <p>This date is used to check whether a request is valid within the Web server's validity time window. For this purpose, the timestamp in this header takes priority over the standard <code>Date</code> header. The Web server first checks for the <code>x-emc-date</code> header and uses its timestamp. If the <code>x-emc-date</code> header is not present, the Web server checks for the <code>Date</code> header and uses its timestamp.</p> <p>This date also is used for signature computation; see <a href="#">“REST Authentication: Algorithm for Securing REST Messages with Signatures”</a> on page 119.</p> <p>This header is provided because some development frameworks set the standard HTTP <code>Date</code> header automatically and do not allow the application developer to set it. In such cases, the developer can set and use this header for signature computation.</p>
<code>x-emc-delta:</code>	<p>Present only in responses from the server. The value of this header specifies the number of bytes by which the total disk space used by the user went up (positive number) or down (negative number) as a result of the operation.</p>
<code>x-emc-groupacl:other=permission</code>	<p>Sets the access rights to this object for the specified user group(s). Valid values are <code>READ</code>, <code>WRITE</code>, <code>NONE</code>, and <code>FULL_CONTROL</code>; these are not case sensitive when specified but always are returned in uppercase. Only the <code>other</code> group is supported; this applies to everyone other than the object owner.</p>
<code>x-emc-include-meta</code>	<p>(Optional) This header can be used only in requests to list objects. If the header's value is 0 or it is omitted, the list-objects request returns only object IDs. If the header's value is 1, the list-objects request returns an object list with system and user metadata.</p>

---

Header	Description
<code>x-emc-limit</code>	<p>(Optional) This header can be used only in list-objects requests. It specifies the maximum number of items that should be returned. (The response may include fewer items.) If this is 0 or not specified, there is no limit.</p> <p>Also see <a href="#">x-emc-token</a>.</p>
<code>x-emc-listable-meta:</code> <code>tag_name1=value1</code> <code>[,tag_name2=value2...]</code>	<p>Used in requests to set listable metadata tags for an object and their values. This header can be used in requests for creating an object, updating an object, and setting user metadata and responses for getting user metadata and reading an object. There can be only one of these headers per request, with up to 127 comma-separated, name-value pairs.</p> <p>Metadata names and values sent through the REST interface can use any characters from the iso-8859-1 character set.</p>
<code>x-emc-listable-tags: tag_name1</code> <code>[,tag_name2...]</code>	<p>Used in responses to return listable metadata tags for an object (which are set with the <code>x-emc-listable-meta</code> header).</p> <p><b>SPECIAL CHARACTERS:</b> If a metadata tag name contains a character that is not in the iso-8859-1 character set, that character is replaced with a question mark (?) character for display purposes. For example, consider a metadata tag name <math>\beta</math>eta (containing the Greek letter Beta). The Beta character may not be sent as a HTTP header, so it is replaced in the returned list as follows:</p> <p><code>x-emc-tags: mykey1, mykey2, ?eta</code></p>
<code>x-emc-meta: tag_name1=value1</code> <code>[,tag_name2=value2...]</code>	<p>Used in requests and responses, to set and get non-listable metadata tags for an object and their values. There can be only one of these headers per request, with up to 128 comma-separated, name-value pairs. Also used in the response for getting system metadata, to list system metadata values.</p> <p>Metadata names and values sent through the REST interface can use any characters from the iso-8859-1 character set.</p>

Header	Description
x-emc-policy	<p>Used in all responses, though the value depends on the type of request:</p> <ul style="list-style-type: none"> <li>▶ For requests that deal with the actual content of an object (e.g., creating, deleting, reading, and versioning an object), the value of <code>x-emc-policy</code> is the name of the policy which was applied to this object.</li> <li>▶ For other operations (e.g., metadata or ACL operations), the value of <code>x-emc-policy</code> is set to the reserved word <code>_int</code>.</li> </ul>
x-emc-signature: <i>signature</i>	<p>The signature provides a means for the system to authenticate the UID making the request. See <a href="#">“REST Authentication: Algorithm for Securing REST Messages with Signatures” on page 119</a> for details on constructing this header.</p>
x-emc-system-tags	<p>(Optional) This header can be used only in requests to list objects. With it, you can specify selected system metadata tags to be returned as <i>key/value</i> pairs for each object that is retrieved.</p>
x-emc-tags: <i>tag_name1</i> [, <i>tag_name2</i> ...]	<p>Used in responses, to get non-listable metadata tags for an object (which are set with the <code>x-emc-meta</code> header). Also used in requests for both user metadata and system metadata. There can be only one of these headers per request or response. For an example, see <a href="#">“Listing Objects” on page 46</a>.</p> <p>Some operations accept only one tag name; others accept multiple tag names, separated by commas. For correct usage, see the documentation for a specific operation.</p> <p>For limits on the character set, see <a href="#">“Special characters” on page 25</a>.</p>

Header	Description
x-emc-token	<p>(Optional) This can appear in list-objects requests or responses, in conjunction with <a href="#">x-emc-limit</a>.</p> <p>If <a href="#">x-emc-limit</a> is specified in a list-objects request, and more than the specified number of items exists, the response will include <a href="#">x-emc-token</a>. When <a href="#">x-emc-token</a> appears in a list-objects response, it is an indication that more data exists than was returned in the response and specifies an identifier for the next item to be retrieved. That identifier can then be used in a subsequent list-objects request, to specify the item at which data retrieval should begin.</p> <p>If <a href="#">x-emc-token</a> is specified and <a href="#">x-emc-limit</a> is 0, all objects from that point on are retrieved.</p> <p><b>NOTE:</b> The token just maintains state and should not be interpreted.</p>
x-emc-uid: <i>subtenant_id/user_id</i>	<p><i>user_id</i> is the UID of an application that is using the API, and <i>subtenant_id</i> is the ID of the subtenant to which <i>user_id</i> belongs.</p> <p>If the subtenant ID is missing, the ID that is used is that of the default subtenant for the tenant who has access to the node to which you are connecting. Only one UID is allowed per request. The shared secret associated with this UID is used for signature generation; see <a href="#">“Managing Authentication” on page 118</a>.</p>
x-emc-unencodable-meta: <i>tag_name1</i> [, <i>tag_name2...</i> ]	<p>Occurs only in responses. Specifies a list of metadata tags that have names and/or values that are unencodable for REST.</p> <p>This can occur if a metadata name/value pair is created or updated in SOAP, using characters in the Unicode/UTF-8 character set (supported by SOAP) which are outside the iso-8859-1 character set (supported by REST).</p> <p>For more on limits on the character set, see <a href="#">“Special characters” on page 25</a>.</p>
x-emc-user-tags	<p>(Optional) This header can be used only in requests to list objects. With it, you can specify selected user metadata tags to be returned as key/value pairs for each object that is retrieved.</p>

---

Header	Description
<code>x-emc-useracl: uid1=permission [,uid2=permission...]</code>	<p>Sets the access rights to this object for the specified UID(s). Valid values are READ, WRITE, NONE, and FULL_CONTROL; these are not case sensitive when specified but always are returned in uppercase.</p> <p>The UID must belong to the same subtenant to which the requesting UID belongs. A UID created under a different subtenant cannot access objects owned by the authenticating subtenant.</p>

---

# 3 REST API Reference

This chapter describes how the various operations that act on object and metadata are mapped to HTTP methods.

Atmos implements a standard REST interface to the Web service. The endpoint URL for the REST interface is `http://dns_name/rest`, with a suffix URI that describes the operation path.

Below, [“Specifying Objects/Files in REST Commands” on page 29](#) provides information about specifying the URI for object or namespace access, and [“REST Commands” on page 30](#) lists the methods in the REST API (along with the URIs). The remainder of this chapter comprises complete descriptions of each command.

---

## Specifying Objects/Files in REST Commands

This section lists the operations and related HTTP methods that act on data and on metadata.

For deleting, updating, reading, and versioning, you must include the object ID (if you use the object interface) or filename (if you use the namespace interface).

When you create an object via Web services (using either the object or namespace interface), you specify a UID. Within the Atmos file system, the object you create is assigned a file-system UID and a default GID (group ID), where the UID is identical to the UID you specified in your create operation. Permissions must be set properly on the authentication system of the file-system mounting host, to ensure that objects created via Web services are accessible from the file-system interface. Failure to set permissions properly may result in an access error when attempting to retrieve a file.

### **Namespace Access**

Atmos Web services allow you to assign a filename to an object when creating the object. This enables clients to use their own name when referring to an object (filename), rather than an object ID that Atmos assigns to the object.

In the REST API, there are two different URL endpoints to access the object and namespace interfaces:

Object:            `/rest/objects`  
Namespace:        `/rest/namespace`

For namespace access, a filename or directory name is sufficient; optionally, a full pathname (for either a file or directory) can be specified. In a create operation, if the pathname contains nonexistent directories, they are created automatically. The ACL specified in the request is applied to all newly created objects (files or directories). The metadata specified in the request is applied only to the leaf object (a file or directory).

The same set of operations is used to create, read, update, and delete both files and directories. When dealing with directories, however, there are two extra considerations:

- ▶ When creating a directory, the specified directory name must end with a forward slash (/):

`/rest/namespace/directory_name/`

For other operations, the forward slash can be used and is correct, but if it is omitted, the system figures it out automatically.

- ▶ There should be no payload in the request. If there is a payload, it is ignored.

**NOTE:** An object can be modified or retrieved via the namespace interface only if it was created via the namespace interface. If it was created with the object interface, it is impossible to assign a filename to it later.

## REST Commands

There are several types of methods:

- ▶ POST methods enable creating objects, creating versions of existing objects, and setting user metadata and ACLs for specified objects.
- ▶ GET methods retrieve object data, including metadata and ACLs.
- ▶ There is a HEAD method corresponding to each GET method. A HEAD request looks exactly like a GET request, except the method name is HEAD instead of GET. The response from the server is different with a HEAD method: there is no response body, only headers are returned. This is especially useful for `ReadObject` requests when one wants to retrieve the object's user metadata, system metadata, and access-control list but not the object itself.
- ▶ PUT methods update object attributes.
- ▶ DELETE methods remove objects and metadata from the system.

In the following table, the entries in the **URI** column is prefixed by `http://dns_name/rest`. The *pathname* variable is the full pathname of a file or directory.

HTTP Method	Operation	URI
<b>Data-management Operations</b>		
POST	Creating an Object	<code>/objects</code> — OR — <code>/namespace/pathname</code>
	Setting an ACL	<code>/objects/objectID?acl</code> — OR — <code>/namespace/pathname?acl</code>
	Versioning an Object	<code>/objects/objectID</code> — OR — <code>/namespace/pathname</code>

HTTP Method	Operation	URI
GET/ HEAD	Getting an ACL	<i>/objects/objectID?acl</i> — OR — <i>/namespace/pathname?acl</i>
	Getting Object Info	<i>/objects/objectid?info</i>  — OR —  <i>/namespace/pathname/myfile?info</i>
	Listing Objects	<i>/objects</i>
	Listing Versions	<i>/objects/objectID?versions</i>
	Reading an Object	<i>/objects/objectID</i> — OR — <i>/namespace/pathname</i>
	PUT	Updating an Object
DELETE	Deleting an Object	<i>/objects/objectID</i> — OR — <i>/namespace/pathname</i>
<b>Metadata-management Operations</b>		
POST	Setting User Metadata	<i>/objects/objectID?metadata/user</i> — OR — <i>/namespace/pathname?metadata/user</i>
GET/HEAD	Getting Listable Tags	<i>/objects?listabletags</i> — OR — <i>/namespace/pathname?listabletags</i>
	Getting System Metadata	<i>/objects/objectID?metadata/system</i> — OR — <i>/namespace/pathname?metadata/system</i>
	Getting User Metadata	<i>/objects/objectID?metadata/user</i> — OR — <i>/namespace/pathname?metadata/user</i>
	Listing User Metadata Tags	<i>/objects/objectID?metadata/tags</i> — OR — <i>/namespace/pathname?metadata/tags</i>
DELETE	Deleting User Metadata	<i>/objects/objectID?metadata/user</i> — OR — <i>/namespace/pathname?metadata/user</i>
<b>Service Operation</b>		
GET/HEAD	Getting Service Information	<i>/rest/service</i>

---

## Creating an Object

This operation creates an object with optional ACL and user metadata. Once created, the object can be increased to any size; see [“Updating an Object” on page 62](#). No validation is done on the metadata. The header content length is required to represent the size of the contents of the HTTP body, or the service will return an error. Appropriate system metadata is generated automatically.

This operation also can be used to create directories. Directories can be created implicitly or explicitly:

- ▶ Implicitly — Specify the full path for an object, and one or more new directories are created automatically as needed, before creating the object itself.
- ▶ Explicitly — End the directory name with a forward slash (/). The request body must be empty.

**NOTE:** *Directly under the / directory, you can create only directories, not files.*

### Object Interface

#### Request

```
POST /rest/objects HTTP/1.1
x-emc-listable-meta: part4/part7/part8=quick
x-emc-meta: part1=buy
accept: */*
x-emc-useracl: john=FULL_CONTROL,mary=READ
date: Wed, 18 Feb 2009 16:03:52 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:03:52 GMT
x-emc-groupacl: other=NONE
host: 168.159.116.96
content-length: 211
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: KpT+3Ini1W+CS6YwJEAwyWvIlIs=
```

#### Response

The response payload contains the `location` header, which specifies the newly created object ID returned as a URI.

```
HTTP/1.1 201 Created
Date: Wed, 18 Feb 2009 16:03:52 GMT
Server: Apache
location: /rest/objects/499ad542a1a8bc200499ad5a6b05580499c3168560a4
x-emc-delta: 211
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: default
```

## **Namespace Interface**

### **Request**

```
POST /rest/namespace/photos/mypicture.jpg HTTP/1.1
x-emc-listable-meta: part4/part7/part8=quick
x-emc-meta: part1=buy
accept: */*
x-emc-useracl: john=FULL_CONTROL,mary=READ
date: Wed, 18 Feb 2009 16:08:12 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:08:12 GMT
x-emc-groupacl: other=NONE
host: 168.159.116.96
content-length: 211
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: GTOC1GqFELjMMH9XIKvYRaHdyrk=
```

### **Response**

```
HTTP/1.1 201 Created
Date: Wed, 18 Feb 2009 16:08:12 GMT
Server: Apache
location: /rest/objects/499ad542a1a8bc200499ad5a6b05580499c326c2f984
x-emc-delta: 211
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: default
```

**NOTE:** Directly under the / directory, you can create only directories, not files. While creating a file, if you refer to a directory that does not exist, it is created automatically. In the example above, if the photos directory does not exist, it is created for you.

---

## Deleting an Object

This operation deletes the object that matches the ObjectID supplied in the URI. The operation also deletes associated metadata.

### **Object Interface**

#### **Request**

```
DELETE /rest/objects/499ad542a2a8bc200499ad5a7099940499c3e6fbcc3 HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:59:41 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:59:41 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: AHnsdoK6vmIEP8mt9708S8j7TKY=
```

#### **Response**

```
HTTP/1.1 204 No Content
Date: Wed, 18 Feb 2009 16:59:41 GMT
Server: Apache
x-emc-delta: -211
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: default
```

### **Namespace Interface**

#### **Request**

```
DELETE /rest/namespace/photos/myoldpicture.jpg HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 17:01:03 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 17:01:03 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: DEIYwSJWGxHD0wuC7xHYen5lDoA=
```

#### **Response**

```
HTTP/1.1 204 No Content
Date: Wed, 18 Feb 2009 17:01:04 GMT
Server: Apache
x-emc-delta: -211
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: default
```

---

## Deleting User Metadata

This operation deletes all user metadata (listable or non-listable) with matching tags from the specified object. Specify the tags of the pairs to be deleted. System metadata can be neither deleted nor modified directly. The request must include the `x-emc-tags` header, which should be a comma-separated list of tag names to delete; otherwise, the service returns an error.

### **Object Interface**

#### **Request**

```
DELETE /rest/objects/499ad542a1a8bc200499ad5a6b05580499c3168560a4?metadata/user HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 17:02:26 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 17:02:26 GMT
x-emc-tags: part1
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: KVD8OvoNsQbO001rWl64c/Pv5UY=
```

#### **Response**

```
HTTP/1.1 204 No Content
Date: Wed, 18 Feb 2009 17:02:26 GMT
Server: Apache
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

### **Namespace Interface**

#### **Request**

```
DELETE /rest/namespace/photos/mypicture.jpg?metadata/user HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 17:02:53 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 17:02:53 GMT
x-emc-tags: part1
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: /5RU66MJp3xGXNeybI8gYoAmXlE=
```

#### **Response**

```
HTTP/1.1 204 No Content
Date: Wed, 18 Feb 2009 17:02:53 GMT
Server: Apache
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

---

## Getting an ACL

This operation returns the ACL details associated with the specified object ID.

### **Object Interface**

#### **Request**

```
GET /rest/objects/499ad542a1a8bc200499ad5a6b05580499c3168560a4?acl HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:33:09 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:33:09 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: s7965CmZ956v9KY8UHmaipS/c/E=
```

#### **Response**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:33:09 GMT
Server: Apache
x-emc-groupacl: other=NONE
x-emc-useracl: fred=FULL_CONTROL, john=FULL_CONTROL, mary=READ, user1=FULL_CONTROL
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

### **Namespace Interface**

#### **Request**

```
GET /rest/namespace/photos/mypicture.jpg?acl HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:33:44 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:33:44 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: 9Yp9xxo8yt2g6QdVE+CQN5NoEow=
```

#### **Response**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:33:44 GMT
Server: Apache
x-emc-groupacl: other=NONE
x-emc-useracl: fred=FULL_CONTROL, john=FULL_CONTROL, mary=READ, user1=FULL_CONTROL
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

---

## Getting Listable Tags

This operation returns all listable tags under the specified input tag. For example, if an object is indexed by `tag1` (in other words, `tag1` is a listable tag for the object), `tag1` is returned by this operation. If no tag is specified, it defaults to `root`, and all top-level tags are returned. Unlike all other operations, this operation is executed under the global namespace (not against an object).

### **Object Interface**

#### **Request**

```
GET /rest/objects?listabletags HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:35:01 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:35:01 GMT
x-emc-tags: part4
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: 1OoKOJo9xoheuY1TFhp0xOHlPks=
```

#### **Response**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:35:01 GMT
Server: Apache
x-emc-listable-tags: part7, part9
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

### **Namespace Interface**

#### **Request**

```
GET /rest/namespace?listabletags
accept: */*
date: Wed, 18 Feb 2009 16:35:01 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:35:01 GMT
x-emc-tags: part4
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: 1OoKOJo9xoheuY1TFhp0xOHlPks=
```

#### **Response**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:35:01 GMT
Server: Apache
x-emc-listable-tags: part7, part9
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

---

## Getting Object Info

This operations returns details about the replicas for an object. Performing this operation on a directory returns error code 1022 because directory's do not have storage.

### **Object Interface**

#### **Request**

```
> GET /rest/objects/4b00fffea12059c104b00ffca1f8e804b040c4d911c9?info HTTP/1.1
> Host: 10.32.89.193
> accept: */*
> date: Fri, 20 Nov 2009 05:47:29 GMT
> content-type: application/octet-stream
> x-emc-date: Fri, 20 Nov 2009 05:47:29 GMT
> x-emc-uid: e103f726a87d45abbd8d5f189a8cecfca/aaa
> x-emc-signature: u/kFWYGR2Uf1/xpIikY/nBAeFXg=
```

#### **Response**

```
< HTTP/1.1 200 OK
< Date: Fri, 20 Nov 2009 05:47:29 GMT
< Server: Apache
< x-emc-policy: _int
< Content-Length: 723
< Connection: close
< Content-Type: text/xml
```

#### **Response Body**

```
<?xml version='1.0' encoding='UTF-8'?>
<GetObjectInfoResponse xmlns='http://www.emc.com/cos/'>
  <objectId>4b00fffea12059c104b00ffca1f8e804b040c4d911c9</objectId>
  <selection></selection>
  <numReplicas>2</numReplicas>
  <replicas>
    <replica>
      <id>3</id>
      <type>sync</type>
      <current>true</current>
      <location>Boston</location>
      <storageType>Normal</storageType>
    </replica>
    <replica>
      <id>5</id>
      <type>sync</type>
      <current>true</current>
      <location>Boston</location>
      <storageType>Normal</storageType>
    </replica>
  </replicas>
  <retention>
    <enabled>false</enabled>
    <endAt></endAt>
  </retention>
  <expiration>
    <enabled>false</enabled>
    <endAt></endAt>
  </expiration>
</GetObjectInfoResponse>
```

## Namespace Interface

### Request

```
GET /rest/namespace/photos/mypicture.jpg?info HTTP/1.1
accept: */
date: Thu, 07 Jan 2010 15:33:00 GMT
content-type: application/octet-stream
x-emc-date: Thu, 07 Jan 2010 15:33:00 GMT
x-emc-uid: e2f3a3f5e3aa4a2d91f532415405d6d3/user1
x-emc-signature: HMcVH8Sf7ciX8qhRPjiSknC0doE=
```

### Response

```
HTTP/1.1 200 OK
Date: Thu, 07 Jan 2010 15:33:00 GMT
Server: Apache
x-emc-policy: _int
Content-Length: 729
Connection: close
Content-Type: text/xml
<?xml version='1.0' encoding='UTF-8'?>
<GetObjectInfoResponse xmlns='http://www.emc.com/cos/'>
  <objectId>4b4502a5a2a8482004b4503232663404b45fe98a5ec1</objectId>
  <selection></selection>
  <numReplicas>2</numReplicas>
  <replicas>
    <replica>
      <id>3</id>
      <type>sync</type>
      <current>true</current>
      <location>cambridge</location>
      <storageType>Normal</storageType>
    </replica>
    <replica>
      <id>5</id>
      <type>sync</type>
      <current>true</current>
      <location>cambridge</location>
      <storageType>Normal</storageType>
    </replica>
  </replicas>
  <retention>
    <enabled>>false</enabled>
    <endAt></endAt>
  </retention>
  <expiration>
    <enabled>>false</enabled>
    <endAt></endAt>
  </expiration>
</GetObjectInfoResponse>
```

## Response XML Elements

XML element	Description
objectId	String. The object's unique identifier.
selection	String. The replica selection for read access. Values can be geographic or random.
numReplicas	Integer. The total number of replicas for this object.
Replicas	Container for set of replica definitions.

<b>XML element</b>	<b>Description</b>
Replica	Container for a replica instance.
replica ID	String. The unique identifier for the replica instance.
type	String. The replica type. Values can be sync or async.
current	Boolean. True if the replica is current, or False if the replica is not current.
location	String. The replica location.
storage type	String. The replica's storage type. Values can be stripe, normal, cloud, compression, and dedup.
retention	Container element for retention values.
enabled	A Boolean value (true/false) that defines whether retention is enabled for the replica.
endAt	<p>When enabled is true, specifies the dateTime when the data retention period expires. When enabled is false, this element is empty.</p> <p>dateTime has this format:</p> <ul style="list-style-type: none"> <li>▶ YYYY— year</li> <li>▶ MM—month</li> <li>▶ DD — day</li> <li>▶ hh — hour</li> <li>▶ mm — minute</li> <li>▶ ss — second</li> </ul>
expiration	Container element for expiration values.

---

<b>XML element</b>	<b>Description</b>
enabled	A Boolean value that specifies if expiration is enabled (true) or not (false)
endAt	<p>When enabled is true, specifies the dateTime at when the deletion expiration ends. When enabled is false, this element is empty.</p> <p>dateTime has this format:</p> <ul style="list-style-type: none"><li>▶ YYYY— year</li><li>▶ MM—month</li><li>▶ DD — day</li><li>▶ hh — hour</li><li>▶ mm — minute</li><li>▶ ss — second</li></ul>

---

---

## Getting Service Information

This operation returns information about the Atmos service; currently, the version of Atmos software in use. The Atmos version is in the following form:

*major.minor.patch*

For example:

1.2.4

### Request

```
GET /rest/service HTTP/1.1
accept: */*
date: Wed, 01 Jul 2009 16:18:16 GMT
x-emc-date: Wed, 01 Jul 2009 16:18:16 GMT
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: RhsBAyHYFYiBj46KSFntrSgkHcs=
```

### Response

```
HTTP/1.1 200 OK
Date: Wed, 01 Jul 2009 16:18:16 GMT
Server: Apache
Content-Length: 138
Connection: close
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<Service xmlns='http://www.emc.com/cos/'>
  <Version>
    <Atmos>1.2.4</Atmos>
  </Version>
</Service>
```

### Response schema

```
<xsd:complexType name="Version">
  <xsd:sequence>
    <xsd:element name="Atmos" type="xsd:string" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="Service">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Version" type="tns:Version" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

---

## Getting System Metadata

This operation returns the system metadata for the object. Specify the tags of the pairs to be returned as comma-separated entries for the `x-emc-tags` header; the absence of that header returns all pairs.

For a list of the system metadata that you can request from the Web service, see [“System Metadata” on page 12](#).

### Object Interface

In the following example, the `x-emc-tags` header is omitted, so all system-metadata pairs are returned (in the `x-emc-meta` header). In the response, `objname` is blank because this object does not have a name.

#### Request 1

```
GET /rest/objects/499ad542a1a8bc200499ad5a6b05580499c3168560a4?metadata/system HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:36:18 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:36:18 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: 2FqzIvlzmGahV6/4KUWzBANkrFc=
```

#### Response 1

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:36:18 GMT
Server: Apache
x-emc-meta: atime=2009-02-18T16:27:24Z, mtime=2009-02-18T16:03:52Z,
ctime=2009-02-18T16:27:24Z, itime=2009-02-18T16:03:52Z, type=regular, uid=user1,
gid=apache, objectid=499ad542a1a8bc200499ad5a6b05580499c3168560a4, objname=,
size=211, nlink=0, policyname=default
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

In the following example, the `x-emc-tags` header is used, so only specified system-metadata pairs are returned.

#### Request 2

```
GET /rest/objects/499ad542a1a8bc200499ad5a6b05580499c3168560a4?metadata/system HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:36:18 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:36:18 GMT
x-emc-tags: atime,uid
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: 2FqzIvlzmGahV6/4KUWzBANkrFc=
```

#### Response 2

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:36:18 GMT
Server: Apache
x-emc-meta: atime=2009-02-18T16:27:24Z, uid=user1
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

## ***Namespace Interface***

### **Request**

```
GET /rest/namespace/photos/mypicture.jpg?metadata/system HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:37:00 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:37:00 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: WFcvgGUu+CzqkGd07EGTiEdb7bs=
```

### **Response**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:37:00 GMT
Server: Apache
x-emc-meta: atime=2009-02-18T16:28:03Z, mtime=2009-02-18T16:08:12Z,
ctime=2009-02-18T16:28:03Z, itime=2009-02-18T16:08:12Z, type=regular, uid=user1,
gid=apache, objectid=499ad542a1a8bc200499ad5a6b05580499c326c2f984,
objname=mypicture.jpg, size=211, nlink=1, policyname=default
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

---

## Getting User Metadata

This operation returns the metadata associated with the specified object, as a comma-separated list of name-value pairs. Specify the tags of the pairs to be returned as comma-separated tag names. To get all pairs, omit the tag names. Regular (non-listable) metadata is returned using the `x-emc-meta` header; listable metadata, the `x-emc-listable-meta` header.

### **Object Interface**

#### **Request**

```
GET /rest/objects/499ad542a1a8bc200499ad5a6b05580499c3168560a4?metadata/user HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:38:06 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:38:06 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: hXmB2b/zdW3tp7qCbSM+SQRMaM4=
```

#### **Response**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:38:06 GMT
Server: Apache
x-emc-listable-meta: part4/part7/part8=quick, part3=fast
x-emc-meta: part1=order
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

### **Namespace Interface**

#### **Request**

```
GET /rest/namespace/photos/mypicture.jpg?metadata/user HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:38:14 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:38:14 GMT
host: 168.159.116.96:8080
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: jhqNQwPrKjc9RpjKmops3fKw+l8=
```

#### **Response**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:38:14 GMT
Server: Apache
x-emc-listable-meta: part4/part7/part8=quick, part3=fast
x-emc-meta: part1=order
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

---

## Listing Objects

This operation retrieves all object IDs indexed by a given tag. To specify the tag, use the `x-emc-tags` header; only one tag name/hierarchy may be included. As mentioned earlier, listable tags are created in a user's own namespace; hence, they are private to that user. Only objects belonging to the requesting UID are returned.

### **Object Interface—without Metadata**

#### **Request**

The `x-emc-include-meta` header, set to 0, indicates that only object IDs should be returned.

```
GET /rest/objects HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:39:49 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:39:49 GMT
x-emc-tags: part4/part7/part8
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: Z1lFtIyYe6kvqibS9eqcIBpiQ7I=
x-emc-include-meta: 0
```

#### **Response**

The response contains an XML payload listing the object IDs for this user. Object IDs are 44 characters long, and there is no limit to how many objects you can store; therefore, it is possible to reach the limit for data in the HTTP header. As a result, the Web service returns the object IDs from a `list-objects` operation into the XML body, not the header.

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:39:49 GMT
Server: Apache
Content-Length: 359
Connection: close
Content-Type: text/xml
x-emc-policy: _int

<?xml version='1.0' encoding='UTF-8'?>
<ListObjectsResponse xmlns='http://www.emc.com/cos/'>
  <Object>
    <ObjectID>499ad542a2a8bc200499ad5a7099940499b44f51e97d</ObjectID>
  </Object>
  <Object>
    <ObjectID>499ad542a1a8bc200499ad5a6b05580499b44f5aff04</ObjectID>
  </Object>
  <Object>
    <ObjectID>499ad542a2a8bc200499ad5a7099940499b44f779a54</ObjectID>
  </Object>
</ListObjectsResponse>
```

### **Object Interface—without Metadata and Using the x-emc-limit Header**

Below, in each request, the user requests up to 50 objects. The first request does not include an `x-emc-token` identifier, so data retrieval starts with the first object available. In the first response, objects 1-50 are returned, along with an `x-emc-token` identifier. That identifier is specified in the second request, as the starting point for data retrieval. In the second response, objects 51-100 are returned, along with another `x-emc-token` identifier. That second identifier is specified in the third request, as the starting point for data retrieval. In the third response, the final 25 objects are returned. This final response does not include an `x-emc-token` identifier, because there are no more objects to be retrieved.

#### **Request 1**

```
GET /rest/objects HTTP/1.1
accept: */*
x-emc-limit: 50
date: Fri, 15 May 2009 14:50:13 GMT
content-type: application/octet-stream
x-emc-date: Fri, 15 May 2009 14:50:13 GMT
x-emc-tags: part1
host: 127.0.0.1
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: v+OUztaBdCqIPO/0p/FyXnosHXC=
x-emc-include-meta: 0
```

#### **Response 1**

```
HTTP/1.1 200 OK
Date: Fri, 15 May 2009 14:50:13 GMT
Server: Apache
x-emc-token: 4a0d6e22a2a8482004a0d6ecd85daf04a0d733b28892
Content-Length: 332
Connection: close
Content-Type: text/xml
x-emc-policy: _int

<?xml version='1.0' encoding='UTF-8'?>
<ListObjectsResponse xmlns='http://www.emc.com/cos/'>
  <Object>
    <ObjectID>4a0d6e22a1a8482004a0d6ecd1247804a0d7337c89fd</ObjectID>
  </Object>
  <Object>
    <ObjectID>4a0d6e22a2a8482004a0d6ecd85daf04a0d733902a93</ObjectID>
  </Object>
  ...
  <Object>
    <ObjectID>n</ObjectID>
  </Object>
</ListObjectsResponse>
```

#### **Request 2**

```
GET /rest/objects HTTP/1.1
x-emc-token: 4a0d6e22a2a8482004a0d6ecd85daf04a0d733b28892
accept: */*
x-emc-limit: 50
date: Fri, 15 May 2009 14:50:39 GMT
content-type: application/octet-stream
x-emc-date: Fri, 15 May 2009 14:50:39 GMT
x-emc-tags: part1
host: 127.0.0.1
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: ozaUkr9upED4iktYlu6KQWgH+v0=
x-emc-include-meta: 0
```

**Response 2**

```

HTTP/1.1 200 OK
Date: Fri, 15 May 2009 14:50:39 GMT
Server: Apache
x-emc-token: 4a0d6e22a2a8482004a0d6ecd85daf04a0d733df3eea
Content-Length: 332
Connection: close
Content-Type: text/xml
x-emc-policy: _int

<?xml version='1.0' encoding='UTF-8'?>
<ListObjectsResponse xmlns='http://www.emc.com/cos/'>
  <Object>
    <ObjectID>4a0d6e22a2a8482004a0d6ecd85daf04a0d733b28892</ObjectID>
  </Object>
  <Object>
    <ObjectID>4a0d6e22a1a8482004a0d6ecd1247804a0d733c19b14</ObjectID>
  </Object>
  ...
  <Object>
    <ObjectID>n</ObjectID>
  </Object>
</ListObjectsResponse>

```

**Request 3**

```

GET /rest/objects HTTP/1.1
x-emc-token: 4a0d6e22a2a8482004a0d6ecd85daf04a0d733df3eea
accept: */*
x-emc-limit: 50
date: Fri, 15 May 2009 14:50:56 GMT
content-type: application/octet-stream
x-emc-date: Fri, 15 May 2009 14:50:56 GMT
x-emc-tags: part1
host: 127.0.0.1
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: 12i2hiJdtosuJsNei2y6BtwN+t4=
x-emc-include-meta: 0

```

**Response 3**

```

HTTP/1.1 200 OK
Date: Fri, 15 May 2009 14:50:56 GMT
Server: Apache
Content-Length: 332
Connection: close
Content-Type: text/xml
x-emc-policy: _int

<?xml version='1.0' encoding='UTF-8'?>
<ListObjectsResponse xmlns='http://www.emc.com/cos/'>
  <Object>
    <ObjectID>4a0d6e22a2a8482004a0d6ecd85daf04a0d733df3eea</ObjectID>
  </Object>
  <Object>
    <ObjectID>4a0d6e22a2a8482004a0d6ecd85daf04a0d733eb2d85</ObjectID>
  </Object>
  ...
  <Object>
    <ObjectID>n</ObjectID>
  </Object>
</ListObjectsResponse>

```

## Object Interface—with All Metadata

### Request

The `x-emc-include-meta` header, set to 1, indicates that an object list should be returned with *all* system and user metadata for each object.

```
GET /rest/objects HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:41:02 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:41:02 GMT
x-emc-tags: part4/part7/part8
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: hEf+WgX/0HLo6zoQKalo6sB/kt0=
x-emc-include-meta: 1
```

### Response

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:41:02 GMT
Server: Apache
Connection: close
Transfer-Encoding: chunked
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int

<?xml version='1.0' encoding='UTF-8'?>
<ListObjectsResponse xmlns='http://www.emc.com/cos/'>
  <Object>
    <ObjectID>499ad542a2a8bc200499ad5a7099940499b44f51e97d</ObjectID>
    <SystemMetadataList>
      <Metadata>
        <Name>atime</Name>
        <Value>2009-02-17T23:15:01Z</Value>
      </Metadata>
      <Metadata>
        <Name>mtime</Name>
        <Value>2009-02-17T23:15:01Z</Value>
      </Metadata>
      <Metadata>
        <Name>ctime</Name>
        <Value>2009-02-17T23:15:01Z</Value>
      </Metadata>
      <Metadata>
        <Name>itime</Name>
        <Value>2009-02-17T23:15:01Z</Value>
      </Metadata>
      <Metadata>
        <Name>type</Name>
        <Value>regular</Value>
      </Metadata>
      <Metadata>
        <Name>uid</Name>
        <Value>user1</Value>
      </Metadata>
      <Metadata>
        <Name>gid</Name>
        <Value>apache</Value>
      </Metadata>
      <Metadata>
        <Name>objectid</Name>
        <Value>499ad542a2a8bc200499ad5a7099940499b44f51e97d</Value>
      </Metadata>
```

```

    <Metadata>
      <Name>objname</Name>
      <Value></Value>
    </Metadata>
    <Metadata>
      <Name>size</Name>
      <Value>7589</Value>
    </Metadata>
    <Metadata>
      <Name>nlink</Name>
      <Value>0</Value>
    </Metadata>
    <Metadata>
      <Name>polycyname</Name>
      <Value>default</Value>
    </Metadata>
  </SystemMetadataList>
  <UserMetadataList>
    <Metadata>
      <Name>part1</Name>
      <Value>order</Value>
      <Listable>>false</Listable>
    </Metadata>
    <Metadata>
      <Name>part4/part7/part8</Name>
      <Value>quick</Value>
      <Listable>>true</Listable>
    </Metadata>
  </UserMetadataList>
</Object>
...
</ListObjectsResponse>

```

### **Object Interface—with Selected Metadata**

Instead of getting *all* system and user metadata key/values for each object, you can specify selected ones you to retrieve. To do this, use the `x-emc-system-tags` and `x-emc-user-tags` headers.

#### **Request**

```

GET /rest/objects HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:41:02 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:41:02 GMT
x-emc-tags: part4/part7/part8
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: hEf+WgX/0HLo6zoQKalo6sB/kt0=
x-emc-system-tags: atime,size
x-emc-user-tags: city

```

## Response

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:41:02 GMT
Server: Apache
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int

<?xml version='1.0' encoding='UTF-8'?>
<ListObjectsResponse xmlns='http://www.emc.com/cos/'>
  <Object>
    <ObjectID>499ad542a2a8bc200499ad5a7099940499b44f51e97d</ObjectID>
    <SystemMetadataList>
      <Metadata>
        <Name>atime</Name>
        <Value>2009-02-17T23:15:01Z</Value>
      </Metadata>
      <Metadata>
        <Name>size</Name>
        <Value>1234</Value>
      </Metadata>
    </SystemMetadataList>
    <UserMetadataList>
      <Metadata>
        <Name>city</Name>
        <Value>boston</Value>
        <Listable>>false</Listable>
      </Metadata>
    </UserMetadataList>
  </Object>
</ListObjectsResponse>
```

---

## Listing User Metadata Tags

This operation returns the user-defined metadata tags assigned to the object. Regular metadata is returned using the `x-emc-tags` header, and listable metadata is returned using the `x-emc-listable-tags` header.

### **Object Interface**

#### **Request**

```
GET /rest/objects/499ad542a1a8bc200499ad5a6b05580499c3168560a4?metadata/tags HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:45:53 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:45:53 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: 2r9FsrSP4UaXTyrTDjhvjQzFJzs=t
```

#### **Response**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:45:53 GMT
Server: Apache
x-emc-tags: part1
x-emc-listable-tags: part3, part4/part7/part8
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

### **Namespace Interface**

#### **Request**

```
GET /rest/namespace/photos/mypicture.jpg?metadata/tags HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:46:33 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:46:33 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: sbifTscR4YrTlkiQQVUSTc/lshC=
```

#### **Response**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:46:33 GMT
Server: Apache
x-emc-tags: part1
x-emc-listable-tags: part3, part4/part7/part8
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

---

## Listing Versions

This operation retrieves the object IDs of all objects in the versioning chain of the specified object, including itself. Versioned objects can also be used as the specified object to the list-versions operation.

### **Object Interface**

#### **Request**

```
GET /rest/objects/499ad542a1a8bc200499ad5a6b05580499c3168560a4?versions HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:47:25 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:47:25 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: eGZ2guZXorYnvv1ICHd/ZSy1bL4=
```

#### **Response**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:47:25 GMT
Server: Apache
Content-Length: 252
Connection: close
Content-Type: text/xml
x-emc-policy: _int

<?xml version='1.0' encoding='UTF-8'?>
<ListVersionsResponse xmlns='http://www.emc.com/cos/'>
  <ObjectID>499ad542a1a8bc200499ad5a6b05580499c3168560a4</ObjectID>
  <ObjectID>499ad542a1a8bc200499ad5a6b05580499c378e58163</ObjectID>
</ListVersionsResponse>
```

### **Namespace Interface**

#### **Request**

```
GET /rest/namespace/photos/mypicture.jpg?versions HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:48:23 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:48:23 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: nXuc5CET9AYb8+lTTNCoppWv3sg=
```

#### **Response**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:48:23 GMT
Server: Apache
Content-Length: 252
Connection: close
Content-Type: text/xml
x-emc-policy: _int

<?xml version='1.0' encoding='UTF-8'?>
<ListVersionsResponse xmlns='http://www.emc.com/cos/'>
  <ObjectID>499ad542a1a8bc200499ad5a6b05580499c326c2f984</ObjectID>
  <ObjectID>499ad542a1a8bc200499ad5a6b05580499c37984f123</ObjectID>
</ListVersionsResponse>
```

---

## Reading an Object

This operation returns the contents of the specified object, along with the associated user metadata, system metadata and access-control list.

An optional `Range` header (see [Chapter 2, “Common REST Headers” on page 22](#)) can be used to read only part of the object rather than the entire object. The value of the `Range` header should be the byte ranges to retrieve. The byte offsets are 0 based: 0 is the first byte, 1 is the second byte, and so on.

### **Object Interface**

#### **Request 1**

```
GET /rest/objects/499ad542a1a8bc200499ad5a6b05580499c3168560a4 HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:49:10 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:49:10 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: MwY3VpEBPkWz7/18CFmKQ/iYqA=
```

#### **Response 1**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:49:10 GMT
Server: Apache
Content-Length: 211
x-emc-groupacl: other=NONE
x-emc-useracl: fred=FULL_CONTROL, john=FULL_CONTROL, mary=READ, user1=FULL_CONTROL
x-emc-listable-meta: part4/part7/part8=quick, part3=fast
x-emc-meta: part1=order, atime=2009-02-18T16:27:24Z, mtime=2009-02-18T16:03:52Z,
ctime=2009-02-18T16:27:24Z, itime=2009-02-18T16:03:52Z, type=regular, uid=user1,
gid=apache, objectid=499ad542a1a8bc200499ad5a6b05580499c3168560a4, objname=,
size=211, nlink=0, policyname=default
Connection: close
Content-Type: application/octet-stream
x-emc-policy: default
```

#### **Request 2**

This request is for a directory that contains one file and one subdirectory.

```
GET /rest/objects/49a2b73da2a8bc20049a2b79d84405049a316695b311 HTTP/1.1
accept: */*
date: Tue, 24 Feb 2009 16:15:50 GMT
content-type: application/octet-stream
x-emc-date: Tue, 24 Feb 2009 16:15:50 GMT
host: 168.159.116.96:8080
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: p0OWEqTr2oUUz3xdzCbjoQk8+mE=
```

**Response 2**

```

HTTP/1.1 200 OK
Date: Tue, 24 Feb 2009 16:15:50 GMT
Server: Apache
Content-Length: 505
x-emc-groupacl: other=NONE
x-emc-useracl: user1=FULL_CONTROL
x-emc-meta: atime=2009-02-23T21:34:33Z, mtime=2009-02-23T21:34:33Z,
ctime=2009-02-23T21:34:33Z, itime=2009-02-23T21:34:33Z, type=directory, uid=user1,
gid=apache, objectid=49a2b73da2a8bc20049a2b79d84405049a316695b311,
objname=mydirectory, size=4096, nlink=1, policyname=default
Connection: close
Content-Type: text/xml
x-emc-policy: default

```

```

<?xml version='1.0' encoding='UTF-8'?>
<ListDirectoryResponse xmlns='http://www.emc.com/cos/'>
  <DirectoryList>
    <DirectoryEntry>
      <ObjectID>49a2b73da2a8bc20049a2b79d84405049a41b41ee06a</ObjectID>
      <FileType>directory</FileType>
      <Filename>mysubdirectory</Filename>
    </DirectoryEntry>
    <DirectoryEntry>
      <ObjectID>49a2b73da2a8bc20049a2b79d84405049a41b5091679</ObjectID>
      <FileType>regular</FileType>
      <Filename>myfile.txt</Filename>
    </DirectoryEntry>
  </DirectoryList>
</ListDirectoryResponse>

```

**Object Interface—With Range Header**

In this section, we use an example object that is 50 bytes long and has the following body:

```
the quick brown fox jumps right over the lazy dog
```

For brevity, all headers *not* dealing directly with ranges were removed.

Example 1 requests the entire object

**Request 1**

```
GET /rest/objects/4acbb971a1a8482004acbb9f355e3a04acf7e8ee8db1 HTTP/1.1
```

**Response 1**

```

HTTP/1.1 200 OK
Content-Length: 50

the quick brown fox jumps right over the lazy dog

```

Example 2 requests bytes 4-8.

**Request 2**

```
GET /rest/objects/4acbb971a1a8482004acbb9f355e3a04acf7e8ee8db1 HTTP/1.1
range: Bytes=4-8
```

**Response 2**

```

HTTP/1.1 206 Partial Content
Content-Range: bytes 4-8/50
Content-Length: 5

quick

```

Example 3 requests bytes 4-8 and 41-44.

**Request 3**

```
GET /rest/objects/4acbb971a1a8482004acbb9f355e3a04acf7e8ee8db1 HTTP/1.1
range: Bytes=4-8,41-44
```

**Response 3**

```
HTTP/1.1 206 Partial Content
Content-Length: 230
Content-Type: multipart/byteranges; boundary=bound04acf7f0ae3ccc

--bound04acf7f0ae3ccc
Content-Type: application/octet-stream
Content-Range: bytes 4-8/50

quick
--bound04acf7f0ae3ccc
Content-Type: application/octet-stream
Content-Range: bytes 41-44/50

lazy
--bound04acf7f0ae3ccc--
```

Example 4 requests from byte 32 until the end of the object.

**Request 4**

```
GET /rest/objects/4acbb971a1a8482004acbb9f355e3a04acf7e8ee8db1 HTTP/1.1
range: Bytes=32-
```

**Response 4**

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 32-49/50
Content-Length: 18
```

over the lazy dog

Example 5 requests the last 9 bytes.

**Request 5**

```
GET /rest/objects/4acbb971a1a8482004acbb9f355e3a04acf7e8ee8db1 HTTP/1.1
range: Bytes=-9
```

**Response 5**

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 41-49/50
Content-Length: 9
```

lazy dog

Example 6 requests bytes 4-8, from bytes 32 until the end of the object, and the last 9 bytes.

**Request 6**

```
GET /rest/objects/4acbb971a1a8482004acbb9f355e3a04acf7e8ee8db1 HTTP/1.1
range: Bytes=4-8,32-,-9
```

**Response 6**

```
HTTP/1.1 206 Partial Content
Content-Length: 351
Content-Type: multipart/byteranges; boundary=bound04acf7f8a23b49

--bound04acf7f8a23b49
Content-Type: application/octet-stream
Content-Range: bytes 4-8/50

quick
--bound04acf7f8a23b49
Content-Type: application/octet-stream
Content-Range: bytes 32-49/50

over the lazy dog

--bound04acf7f8a23b49
Content-Type: application/octet-stream
Content-Range: bytes 41-49/50

lazy dog

--bound04acf7f8a23b49--
```

Example 7 requests a range that is valid but not satisfiable.

**Request 7**

```
GET /rest/objects/4acbb971a1a8482004acbb9f355e3a04acf7e8ee8db1 HTTP/1.1
range: Bytes=1000-
```

**Response 7**

```
HTTP/1.1 416 Requested Range Not Satisfiable
Content-Length: 136
Content-Range: bytes */50
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<Error>
  <Code>1004</Code>
  <Message>The specified range cannot be satisfied.</Message>
</Error>
```

Example 8 requests one range that is not satisfiable and one range that is satisfiable.

**Request 8**

```
GET /rest/objects/4acbb971a1a8482004acbb9f355e3a04acf7e8ee8db1 HTTP/1.1
range: Bytes=1000-,4-8
```

**Response 8**

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 4-8/50
Content-Length: 5

quick
```

Example 9 requests an invalid byte range. The entire object is returned.

**Request 9** GET /rest/objects/4acbb971a1a8482004acbb9f355e3a04acf7e8ee8db1 HTTP/1.1  
range: Bytes=a-100

**Response 9** HTTP/1.1 200 OK  
Content-Length: 50

the quick brown fox jumps right over the lazy dog

### ***Namespace Interface***

**Request 1** GET /rest/namespace/photos/mypicture.jpg HTTP/1.1  
accept: \*/\*  
date: Wed, 18 Feb 2009 16:52:05 GMT  
content-type: application/octet-stream  
x-emc-date: Wed, 18 Feb 2009 16:52:05 GMT  
host: 168.159.116.96  
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1  
x-emc-signature: LYcvpkX1jpdguTf2VpO5Dkt4TM=

**Response 1** HTTP/1.1 200 OK  
Date: Wed, 18 Feb 2009 16:52:05 GMT  
Server: Apache  
Content-Length: 211  
x-emc-groupacl: other=NONE  
x-emc-useracl: fred=FULL\_CONTROL, john=FULL\_CONTROL, mary=READ, user1=FULL\_CONTROL  
x-emc-listable-meta: part4/part7/part8=quick, part3=fast  
x-emc-meta: part1=order, atime=2009-02-18T16:28:03Z, mtime=2009-02-18T16:08:12Z, ctime=2009-02-18T16:28:03Z, itime=2009-02-18T16:08:12Z, type=regular, uid=user1, gid=apache, objectid=499ad542a1a8bc200499ad5a6b05580499c326c2f984, objname=mypicture.jpg, size=211, nlink=1, policyname=default  
Connection: close  
Content-Type: application/octet-stream  
x-emc-policy: default

**Request 2** This request is for a directory that contains one file and one subdirectory.

GET /rest/namespace/photos/mydirectory HTTP/1.1  
accept: \*/\*  
date: Tue, 24 Feb 2009 16:16:17 GMT  
content-type: application/octet-stream  
x-emc-date: Tue, 24 Feb 2009 16:16:17 GMT  
host: 168.159.116.96:8080  
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1  
x-emc-signature: FcGSy/D7jyjjyifx2U/1yrO9Vfd8=

## Response 2

```
HTTP/1.1 200 OK
Date: Tue, 24 Feb 2009 16:16:17 GMT
Server: Apache
Content-Length: 505
x-emc-groupacl: other=
x-emc-useracl: user1=FULL_CONTROL
x-emc-meta: atime=2009-02-23T21:34:33Z, mtime=2009-02-23T21:34:33Z,
ctime=2009-02-23T21:34:33Z, itime=2009-02-23T21:34:33Z, type=directory, uid=user1,
gid=apache, objectid=49a2b73da2a8bc20049a2b79d84405049a316695b311,
objname=mydirectory, size=4096, nlink=1, policyname=default
Connection: close
Content-Type: text/xml
x-emc-policy: default
```

```
<?xml version='1.0' encoding='UTF-8'?>
<ListDirectoryResponse xmlns='http://www.emc.com/cos/'>
  <DirectoryList>
    <DirectoryEntry>
      <ObjectID>49a2b73da2a8bc20049a2b79d84405049a41b41ee06a</ObjectID>
      <FileType>directory</FileType>
      <Filename>mysubdirectory</Filename>
    </DirectoryEntry>
    <DirectoryEntry>
      <ObjectID>49a2b73da2a8bc20049a2b79d84405049a41b5091679</ObjectID>
      <FileType>regular</FileType>
      <Filename>myfile.txt</Filename>
    </DirectoryEntry>
  </DirectoryList>
</ListDirectoryResponse>
```

---

## Setting an ACL

Use this operation to set the access control for this object. The operation can be used for setting or resetting permissions. Either `x-emc-groupacl` or `x-emc-useracl` must be included, or the server will return an error.

### **Object Interface**

#### **Request**

```
POST /rest/objects/499ad542a1a8bc200499ad5a6b05580499c3168560a4?acl HTTP/1.1
accept: */*
x-emc-useracl: fred=FULL_CONTROL
date: Wed, 18 Feb 2009 16:21:00 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:21:00 GMT
x-emc-groupacl: other=NONE
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: nym3OK8krG6uDOpmomnsedRi8YY=
```

#### **Response**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:21:00 GMT
Server: Apache
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

### **Namespace Interface**

#### **Request**

```
POST /rest/namespace/photos/mypicture.jpg?acl HTTP/1.1
accept: */*
x-emc-useracl: fred=FULL_CONTROL
date: Wed, 18 Feb 2009 16:22:17 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:22:17 GMT
x-emc-groupacl: other=NONE
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: 93zwmHIQmn5wLxJUCZOcnobw/mY=
```

#### **Response**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:22:17 GMT
Server: Apache
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

---

## Setting User Metadata

The operation writes the metadata into the object. Either `x-emc-listable-meta` or `x-emc-meta` must be included in the request, or the server will return an error.

### **Object Interface**

**Request**

```
POST /rest/objects/499ad542a1a8bc200499ad5a6b05580499c3168560a4?metadata/user HTTP/1.1
x-emc-listable-meta: part3=fast
x-emc-meta: part1=order
accept: */*
date: Wed, 18 Feb 2009 16:27:24 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:27:24 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: OLI2TcDNWQ29gZv+ONrlufCKA9M=
```

**Response**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:27:24 GMT
Server: Apache
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

### **Namespace Interface**

**Request**

```
POST /rest/namespace/photos/mypicture.jpg?metadata/user HTTP/1.1
x-emc-listable-meta: part3=fast
x-emc-meta: part1=order
accept: */*
date: Wed, 18 Feb 2009 16:28:03 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:28:03 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: mfz9JwQU+7Wu5T2KFIiNZBetJ4g=
```

**Response**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:28:03 GMT
Server: Apache
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

---

## Updating an Object

This operation can be used for updating the contents of an object, including its metadata and ACLs. You can update just part of the object or the complete object. To update part of the object, use the `Range` header (see [Chapter 2, “Common REST Headers” on page 22](#)) to specify the beginning and ending offsets. This operation also can be used to add or modify existing metadata or ACLs; for example, metadata can be changed from listable to non-listable and vice versa.

You can use this operation to change the size of an object; do one of the following:

- ▶ To truncate an object to `size=0`, omit the `Range` header, and specify an empty request body. Truncating an object to `size=0` leaves the object ID unchanged.
- ▶ To overwrite an object, omit the `Range` header and attach the new object content to the request body.
- ▶ To append to an object, specify the `Range` header with:

```
beginOffset=currentSizeOfTheObject
endOffset=newSizeOfTheObject - 1
```

and attach the data corresponding to the content increase to the request body.

### **Object Interface**

#### **Request**

```
PUT /rest/objects/499ad542a1a8bc200499ad5a6b05580499c3168560a4 HTTP/1.1
x-emc-listable-meta: part4/part9=slow
x-emc-meta: part2=here
accept: */*
x-emc-useracl: john=WRITE
date: Wed, 18 Feb 2009 16:56:31 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:56:31 GMT
range: Bytes=10-18
host: 168.159.116.96
content-length: 9
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: opW4gNiT+MiOt/w7IxGgIeP6B+Q=
```

#### **Response**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:56:31 GMT
Server: Apache
x-emc-delta: 0
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: default
```

## ***Namespace Interface***

### **Request**

```
PUT /rest/namespace/photos/mypicture.jpg HTTP/1.1
x-emc-listable-meta: part4/part9=slow
x-emc-meta: part2=here
accept: */*
x-emc-useracl: john=WRITE
date: Wed, 18 Feb 2009 16:58:06 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:58:06 GMT
range: Bytes=10-18
host: 168.159.116.96
content-length: 9
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: Z5Sl6Pyeu0ehqcyXx7TZgfifle8o=
```

### **Response**

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2009 16:58:06 GMT
Server: Apache
x-emc-delta: 0
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: default
```

---

## Versioning an Object

This operation creates a fixed (immutable) version of the object.

It is supported only for objects, not for directories or the namespace interface.

**NOTE:** User metadata is copied from the specified object to its new version.

### **Object Interface**

#### **Request**

```
POST /rest/objects/499ad542a1a8bc200499ad5a6b05580499c3168560a4?versions HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:30:06 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:30:06 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: LALLRwk3t1Q5su+xG1P6hjJVFXQ=
```

#### **Response**

The response payload contains the `location` header, which specifies the newly created object ID returned as a URI.

```
HTTP/1.1 201 Created
Date: Wed, 18 Feb 2009 16:30:06 GMT
Server: Apache
location: /rest/objects/499ad542a1a8bc200499ad5a6b05580499c378e58163
x-emc-delta: 211
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: default
```

### **Namespace Interface**

#### **Request**

```
POST /rest/namespace/photos/mypicture.jpg?versions HTTP/1.1
accept: */*
date: Wed, 18 Feb 2009 16:30:16 GMT
content-type: application/octet-stream
x-emc-date: Wed, 18 Feb 2009 16:30:16 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: 3yr7jnlAUAQRS8LVhOhPTGhUcJY=
```

#### **Response**

```
HTTP/1.1 201 Created
Date: Wed, 18 Feb 2009 16:30:16 GMT
Server: Apache
location: /rest/objects/499ad542a1a8bc200499ad5a6b05580499c37984f123
x-emc-delta: 211
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: default
```



**PART III:**

**SOAP API**

# 4 SOAP WSDL Schema Structure

The SOAP API uses two Web Service Definition Language (WSDL) files that define the object and metadata API interfaces:

- ▶ `objectws.wsdl` — Retrieve this from [http://dns\\_name/soap/object?wsdl](http://dns_name/soap/object?wsdl).
- ▶ `metadatas.wsdl` — Retrieve this from [http://dns\\_name/soap/metadata?wsdl](http://dns_name/soap/metadata?wsdl).

Refer to each WSDL for its specific structure.

Clients can use the WSDL files to generate proxy classes, which they then can use to communicate with the service. The Atmos Web service responds with a SOAP response (when an operation is successful) or a SOAP fault (in case of an error).

The SOAP and WSDL standards are implemented as defined by <http://www.w3.org>.

The WSDL files contain all aspects of the XML structure, including schema definitions. As a result, you do not need to make an external reference to an XSD file to perform validation. The XSD section of the Atmos WSDL files describes the XML structure represented in the body of the SOAP envelope. The proxy classes you build and call with each XML request define the structure and typing to be included in the body of the SOAP envelope.

---

## Common Elements

This following table lists the XML elements that are common across the Web service. Each SOAP operation in this guide describes the specific behavior of an element.

Some elements are globally defined from within the XSD section of the WSDL files. Those elements are listed in the table as links and described in more detail in sections following the table.

---

Element	Description
<a href="#">ACL</a>	Permissions for the object ID.
<a href="#">DirectoryList</a>	A listing of contents for a specified directory.
<a href="#">ExtentType</a>	The offset and size of an Atmos object on the Atmos server.
Filename	The optional namespace identifier of an object.
<a href="#">MetadataTags</a>	The content related to metadata output.
Object	The binary content of the object.
<a href="#">ObjectEntry</a>	Represents a single object, including its object ID and optional system and user metadata.

---

Element	Description
ObjectID	The ID of an Atmos object assigned by the Atmos server. An ID is unique to an object.
ObjectLength	The size (in bytes) of the object body content. This element is represented as an integer.
ObjectType	The MIME type of the object.
<a href="#">SystemMetadataList</a>	The system metadata that will be set on the Atmos metadata server. The metadata are specified as name-value pairs.
<a href="#">UID</a>	The subtenant ID and user ID (UID) of an Atmos user. Only one UID is allowed for authentication, but multiple UIDs are allowed for setting ACLs. See ACL in this table.
<a href="#">UserMetadataList</a>	The user metadata that will be set on the Atmos metadata server. The metadata are specified as name-value pairs.

## ACL

The ACL (Access Control Lists) element is a global definition from the object WSDL file, which specifies the access-control permissions assigned to an object. You can assign permissions to an object at the `USER` or `GROUP` level. The following XML example shows the structure of the ACL definition and related types: `Grant`, `Grantee`, and `Permission`.

### Schema Representation

```
<xsd:complexType name="ACL">
  <xsd:sequence>
    <xsd:element name="Grant" type="tns:Grant" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

## Grant

Grant is a complex type that uses the types `Grantee` and `Permission`.

### Schema Representation

```
<xsd:complexType name="Grant">
  <xsd:sequence>
    <xsd:element name="Grantee" type="tns:Grantee"/>
    <xsd:element name="Permission" type="tns:Permission"/>
  </xsd:sequence>
</xsd:complexType>
```

## Grantee

Grantee is a complex type that describes whether the permission granted applies to a USER or GROUP. The only valid input for the GROUP type is `other`, which specifies that the object can be acted on by all other users.

### Schema Representation

```
<xsd:complexType name="Grantee">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="Type" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="USER"/>
            <xsd:enumeration value="GROUP"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

## Permission

Permission is a simple type that works in conjunction with Grantee to specify the type of control a user has over a file, including:

- ▶ READ
- ▶ WRITE
- ▶ FULL\_CONTROL
- ▶ NONE

### Schema Representation

```
<xsd:simpleType name="Permission">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="READ">
    <xsd:enumeration value="WRITE">
    <xsd:enumeration value="FULL_CONTROL">
    <xsd:enumeration value="NONE">
  </xsd:restriction>
</xsd:simpleType>
```

---

## DirectoryList

A directory list is a listing of contents for a specified directory.

### Schema Representation

```
<xsd:complexType name="DirectoryEntry">
  <xsd:sequence>
    <xsd:element name="ObjectID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="FileType" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="FileName" type="xsd:string" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="DirectoryList">
  <xsd:sequence>
    <xsd:element name="DirectoryEntry" type="tns:DirectoryEntry" minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

---

## ExtentType

`ExtentType` is a globally defined, complex type that describes the *Size* and *Offset* of an Atmos object on the Atmos server. Both *Offset* and *Size* must be non-negative integers.

*Size* should never be 0 (or an error is returned).

To read/update the entire object, do not include `<Extent/>`.

It is not possible to read from an offset to the end of the object, without knowing the object's size.

### Schema Representation

```
<xsd:complexType name="ExtentType">
  <xsd:sequence>
    <xsd:element name="Size" type="xsd:int" minOccurs="1" maxOccurs="1" />
    <xsd:element name="Offset" type="xsd:int" minOccurs="1" maxOccurs="1" />
  </xsd:sequence>
</xsd:complexType>
```

### Sample

```
<soapenv:Body wsu:Id="SigID-90772948-37ac-1dd1" xmlns:wsu="http://docs.oasis-open.org/
  <cos:ReadObject xmlns:cos="http://www.emc.com/cos"
    <cos:ObjectID>5calable0a0573760484f89ce0ffa30484fbbfa789ee</cos:ObjectID>
    <cos:Extent>
      <cos:Size>0</cos:Size>
      <cos:Offset>0</cos:Offset>
    </cos:Extent>
  </cos:ReadObject>
</soapenv:Body>
```

---

## MetadataTags

Tags are a way of classifying an object. For example, a user who wants to assign tags that classify the photos he took while on vacation might create tags called beach, hotel, restaurant, and so on.

MetadataTags is a list of tags. When used in a response, the Listable element will be used to specify whether the given tag is listable, or non-listable

MetadataTags define the content for Tag, which resolves to the TagEntry type:

```
<xsd:complexType name="MetadataTags">
  <xsd:sequence>
    <xsd:element name="Tag" type="tns:TagEntry" minOccurs="0" maxOccurs="128" />
  </xsd:sequence>
</xsd:complexType>
```

## TagEntry

TagEntry defines the elements Name and Listable. Name returns the tag name, and Listable returns a boolean that specifies whether the tag name is listable:

```
<xsd:complexType name="TagEntry">
  <xsd:sequence>
    <xsd:element name="Name" type="xsd:string" minOccurs="1" maxOccurs="1" />
    <xsd:element name="Listable" type="xsd:boolean" minOccurs="0" maxOccurs="1" />
  </xsd:sequence>
</xsd:complexType>
```

The GetUserMetadata and ListUserMetadata operations automatically include the Listable flag. When you want to retrieve metadata for specific tags, you do not have to specify this flag value.

User metadata is not constrained. The Listable flag is part of the metadata. Any time you set or reset metadata, you can set or reset this flag as well.

---

## ObjectEntry

An object entry represents a single object, including its object ID and optional system and user metadata.

### **Schema Representation**

```
<xsd:complexType name="ObjectEntry">
  <xsd:sequence>
    <xsd:element name="ObjectID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="SystemMetadataList" type="tns:SystemMetadataList"
minOccurs="0" maxOccurs="1"/>
    <xsd:element name="UserMetadataList" type="tns:UserMetadataList"
minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

---

## SystemMetadataList

System metadata describes attributes like change time and last access time. System metadata is returned by the <metadata> tag, which resolves to the [SystemMetadataEntry](#) type.

## SystemMetadataEntry

### Schema Representation

```
<xsd:complexType name="SystemMetadataList">
  <xsd:sequence>
    <xsd:element name="Metadata" type="tns:SystemMetadataEntry" minOccurs="0"
      maxOccurs="128"/>
  </xsd:sequence>
</xsd:complexType>
```

The following WSDL section shows the sequence for the `SystemMetadataEntry` type.

```
<xsd:complexType name="SystemMetadataEntry">
  <xsd:sequence>
    <xsd:element name="Name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="Value" type="xsd:string" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

For a list of system metadata, see [“System Metadata” on page 12](#).

---

## UserMetadataList

`UserMetadataList` specifies the user metadata associated with an object. `UserMetadataList` is a globally defined, complex type.

Metadata names and values sent through the SOAP interface can use any characters from the Unicode/UTF-8 character set.

**NOTE:** The SOAP interface accepts a wider range of characters for metadata than the REST interface (which accepts the more limited iso-8859-1 character set). If an object that is created or updated via SOAP has metadata names or values that contain characters outside the iso-8859-1 character set, those characters cannot be encoded correctly if that object is requested via REST. In this case, the metadata name/value pair is not returned; instead a separate header is returned, containing a list of metadata names with characters that are unencodable for REST. This response header is `x-emc-unencodable-meta`; for example:

```
x-emc-unencodable-meta: mymetakey1, mymetakey2
```

## UserMetadataEntry

UserMetadataEntry references a name-value pair and a boolean value that specifies whether the tag is listable.

### Schema Representation

```
<xsd:complexType name="UserMetadataList">
  <xsd:sequence>
    <xsd:element name="Metadata" type="tns:UserMetadataEntry" minOccurs="0"
      maxOccurs="128"/>
  </xsd:sequence>
</xsd:complexType>
```

The following WSDL section shows the sequence for the UserMetadataEntry type:

```
<xsd:complexType name="UserMetadataEntry">
  <xsd:sequence>
    <xsd:element name="Name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="Value" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    <xsd:element name="Listable" type="xsd:boolean" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

### Sample

```
<cos:UserMetadataList>
  <cos:Metadata>
    <cos:Name>part2</cos:Name>
    <cos:Value>here</cos:Value>
    <cos:Listable>>false</cos:Listable>
  </cos:Metadata>
  <cos:Metadata>
    <cos:Name>part4</cos:Name>
    <cos:Value>slow</cos:Value>
    <cos:Listable>>true</cos:Listable>
  </cos:Metadata>
</cos:UserMetadataList>
```

---

## UID

This element specifies the UID of an application that is consuming the Atmos API and the ID of the subtenant to which the UID belongs. If the subtenant ID is missing, the ID that is used is that of the default subtenant for the tenant who has access to the node to which you are connecting. Only one UID is allowed per request. The subtenant ID and UID are passed within the <soapenv:Header...> section of the XML payload.

A UID is a global definition for both WSDL files. It is expressed in the schema as follows:

```
<xsd:element name="UID" type="xsd:string" />
```

For example:

```
<soapenv:Header>
  <cos:UID xmlns:cos="http://www.emc.com/cos">9907fb118be24f5d8619567bfb207eeb/user1</cos:UID>
</soapenv:Header>
```

# 5 SOAP API Reference

This chapter describes the SOAP API. Each section includes a description, typing, sample data structure, and schema. Each operation describes the content required to appear within the `<soapenv:Body>` and `</soapenv:Body>` tags.

Operations are defined in two WSDLs. See [Chapter 4, “SOAP WSDL Schema Structure” on page 66](#).

When you create an object via Web services (using either the object or namespace interface), you specify a UID. Within the Atmos file system, the object you create is assigned a file-system UID and a default GID (group ID), where the UID is identical to the UID you specified in your create operation. Permissions must be set properly on the authentication system of the file-system mounting host, to ensure that objects created via Web services are accessible from the file-system interface. Failure to set permissions properly may result in an access error when attempting to retrieve a file.

The SOAP API has the following operations:

Type of Operation	Operation Name
Data management	<a href="#">Creating an Object</a>
	<a href="#">Deleting an Object</a>
	<a href="#">Getting an ACL</a>
	<a href="#">Listing Objects</a>
	<a href="#">Listing Versions</a>
	<a href="#">Reading an Object</a>
	<a href="#">Setting an ACL</a>
	<a href="#">Updating an Object</a>
	<a href="#">Versioning an Object</a>
Metadata management	<a href="#">Deleting User Metadata</a>
	<a href="#">Getting Listable Tags</a>
	<a href="#">Getting User Metadata</a>
	<a href="#">Getting System Metadata</a>
	<a href="#">Listing User Metadata Tags</a>
	<a href="#">Setting User Metadata</a>

---

## Creating an Object

`CreateObject` creates an object with optional user metadata and ACLs. Once created, an object can be increased to any size. No validation is done on the metadata. Appropriate system metadata is generated automatically. You can specify the permissions associated with the object, the metadata tags assigned to the object, and whether the tag is listable. For details on indexing user metadata, see [Chapter 4, “SOAP WSDL Schema Structure” on page 66](#).

`CreateObject` also can be used to create directories. Directories can be created implicitly or explicitly:

- ▶ Implicitly — Specify the full path for an object, and the new director(y)ies is(are) are created automatically created as needed, before creating the object itself.
- ▶ Explicitly — Use `CreateObject` and end the directory name with a forward slash (/). There should be no attachment with the request.

The response content is the Atmos object ID/filename submitted with the original request.

**NOTE:** *Directly under the / directory, you can create only directories, not files.*

### Input parameters

Element	Type	Min Occurs	Max Occurs
ACL	<a href="#">ACL</a>	0	1
FileName	String	0	1
Object	<code>xmime:base64Binary</code>	0	1
ObjectLength	Integer	0	1
ObjectType	String	0	1
UserMetadataList	<a href="#">UserMetadataList</a>	0	1

### Output parameters

Element	Type	Min Occurs	Max Occurs
ObjectID	String	1	1

## Object Interface

### Request

```
<cos:CreateObject xmlns:cos="http://www.emc.com/cos">
  <cos:ACL>
    <cos:Grant>
      <cos:Grantee Type="USER">john</cos:Grantee>
      <cos:Permission>FULL_CONTROL</cos:Permission>
    </cos:Grant>
    <cos:Grant>
      <cos:Grantee Type="GROUP">other</cos:Grantee>
      <cos:Permission>NONE</cos:Permission>
    </cos:Grant>
  </cos:ACL>
  <cos:UserMetadataList>
    <cos:Metadata>
      <cos:Name>part1</cos:Name>
      <cos:Value>buy</cos:Value>
      <cos:Listable>>false</cos:Listable>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>part4/part7/part8</cos:Name>
      <cos:Value>quick</cos:Value>
      <cos:Listable>>true</cos:Listable>
    </cos:Metadata>
  </cos:UserMetadataList>
  <cos:Object>
    <xop:Include href="cid:1.4e0fb8e6-fe98-1dd1-2c42-000c29777466@apache.org"
xmlns:xop="http://www.w3.org/2004/08/xop/include"/>
  </cos:Object>
  <cos:ObjectLength>211</cos:ObjectLength>
</cos:CreateObject>
```

### Response

```
<cos:CreateObjectResponse xmlns:cos="http://www.emc.com/cos">
  <cos:ObjectID>4924264aa10573d404924281caf51f049242d810edc8</cos:ObjectID>
</cos:CreateObjectResponse>
```

## Namespace Interface

### Request

```
<cos:CreateObject xmlns:cos="http://www.emc.com/cos">
  <cos:ACL>
    <cos:Grant>
      <cos:Grantee Type="USER">john</cos:Grantee>
      <cos:Permission>FULL_CONTROL</cos:Permission>
    </cos:Grant>
    <cos:Grant>
      <cos:Grantee Type="GROUP">other</cos:Grantee>
      <cos:Permission>NONE</cos:Permission>
    </cos:Grant>
  </cos:ACL>
  <cos:UserMetadataList>
    <cos:Metadata>
      <cos:Name>part1</cos:Name>
      <cos:Value>buy</cos:Value>
      <cos:Listable>>false</cos:Listable>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>part4/part7/part8</cos:Name>
      <cos:Value>quick</cos:Value>
      <cos:Listable>>true</cos:Listable>
    </cos:Metadata>
  </cos:UserMetadataList>
  <cos:FileName>/photos/mypicture.jpg</cos:FileName>
  <cos:Object>
    <xop:Include href="cid:1.7af5f9e2-fe98-1dd1-29b6-000c29777466@apache.org"
  xmlns:xop="http://www.w3.org/2004/08/xop/include"/>
  </cos:Object>
  <cos:ObjectLength>211</cos:ObjectLength>
</cos:CreateObject>
```

### Response

```
<cos:CreateObjectResponse xmlns:cos="http://www.emc.com/cos">
  <cos:ObjectID>499ad542a1a8bc200499ad5a6b05580499d78224dd0f</cos:ObjectID>
</cos:CreateObjectResponse>
```

---

## Deleting an Object

DeleteObject deletes the object associated with the specified object ID/filename. By deleting the object, all metadata is deleted as well.

If the operation succeeds, the response returns code 1000. If the operation fails, the SOAP envelope returns the error status; see [Appendix 8, "Error Messages and Status Codes"](#) on page 126.

### Input parameters

Element	Type	Min Occurs	Max Occurs
Filename	String	1	1
— OR —			
ObjectID			

### Output parameters

Element	Type	Min Occurs	Max Occurs
Code	String	1	1
Description	String	1	1

### Object Interface

#### Request

```
<cos:DeleteObject xmlns:cos="http://www.emc.com/cos">
  <cos:ObjectID>499ad542a1a8bc200499ad5a6b05580499d813016a65</cos:ObjectID>
</cos:DeleteObject>
```

#### Response

```
<cos:DeleteObjectResponse xmlns:cos="http://www.emc.com/cos">
  <cos:Code>1000</cos:Code>
  <cos:Description>OK</cos:Description>
</cos:DeleteObjectResponse>
```

### Namespace Interface

#### Request

```
<cos:DeleteObject xmlns:cos="http://www.emc.com/cos">
  <cos:FileName>/photos/myoldpicture.jpg</cos:FileName>
</cos:DeleteObject>
```

#### Response

```
<cos:DeleteObjectResponse xmlns:cos="http://www.emc.com/cos">
  <cos:Code>1000</cos:Code>
  <cos:Description>OK</cos:Description>
</cos:DeleteObjectResponse>
```

## Deleting User Metadata

DeleteUserMetadata deletes all user metadata with the specified user-metadata tags, for the specified object. You specify the tags of the pairs to be deleted. *This operation applies only to user metadata.*

If the operation succeeds, the response returns code 1000. If the operation fails, the SOAP envelope contains the error status; see [Appendix 8, "Error Messages and Status Codes"](#) on page 126.

### Input parameters

Element	Type	Min Occurs	Max Occurs
Filename — OR — ObjectID	String	1	1
MetadataTags	<a href="#">MetadataTags</a>	1	1

### Output parameters

Element	Type	Min Occurs	Max Occurs
Code	String	1	1
Description	String	1	1

### Object Interface

#### Request

```
<cos:DeleteUserMetadata xmlns:cos="http://www.emc.com/cos">
  <cos:ObjectID>499ad542a2a8bc200499ad5a7099940499d77d6e8a01</cos:ObjectID>
  <cos:MetadataTags>
    <cos:Tag>
      <cos:Name>part1</cos:Name>
    </cos:Tag>
  </cos:MetadataTags>
</cos:DeleteUserMetadata>
```

#### Response

```
<cos:DeleteUserMetadataResponse xmlns:cos="http://www.emc.com/cos">
  <cos:Code>1000</cos:Code>
  <cos:Description>OK</cos:Description>
</cos:DeleteUserMetadataResponse>
```

### Namespace Interface

#### Request

```
<cos:DeleteUserMetadata xmlns:cos="http://www.emc.com/cos">
  <cos:FileName>/photos/mypicture.jpg</cos:FileName>
  <cos:MetadataTags>
    <cos:Tag>
      <cos:Name>part1</cos:Name>
    </cos:Tag>
  </cos:MetadataTags>
</cos:DeleteUserMetadata>
```

#### Response

```
<cos:DeleteUserMetadataResponse xmlns:cos="http://www.emc.com/cos">
  <cos:Code>1000</cos:Code>
  <cos:Description>OK</cos:Description>
</cos:DeleteUserMetadataResponse>
```

---

## Getting an ACL

GetACL retrieves the permissions associated with the specified object.

### Input parameters

Element	Type	Min Occurs	Max Occurs
<i>Filename</i>	String	1	1
— OR —			
<i>ObjectID</i>			

### Output parameters

Element	Type	Min Occurs	Max Occurs
ACL	<a href="#">ACL</a>	1	1

### Object Interface

#### Request

```
<cos:GetACL xmlns:cos="http://www.emc.com/cos">
  <cos:ObjectID>499ad542a2a8bc200499ad5a7099940499d77d6e8a01</cos:ObjectID>
</cos:GetACL>
```

#### Response

```
<cos:GetACLResponse xmlns:cos="http://www.emc.com/cos">
  <cos:ACL>
    <cos:Grant>
      <cos:Grantee Type="GROUP">other</cos:Grantee>
      <cos:Permission>READ</cos:Permission>
    </cos:Grant>
    <cos:Grant>
      <cos:Grantee Type="USER">fred</cos:Grantee>
      <cos:Permission>FULL_CONTROL</cos:Permission>
    </cos:Grant>
    <cos:Grant>
      <cos:Grantee Type="USER">john</cos:Grantee>
      <cos:Permission>FULL_CONTROL</cos:Permission>
    </cos:Grant>
    <cos:Grant>
      <cos:Grantee Type="USER">user1</cos:Grantee>
      <cos:Permission>FULL_CONTROL</cos:Permission>
    </cos:Grant>
  </cos:ACL>
</cos:GetACLResponse>
```

## Namespace Interface

### Request

```
<cos:GetACL xmlns:cos="http://www.emc.com/cos">
  <cos:FileName>/photos/mypicture.jpg</cos:FileName>
</cos:GetACL>
```

### Response

```
<cos:GetACLResponse xmlns:cos="http://www.emc.com/cos">
  <cos:ACL>
    <cos:Grant>
      <cos:Grantee Type="GROUP">other</cos:Grantee>
      <cos:Permission>READ</cos:Permission>
    </cos:Grant>
    <cos:Grant>
      <cos:Grantee Type="USER">fred</cos:Grantee>
      <cos:Permission>FULL_CONTROL</cos:Permission>
    </cos:Grant>
    <cos:Grant>
      <cos:Grantee Type="USER">john</cos:Grantee>
      <cos:Permission>FULL_CONTROL</cos:Permission>
    </cos:Grant>
    <cos:Grant>
      <cos:Grantee Type="USER">user1</cos:Grantee>
      <cos:Permission>FULL_CONTROL</cos:Permission>
    </cos:Grant>
  </cos:ACL>
</cos:GetACLResponse>
```

---

## Getting Listable Tags

GetListableTags retrieves all listable metadata tags; i.e., tags that act as a directory for indexed objects. For example, if an object using the tag name tag1 is indexed by tag1, tag1 is returned by this operation. Unlike all other operations, this operation is executed under the global namespace (not against an object/file).

The response payload identifies the listable tags.

### Input parameters

Element	Type	Min Occurs	Max Occurs
Tag	String	0	1

### Output parameters

Element	Type	Min Occurs	Max Occurs
MetadataTags	<a href="#">MetadataTags</a>	1	1

### Request

This executes against the root, so all top-level tags are returned. Specifying a tag is optional.

```
<cos:GetListableTags xmlns:cos="http://www.emc.com/cos"/>
```

### Response

```
<cos:GetListableTagsResponse xmlns:cos="http://www.emc.com/cos">
  <cos:MetadataTags>
    <cos:Tag>
      <cos:Name>part4</cos:Name>
    </cos:Tag>
    <cos:Tag>
      <cos:Name>part3</cos:Name>
    </cos:Tag>
    <cos:Tag>
      <cos:Name>part1</cos:Name>
    </cos:Tag>
    <cos:Tag>
      <cos:Name>location</cos:Name>
    </cos:Tag>
  </cos:MetadataTags>
</cos:GetListableTagsResponse>
```

## Getting System Metadata

GetSystemMetadata returns the system metadata for the specified object. You specify the types of system metadata to be returned; if there is no tag, all types are returned.

For a list of the system metadata that you can request from the Web service, see [“System Metadata” on page 12](#).

### Input parameters

Element	Type	Min Occurs	Max Occurs
<i>Filename</i> — OR — <i>ObjectID</i>	String	1	1
MetadataTags	<a href="#">MetadataTags</a>	1	1

### Output parameters

Element	Type	Min Occurs	Max Occurs
SystemMetadataList	<a href="#">SystemMetadataList</a>	1	1

### Object Interface

#### Request

```
<cos:GetSystemMetadata xmlns:cos="http://www.emc.com/cos">
  <cos:ObjectID>499ad542a2a8bc200499ad5a7099940499d77d6e8a01</cos:ObjectID>
</cos:GetSystemMetadata>
```

#### Response

```
<cos:GetSystemMetadataResponse xmlns:cos="http://www.emc.com/cos">
  <cos:SystemMetadataList>
    <cos:Metadata>
      <cos>Name>atime</cos>Name>
      <cos:Value>2009-02-19T15:31:53Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos>Name>mtime</cos>Name>
      <cos:Value>2009-02-19T15:16:39Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos>Name>ctime</cos>Name>
      <cos:Value>2009-02-19T15:31:53Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos>Name>itime</cos>Name>
      <cos:Value>2009-02-19T15:16:38Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos>Name>type</cos>Name>
      <cos:Value>regular</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos>Name>uid</cos>Name>
      <cos:Value>user1</cos:Value>
    </cos:Metadata>
  </cos:SystemMetadataList>
</cos:GetSystemMetadataResponse>
```

```

<cos:Metadata>
  <cos:Name>gid</cos:Name>
  <cos:Value>apache</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>objectid</cos:Name>
  <cos:Value>499ad542a2a8bc200499ad5a7099940499d77d6e8a01</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>objname</cos:Name>
  <cos:Value></cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>size</cos:Name>
  <cos:Value>211</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>nlink</cos:Name>
  <cos:Value>0</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>policyname</cos:Name>
  <cos:Value>default</cos:Value>
</cos:Metadata>
</cos:SystemMetadataList>
</cos:GetSystemMetadataResponse>

```

### **Namespace Interface**

#### **Request**

```

<cos:GetSystemMetadata xmlns:cos="http://www.emc.com/cos">
  <cos:FileName>/photos/mypicture.jpg</cos:FileName>
</cos:GetSystemMetadata>

```

#### **Response**

```

<cos:GetSystemMetadataResponse xmlns:cos="http://www.emc.com/cos">
  <cos:SystemMetadataList>
    <cos:Metadata>
      <cos:Name>atime</cos:Name>
      <cos:Value>2009-02-19T15:33:27Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>mtime</cos:Name>
      <cos:Value>2009-02-19T15:17:54Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>ctime</cos:Name>
      <cos:Value>2009-02-19T15:33:27Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>itime</cos:Name>
      <cos:Value>2009-02-19T15:17:54Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>type</cos:Name>
      <cos:Value>regular</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>uid</cos:Name>
      <cos:Value>user1</cos:Value>
    </cos:Metadata>
  </cos:SystemMetadataList>
</cos:GetSystemMetadataResponse>

```

```
<cos:Metadata>
  <cos:Name>gid</cos:Name>
  <cos:Value>apache</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>objectid</cos:Name>
  <cos:Value>499ad542a1a8bc200499ad5a6b05580499d78224dd0f</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>objname</cos:Name>
  <cos:Value>mypicture.jpg</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>size</cos:Name>
  <cos:Value>211</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>nlink</cos:Name>
  <cos:Value>1</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>policyname</cos:Name>
  <cos:Value>default</cos:Value>
</cos:Metadata>
</cos:SystemMetadataList>
</cos:GetSystemMetadataResponse>
```

---

## Getting User Metadata

GetUserMetadata returns metadata name-value pairs associated with the user-defined tags for the specified object. You specify the tags of the pairs to be returned. If no tag is specified, all pairs are returned.

### Input parameters

Element	Type	Min Occurs	Max Occurs
<i>Filename</i> — OR — <i>ObjectID</i>	String	1	1
MetadataTags	<a href="#">MetadataTags</a>	1	1

### Output parameters

Element	Type	Min Occurs	Max Occurs
UserMetadataList	<a href="#">UserMetadataList</a>	0	1

### Object Interface

#### Request

```
<cos:GetUserMetadata xmlns:cos="http://www.emc.com/cos">
  <cos:ObjectID>499ad542a2a8bc200499ad5a7099940499d77d6e8a01</cos:ObjectID>
</cos:GetUserMetadata>
```

#### Response

```
<cos:GetUserMetadataResponse xmlns:cos="http://www.emc.com/cos">
  <cos>UserMetadataList<
    <cos:Metadata>
      <cos>Name>part1</cos>Name>
      <cos:Value>buy</cos:Value>
      <cos>Listable>>false</cos>Listable>
    </cos:Metadata>
    <cos:Metadata>
      <cos>Name>part4/part7/part8</cos>Name>
      <cos:Value>quick</cos:Value>
      <cos>Listable>>true</cos>Listable>
    </cos:Metadata>
    <cos:Metadata>
      <cos>Name>part3</cos>Name>
      <cos:Value>now</cos:Value>
      <cos>Listable>>false</cos>Listable>
    </cos:Metadata>
  </cos>UserMetadataList>
</cos:GetUserMetadataResponse>
```

## Namespace Interface

### Request

```
<cos:GetUserMetadata xmlns:cos="http://www.emc.com/cos">
  <cos:FileName>/photos/mypicture.jpg</cos:FileName>
</cos:GetUserMetadata>
```

### Response

```
<cos:GetUserMetadataResponse xmlns:cos="http://www.emc.com/cos">
  <cos:UserMetadataList>
    <cos:Metadata>
      <cos:Name>part1</cos:Name>
      <cos:Value>buy</cos:Value>
      <cos:Listable>>false</cos:Listable>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>part4/part7/part8</cos:Name>
      <cos:Value>quick</cos:Value>
      <cos:Listable>>true</cos:Listable>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>part3</cos:Name>
      <cos:Value>now</cos:Value>
      <cos:Listable>>false</cos:Listable>
    </cos:Metadata>
  </cos:UserMetadataList>
</cos:GetUserMetadataResponse>
```

---

## Listing Objects

`ListObjects` accepts one listable metadata tag name or a set of tag names separated by a slash (/) in the request, and it returns the objects to which you have access, indexed by those tags. The requesting UID will not be able to read the contents of the object unless the owner of the object has explicitly granted access to the requesting UID.

If `IncludeMetadata` is used in the request with a value of `false` or omitted from the request, `ListObjects` returns only object IDs. If `IncludeMetadata` is used in the request with a value of `true`, `ListObjects` returns an object list with system and user metadata.

The response returns object IDs. The output corresponds to the tag supplied in the request payload.

### Input parameters

Element	Type	Min Occurs	Max Occurs
<code>IncludeMetadata</code>	Boolean	0	1
<code>Tag</code>	String	1	1

### Output parameters

Element	Type	Min Occurs	Max Occurs
<code>Object</code>	<a href="#">ObjectEntry</a>	0	unbounded

### *Object Interface—without Metadata*

#### Request

```
<cos:ListObjects xmlns:cos="http://www.emc.com/cos">
  <cos:Tag>location</cos:Tag>
  <cos:IncludeMetadata>>false</cos:IncludeMetadata>
</cos:ListObjects>
```

#### Response

```
<cos:ListObjectsResponse xmlns:cos="http://www.emc.com/cos">
  <cos:Object>
    <cos:ObjectID>499ad542a2a8bc200499ad5a7099940499d7fa9f3e60</cos:ObjectID>
  </cos:Object>
  <cos:Object>
    <cos:ObjectID>499ad542a2a8bc200499ad5a7099940499d7fb1ed4b7</cos:ObjectID>
  </cos:Object>
  <cos:Object>
    <cos:ObjectID>499ad542a1a8bc200499ad5a6b05580499d7fc150a02</cos:ObjectID>
  </cos:Object>
</cos:ListObjectsResponse>
```

## Object Interface—with Metadata

### Request

```
<cos:ListObjects xmlns:cos="http://www.emc.com/cos">
  <cos:Tag>location</cos:Tag>
  <cos:IncludeMetadata>true</cos:IncludeMetadata>
</cos:ListObjects>
```

### Response

```
<cos:ListObjectsResponse xmlns:cos="http://www.emc.com/cos">
  <cos:Object>
    <cos:ObjectID>499ad542a2a8bc200499ad5a7099940499d7fa9f3e60</cos:ObjectID>
    <cos:SystemMetadataList>
      <cos:Metadata>
        <cos:Name>atime</cos:Name>
        <cos:Value>2009-02-19T15:50:02Z</cos:Value>
      </cos:Metadata>
      <cos:Metadata>
        <cos:Name>mtime</cos:Name>
        <cos:Value>2009-02-19T15:50:02Z</cos:Value>
      </cos:Metadata>
      <cos:Metadata>
        <cos:Name>ctime</cos:Name>
        <cos:Value>2009-02-19T15:50:02Z</cos:Value>
      </cos:Metadata>
      <cos:Metadata>
        <cos:Name>itime</cos:Name>
        <cos:Value>2009-02-19T15:50:01Z</cos:Value>
      </cos:Metadata>
      <cos:Metadata>
        <cos:Name>type</cos:Name>
        <cos:Value>regular</cos:Value>
      </cos:Metadata>
      <cos:Metadata>
        <cos:Name>uid</cos:Name>
        <cos:Value>user1</cos:Value>
      </cos:Metadata>
      <cos:Metadata>
        <cos:Name>gid</cos:Name>
        <cos:Value>apache</cos:Value>
      </cos:Metadata>
      <cos:Metadata>
        <cos:Name>objectid</cos:Name>
        <cos:Value>499ad542a2a8bc200499ad5a7099940499d7fa9f3e60</cos:Value>
      </cos:Metadata>
      <cos:Metadata>
        <cos:Name>objname</cos:Name>
        <cos:Value></cos:Value>
      </cos:Metadata>
      <cos:Metadata>
        <cos:Name>size</cos:Name>
        <cos:Value>211</cos:Value>
      </cos:Metadata>
      <cos:Metadata>
        <cos:Name>nlink</cos:Name>
        <cos:Value>0</cos:Value>
      </cos:Metadata>
      <cos:Metadata>
        <cos:Name>policyname</cos:Name>
        <cos:Value>default</cos:Value>
      </cos:Metadata>
    </cos:SystemMetadataList>
```

```
<cos:UserMetadataList>
  <cos:Metadata>
    <cos:Name>location</cos:Name>
    <cos:Value>cambridge</cos:Value>
    <cos:Listable>true</cos:Listable>
  </cos:Metadata>
</cos:UserMetadataList>
</cos:Object>
...
</cos:ListObjectsResponse>
```

---

## Listing User Metadata Tags

ListUserMetadataTags returns the user metadata tags for the specified object.

### Input parameters

Element	Type	Min Occurs	Max Occurs
<i>Filename</i>	String	1	1
— OR —			
<i>ObjectID</i>			

### Output parameters

Element	Type	Min Occurs	Max Occurs
MetadataTags	<a href="#">MetadataTags</a>	1	1

### Object Interface

#### Request

```
<cos:ListUserMetadataTags xmlns:cos="http://www.emc.com/cos">  
  <cos:ObjectID>499ad542a2a8bc200499ad5a7099940499d77d6e8a01</cos:ObjectID>  
</cos:ListUserMetadataTags>
```

#### Response

```
<cos:ListUserMetadataTagsResponse xmlns:cos="http://www.emc.com/cos">  
  <cos:MetadataTags>  
    <cos:Tag>  
      <cos:Name>part2</cos:Name>  
      <cos:Listable>>false</cos:Listable>  
    </cos:Tag>  
    <cos:Tag>  
      <cos:Name>part3</cos:Name>  
      <cos:Listable>>false</cos:Listable>  
    </cos:Tag>  
    <cos:Tag>  
      <cos:Name>part4/part7/part8</cos:Name>  
      <cos:Listable>>true</cos:Listable>  
    </cos:Tag>  
    <cos:Tag>  
      <cos:Name>part4/part9</cos:Name>  
      <cos:Listable>>true</cos:Listable>  
    </cos:Tag>  
  </cos:MetadataTags>  
</cos:ListUserMetadataTagsResponse>
```

## **Namespace Interface**

### **Request**

```
<cos:ListUserMetadataTags xmlns:cos="http://www.emc.com/cos">
  <cos:FileName>/photos/mypicture.jpg</cos:FileName>
</cos:ListUserMetadataTags>
```

### **Response**

```
<cos:ListUserMetadataTagsResponse xmlns:cos="http://www.emc.com/cos">
  <cos:MetadataTags>
    <cos:Tag>
      <cos:Name>part2</cos:Name>
      <cos:Listable>>false</cos:Listable>
    </cos:Tag>
    <cos:Tag>
      <cos:Name>part3</cos:Name>
      <cos:Listable>>false</cos:Listable>
    </cos:Tag>
    <cos:Tag>
      <cos:Name>part4/part7/part8</cos:Name>
      <cos:Listable>>true</cos:Listable>
    </cos:Tag>
    <cos:Tag>
      <cos:Name>part4/part9</cos:Name>
      <cos:Listable>>true</cos:Listable>
    </cos:Tag>
  </cos:MetadataTags>
</cos:ListUserMetadataTagsResponse>
```

---

## Listing Versions

ListVersions retrieves the object IDs of all objects in the versioning chain of the specified object, including itself. Versioned objects also can be used as the specified object to the ListVersions operation. Object IDs are 44 characters long, and there is no limit to how many objects you can retrieve.

### Input parameters

Element	Type	Min Occurs	Max Occurs
Filename	String	1	1
— OR —			
ObjectID			

### Output parameters

Element	Type	Min Occurs	Max Occurs
ObjectID	String	0	unbounded

### Object Interface

#### Request

```
<cos:ListVersions xmlns:cos="http://www.emc.com/cos">
  <cos:ObjectID>499ad542a2a8bc200499ad5a7099940499d77d6e8a01</cos:ObjectID>
</cos:ListVersions>
```

#### Response

```
<cos:ListVersionsResponse xmlns:cos="http://www.emc.com/cos">
  <cos:ObjectID>499ad542a2a8bc200499ad5a7099940499d77d6e8a01</cos:ObjectID>
  <cos:ObjectID>499ad542a2a8bc200499ad5a7099940499d7bf6e4ee0</cos:ObjectID>
</cos:ListVersionsResponse>
```

### Namespace Interface

#### Request

```
<cos:ListVersions xmlns:cos="http://www.emc.com/cos">
  <cos:FileName>/photos/mypicture.jpg</cos:FileName>
</cos:ListVersions>
```

#### Response

```
<cos:ListVersionsResponse xmlns:cos="http://www.emc.com/cos">
  <cos:ObjectID>499ad542a1a8bc200499ad5a6b05580499d78224dd0f</cos:ObjectID>
  <cos:ObjectID>499ad542a1a8bc200499ad5a6b05580499d7c109f443</cos:ObjectID>
</cos:ListVersionsResponse>
```

## Reading an Object

`ReadObjects` returns the binary contents of the specified object from `offset` to `offset + size`.

### Input parameters

Element	Type	Min Occurs	Max Occurs
<i>Filename</i> — OR — <i>ObjectID</i>	String	1	1
Extent	<a href="#">ExtentType</a>	0	1

### Output parameters

The response provides `SystemMetadataList`, `UserMetadataList`, `ACL`, and either `Object` and `ObjectType` or `DirectoryList`.

Element	Type	Min Occurs	Max Occurs
ACL	<a href="#">ACL</a>	0	1
DirectoryList	<a href="#">DirectoryList</a>	1	1
Object	xmime:base64Binary	1	1
ObjectType	String	0	1
SystemMetadataList	<a href="#">SystemMetadataList</a>	0	1
UserMetadataList	<a href="#">UserMetadataList</a>	0	1

### Object Interface

#### Request 1

```
<cos:ReadObject xmlns:cos="http://www.emc.com/cos">
  <cos:ObjectID>499ad542a2a8bc200499ad5a7099940499d77d6e8a01</cos:ObjectID>
</cos:ReadObject>
```

#### Response 1

```
<cos:ReadObjectResponse xmlns:cos="http://www.emc.com/cos">
  <cos:SystemMetadataList>
    <cos:Metadata>
      <cos:Name>atime</cos:Name>
      <cos:Value>2009-02-19T15:46:13Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>mtime</cos:Name>
      <cos:Value>2009-02-19T15:46:13Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>ctime</cos:Name>
      <cos:Value>2009-02-19T15:46:13Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>itime</cos:Name>
      <cos:Value>2009-02-19T15:16:38Z</cos:Value>
    </cos:Metadata>
  </cos:SystemMetadataList>
</cos:ReadObjectResponse>
```

```

<cos:Metadata>
  <cos:Name>type</cos:Name>
  <cos:Value>regular</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>uid</cos:Name>
  <cos:Value>user1</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>gid</cos:Name>
  <cos:Value>apache</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>objectid</cos:Name>
  <cos:Value>499ad542a2a8bc200499ad5a7099940499d77d6e8a01</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>objname</cos:Name>
  <cos:Value></cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>size</cos:Name>
  <cos:Value>211</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>nlink</cos:Name>
  <cos:Value>0</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>policyname</cos:Name>
  <cos:Value>default</cos:Value>
</cos:Metadata>
</cos:SystemMetadataList>
<cos:UserMetadataList>
  <cos:Metadata>
    <cos:Name>part1</cos:Name>
    <cos:Value>buy</cos:Value>
    <cos:Listable>>false</cos:Listable>
  </cos:Metadata>
  <cos:Metadata>
    <cos:Name>part4/part7/part8</cos:Name>
    <cos:Value>quick</cos:Value>
    <cos:Listable>>true</cos:Listable>
  </cos:Metadata>
  <cos:Metadata>
    <cos:Name>part3</cos:Name>
    <cos:Value>now</cos:Value>
    <cos:Listable>>false</cos:Listable>
  </cos:Metadata>
  <cos:Metadata>
    <cos:Name>part2</cos:Name>
    <cos:Value>here</cos:Value>
    <cos:Listable>>false</cos:Listable>
  </cos:Metadata>
  <cos:Metadata>
    <cos:Name>part4/part9</cos:Name>
    <cos:Value>slow</cos:Value>
    <cos:Listable>>true</cos:Listable>
  </cos:Metadata>
</cos:UserMetadataList>

```

```

<cos:ACL>
  <cos:Grant>
    <cos:Grantee Type="GROUP">other</cos:Grantee>
    <cos:Permission>READ</cos:Permission>
  </cos:Grant>
  <cos:Grant>
    <cos:Grantee Type="USER">fred</cos:Grantee>
    <cos:Permission>FULL_CONTROL</cos:Permission>
  </cos:Grant>
  <cos:Grant>
    <cos:Grantee Type="USER">john</cos:Grantee>
    <cos:Permission>WRITE</cos:Permission>
  </cos:Grant>
  <cos:Grant>
    <cos:Grantee Type="USER">user1</cos:Grantee>
    <cos:Permission>FULL_CONTROL</cos:Permission>
  </cos:Grant>
</cos:ACL>
<cos:ObjectType/>
<cos:Object>
  <xop:Include href="cid:1.ce7e9c78-fe9c-1dd1-3c5f-000c29777466@apache.org"
xmlns:xop="http://www.w3.org/2004/08/xop/include"/>
</cos:Object>
</cos:ReadObjectResponse>

```

**Request 2** This request is for a directory that contains one subdirectory and one file.

```

<cos:ReadObject xmlns:cos="http://www.emc.com/cos">
  <cos:ObjectID>49a2b73da2a8bc20049a2b79d84405049a316695b311</cos:ObjectID>
</cos:ReadObject>

```

**Response 2**

```

<cos:ReadObjectResponse xmlns:cos="http://www.emc.com/cos">
  <cos:SystemMetadataList>
    <cos:Metadata>
      <cos:Name>atime</cos:Name>
      <cos:Value>2009-02-23T21:34:33Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>mtime</cos:Name>
      <cos:Value>2009-02-23T21:34:33Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>ctime</cos:Name>
      <cos:Value>2009-02-23T21:34:33Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>itime</cos:Name>
      <cos:Value>2009-02-23T21:34:33Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>type</cos:Name>
      <cos:Value>directory</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>uid</cos:Name>
      <cos:Value>user1</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>gid</cos:Name>
      <cos:Value>apache</cos:Value>
    </cos:Metadata>
  </cos:SystemMetadataList>
</cos:ReadObjectResponse>

```

```

<cos:Metadata>
  <cos:Name>objectid</cos:Name>
  <cos:Value>49a2b73da2a8bc20049a2b79d84405049a316695b311</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>objname</cos:Name>
  <cos:Value>mydirectory</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>size</cos:Name>
  <cos:Value>4096</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>nlink</cos:Name>
  <cos:Value>1</cos:Value>
</cos:Metadata>
<cos:Metadata>
  <cos:Name>policyname</cos:Name>
  <cos:Value>default</cos:Value>
</cos:Metadata>
</cos:SystemMetadataList>
<cos:UserMetadataList/>
<cos:ACL>
  <cos:Grant>
    <cos:Grantee Type="GROUP">other</cos:Grantee>
    <cos:Permission>NONE</cos:Permission>
  </cos:Grant>
  <cos:Grant>
    <cos:Grantee Type="USER">user1</cos:Grantee>
    <cos:Permission>FULL_CONTROL</cos:Permission>
  </cos:Grant>
</cos:ACL>
<cos:DirectoryList>
  <cos:DirectoryEntry>
    <cos:ObjectID>49a2b73da2a8bc20049a2b79d84405049a41b41ee06a</cos:ObjectID>
    <cos:FileType>directory</cos:FileType>
    <cos:FileName>mysubdirectory</cos:FileName>
  </cos:DirectoryEntry>
  <cos:DirectoryEntry>
    <cos:ObjectID>49a2b73da2a8bc20049a2b79d84405049a41b5091679</cos:ObjectID>
    <cos:FileType>regular</cos:FileType>
    <cos:FileName>myfile.txt</cos:FileName>
  </cos:DirectoryEntry>
</cos:DirectoryList>
</cos:ReadObjectResponse>

```

## Namespace Interface

### Request 1

```
<cos:ReadObject xmlns:cos="http://www.emc.com/cos">
  <cos:FileName>/photos/mypicture.jpg</cos:FileName>
</cos:ReadObject>
```

### Response 1

```
<cos:ReadObjectResponse xmlns:cos="http://www.emc.com/cos">
  <cos:SystemMetadataList>
    <cos:Metadata>
      <cos:Name>atime</cos:Name>
      <cos:Value>2009-02-19T15:46:56Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>mtime</cos:Name>
      <cos:Value>2009-02-19T15:46:55Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>ctime</cos:Name>
      <cos:Value>2009-02-19T15:46:56Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>itime</cos:Name>
      <cos:Value>2009-02-19T15:17:54Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>type</cos:Name>
      <cos:Value>regular</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>uid</cos:Name>
      <cos:Value>user1</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>gid</cos:Name>
      <cos:Value>apache</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>objectid</cos:Name>
      <cos:Value>499ad542a1a8bc200499ad5a6b05580499d78224dd0f</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>objname</cos:Name>
      <cos:Value>mypicture.jpg</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>size</cos:Name>
      <cos:Value>211</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>nlink</cos:Name>
      <cos:Value>1</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>polycyname</cos:Name>
      <cos:Value>default</cos:Value>
    </cos:Metadata>
  </cos:SystemMetadataList>
```

```

<cos:UserMetadataList>
  <cos:Metadata>
    <cos:Name>part1</cos:Name>
    <cos:Value>buy</cos:Value>
    <cos:Listable>>false</cos:Listable>
  </cos:Metadata>
  <cos:Metadata>
    <cos:Name>part4/part7/part8</cos:Name>
    <cos:Value>quick</cos:Value>
    <cos:Listable>>true</cos:Listable>
  </cos:Metadata>
  <cos:Metadata>
    <cos:Name>part3</cos:Name>
    <cos:Value>now</cos:Value>
    <cos:Listable>>false</cos:Listable>
  </cos:Metadata><cos:Metadata>
    <cos:Name>part2</cos:Name>
    <cos:Value>here</cos:Value>
    <cos:Listable>>false</cos:Listable>
  </cos:Metadata>
  <cos:Metadata>
    <cos:Name>part4/part9</cos:Name>
    <cos:Value>slow</cos:Value>
    <cos:Listable>>true</cos:Listable>
  </cos:Metadata>
</cos:UserMetadataList>
<cos:ACL>
  <cos:Grant>
    <cos:Grantee Type="GROUP">other</cos:Grantee>
    <cos:Permission>READ</cos:Permission>
  </cos:Grant>
  <cos:Grant>
    <cos:Grantee Type="USER">fred</cos:Grantee>
    <cos:Permission>FULL_CONTROL</cos:Permission>
  </cos:Grant>
  <cos:Grant>
    <cos:Grantee Type="USER">john</cos:Grantee>
    <cos:Permission>WRITE</cos:Permission>
  </cos:Grant>
  <cos:Grant>
    <cos:Grantee Type="USER">user1</cos:Grantee>
    <cos:Permission>FULL_CONTROL</cos:Permission>
  </cos:Grant>
</cos:ACL>
<cos:ObjectType/>
<cos:Object>
  <xop:Include href="cid:1.d8bd826c-fe9c-1dd1-39af-000c29777466@apache.org"
xmlns:xop="http://www.w3.org/2004/08/xop/include"/>
</cos:Object>
</cos:ReadObjectResponse>

```

**Request 2** This request is for a directory that contains one subdirectory and one file.

```
<cos:ReadObject xmlns:cos="http://www.emc.com/cos">
  <cos:FileName>mydirectory</cos:FileName>
</cos:ReadObject>
```

**Response 2**

```
<cos:ReadObjectResponse xmlns:cos="http://www.emc.com/cos">
  <cos:SystemMetadataList>
    <cos:Metadata>
      <cos:Name>atime</cos:Name>
      <cos:Value>2009-02-23T21:34:33Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>mtime</cos:Name>
      <cos:Value>2009-02-23T21:34:33Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>ctime</cos:Name>
      <cos:Value>2009-02-23T21:34:33Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>itime</cos:Name>
      <cos:Value>2009-02-23T21:34:33Z</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>type</cos:Name>
      <cos:Value>directory</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>uid</cos:Name>
      <cos:Value>user1</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>gid</cos:Name>
      <cos:Value>apache</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>objectid</cos:Name>
      <cos:Value>49a2b73da2a8bc20049a2b79d84405049a316695b311</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>objname</cos:Name>
      <cos:Value>mydirectory</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>size</cos:Name>
      <cos:Value>4096</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>nlink</cos:Name>
      <cos:Value>1</cos:Value>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>policyname</cos:Name>
      <cos:Value>default</cos:Value>
    </cos:Metadata>
  </cos:SystemMetadataList>
</cos:UserMetadataList/>
```

```
<cos:ACL>
  <cos:Grant>
    <cos:Grantee Type="GROUP">other</cos:Grantee>
    <cos:Permission>NONE</cos:Permission>
  </cos:Grant>
  <cos:Grant>
    <cos:Grantee Type="USER">user1</cos:Grantee>
    <cos:Permission>FULL_CONTROL</cos:Permission>
  </cos:Grant>
</cos:ACL>
<cos:DirectoryList>
  <cos:DirectoryEntry>
    <cos:ObjectID>49a2b73da2a8bc20049a2b79d84405049a41b41ee06a</cos:ObjectID>
    <cos:FileType>directory</cos:FileType>
    <cos:FileName>mysubdirectory</cos:FileName>
  </cos:DirectoryEntry>
  <cos:DirectoryEntry>
    <cos:ObjectID>49a2b73da2a8bc20049a2b79d84405049a41b5091679</cos:ObjectID>
    <cos:FileType>regular</cos:FileType>
    <cos:FileName>myfile.txt</cos:FileName>
  </cos:DirectoryEntry>
</cos:DirectoryList>
</cos:ReadObjectResponse>
```

---

## Setting an ACL

SetACL sets/modifies the ACL permissions for the specified object.

If the operation succeeds, the response returns code 1000. If the operation fails, the SOAP envelope contains the error status; see [Appendix 8, "Error Messages and Status Codes"](#) on page 126.

### Input parameters

Element	Type	Min Occurs	Max Occurs
<i>Filename</i> — OR — <i>ObjectID</i>	String	1	1
ACL	<a href="#">ACL</a>	1	1

### Output parameters

Element	Type	Min Occurs	Max Occurs
Code	String	1	1
Description	String	1	1

### Object Interface

#### Request

```
<cos:SetACL xmlns:cos="http://www.emc.com/cos">
  <cos:ObjectID>499ad542a2a8bc200499ad5a7099940499d77d6e8a01</cos:ObjectID>
  <cos:ACL>
    <cos:Grant>
      <cos:Grantee Type="USER">fred</cos:Grantee>
      <cos:Permission>FULL_CONTROL</cos:Permission>
    </cos:Grant>
    <cos:Grant>
      <cos:Grantee Type="GROUP">other</cos:Grantee>
      <cos:Permission>READ</cos:Permission>
    </cos:Grant>
  </cos:ACL>
</cos:SetACL>
```

#### Response

```
<cos:SetACLResponse xmlns:cos="http://www.emc.com/cos">
  <cos:Code>1000</cos:Code>
  <cos:Description>OK</cos:Description>
</cos:SetACLResponse>
```

## ***Namespace Interface***

### **Request**

```
<cos:SetACL xmlns:cos="http://www.emc.com/cos">
  <cos:FileName>/photos/mypicture.jpg</cos:FileName>
  <cos:ACL>
    <cos:Grant>
      <cos:Grantee Type="USER">fred</cos:Grantee>
      <cos:Permission>FULL_CONTROL</cos:Permission>
    </cos:Grant>
    <cos:Grant>
      <cos:Grantee Type="GROUP">other</cos:Grantee>
      <cos:Permission>READ</cos:Permission>
    </cos:Grant>
  </cos:ACL>
</cos:SetACL>
```

### **Response**

```
<cos:SetACLResponse xmlns:cos="http://www.emc.com/cos">
  <cos:Code>1000</cos:Code>
  <cos:Description>OK</cos:Description>
</cos:SetACLResponse>
```

---

## Setting User Metadata

SetUserMetadata writes the user metadata into the specified object. If the tag does not exist, you can create it and write the corresponding value. If the tag exists, you can replace the existing value. All metadata elements must be specified, or the server returns an error.

If the operation succeeds, the response returns code 1000. If the operation fails, the SOAP envelope contains the error status; see [Appendix 8, "Error Messages and Status Codes"](#) on page 126.

### Input parameters

Element	Type	Min Occurs	Max Occurs
<i>Filename</i> — OR — <i>ObjectID</i>	String	1	1
UserMetadataList	<a href="#">UserMetadataList</a>	1	1

### Output parameters

Element	Type	Min Occurs	Max Occurs
Code	String	1	1
Description	String	1	1

### Object Interface

#### Request

```
<cos:SetUserMetadata xmlns:cos="http://www.emc.com/cos">
  <cos:ObjectID>499ad542a2a8bc200499ad5a7099940499d77d6e8a01</cos:ObjectID>
  <cos>UserMetadataList>
    <cos:Metadata>
      <cos>Name>part3</cos>Name>
      <cos:Value>now</cos:Value>
      <cos>Listable>>false</cos>Listable>
    </cos:Metadata>
  </cos>UserMetadataList>
</cos:SetUserMetadata>
```

#### Response

```
<cos:SetUserMetadataResponse xmlns:cos="http://www.emc.com/cos">
  <cos:Code>1000</cos:Code>
  <cos>Description>OK</cos>Description>
</cos:SetUserMetadataResponse>
```

## ***Namespace Interface***

### **Request**

```
<cos:SetUserMetadata xmlns:cos="http://www.emc.com/cos">
  <cos:FileName>/photos/mypicture.jpg</cos:FileName>
  <cos:UserMetadataList>
    <cos:Metadata>
      <cos:Name>part3</cos:Name>
      <cos:Value>now</cos:Value>
      <cos:Listable>>false</cos:Listable>
    </cos:Metadata>
  </cos:UserMetadataList>
</cos:SetUserMetadata>
```

### **Response**

```
<cos:SetUserMetadataResponse xmlns:cos="http://www.emc.com/cos">
  <cos:Code>1000</cos:Code>
  <cos:Description>OK</cos:Description>
</cos:SetUserMetadataResponse>
```

---

## Updating an Object

`UpdateObject` updates (replaces) the data at the specified size and offset. If no `Extent` is specified, the object is replaced. If `object.size <= (offset + size)`, the update replaces the data up to `size`, then appends the rest.

You can use `UpdateObject` to truncate an object. To change the object's size, do one of the following:

- ▶ To truncate an object to `size=0`, omit `Extent`, and specify a request that does not contain an attachment. Truncating an object to `size=0` leaves the object ID unchanged.
- ▶ To overwrite an object, omit `Extent` and attach the new object content to the request body.
- ▶ To append to an object, specify `Extent` with:

```
offset=currentSizeOfTheObject
size=sizeOfTheAttachment
```

and attach the data corresponding to the content increase to the request body.

If the operation succeeds, the response returns code 1000. If the operation fails, the SOAP envelope contains the error status; [Appendix 8, "Error Messages and Status Codes" on page 126](#).

### Input parameters

Element	Type	Min Occurs	Max Occurs
ACL	<a href="#">ACL</a>	0	1
Extent	<a href="#">ExtentType</a>	0	1
<i>Filename</i> — OR — <i>ObjectID</i>	String	1	1
Object	<code>xmime:base64Binary</code>	1	1
ObjectLength	Integer	1	1
ObjectType	String	0	1
UserMetadataList	<a href="#">UserMetadataList</a>	0	1

### Output parameters

Element	Type	Min Occurs	Max Occurs
Code	String	1	1
Description	String	1	1

## Object Interface

### Request

```
<cos:UpdateObject xmlns:cos="http://www.emc.com/cos">
  <cos:ACL>
    <cos:Grant>
      <cos:Grantee Type="USER">john</cos:Grantee>
      <cos:Permission>WRITE</cos:Permission>
    </cos:Grant>
  </cos:ACL>
  <cos:UserMetadataList>
    <cos:Metadata>
      <cos:Name>part2</cos:Name>
      <cos:Value>here</cos:Value>
      <cos:Listable>>false</cos:Listable>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>part4/part9</cos:Name>
      <cos:Value>slow</cos:Value>
      <cos:Listable>>true</cos:Listable>
    </cos:Metadata>
  </cos:UserMetadataList>
  <cos:Extent>
    <cos:Size>9</cos:Size>
    <cos:Offset>10</cos:Offset>
  </cos:Extent>
  <cos:ObjectID>499ad542a2a8bc200499ad5a7099940499d77d6e8a01</cos:ObjectID>
  <cos:Object>
    <xop:Include href="cid:1.6f5c20bc-fe9c-1dd1-3cff-000c29777466@apache.org"
xmlns:xop="http://www.w3.org/2004/08/xop/include"/>
  </cos:Object>
  <cos:ObjectLength>9</cos:ObjectLength>
</cos:UpdateObject>
```

### Response

```
<cos:UpdateObjectResponse xmlns:cos="http://www.emc.com/cos">
  <cos:Code>1000</cos:Code>
  <cos:Description>OK</cos:Description>
</cos:UpdateObjectResponse>
```

## Namespace Interface

### Request

```
<cos:UpdateObject xmlns:cos="http://www.emc.com/cos">
  <cos:ACL>
    <cos:Grant>
      <cos:Grantee Type="USER">john</cos:Grantee>
      <cos:Permission>WRITE</cos:Permission>
    </cos:Grant>
  </cos:ACL>
  <cos:UserMetadataList>
    <cos:Metadata>
      <cos:Name>part2</cos:Name>
      <cos:Value>here</cos:Value>
      <cos:Listable>>false</cos:Listable>
    </cos:Metadata>
    <cos:Metadata>
      <cos:Name>part4/part9</cos:Name>
      <cos:Value>slow</cos:Value>
      <cos:Listable>>true</cos:Listable>
    </cos:Metadata>
  </cos:UserMetadataList>
  <cos:Extent>
    <cos:Size>9</cos:Size>
    <cos:Offset>10</cos:Offset>
  </cos:Extent>
  <cos:FileName>/photos/mypicture.jpg</cos:FileName>
  <cos:Object>
    <xop:Include href="cid:1.890abb5e-fe9c-1dd1-2c6b-000c29777466@apache.org"
xmlns:xop="http://www.w3.org/2004/08/xop/include"/>
  </cos:Object>
  <cos:ObjectLength>9</cos:ObjectLength>
</cos:UpdateObject>
```

### Response

```
<cos:UpdateObjectResponse xmlns:cos="http://www.emc.com/cos">
  <cos:Code>1000</cos:Code>
  <cos:Description>OK</cos:Description>
</cos:UpdateObjectResponse>
```

---

## Versioning an Object

`VersionObject` creates an immutable copy of the specified object.

It is supported only for objects, not for directories or the namespace interface.

### Input parameters

Element	Type	Min Occurs	Max Occurs
<i>Filename</i>	String	1	1
— OR —			
<i>ObjectID</i>			

### Output parameters

Element	Type	Min Occurs	Max Occurs
<i>ObjectID</i>	String	1	1

### **Object Interface**

#### Request

```
<cos:VersionObject xmlns:cos="http://www.emc.com/cos">
  <cos:ObjectID>499ad542a2a8bc200499ad5a7099940499d77d6e8a01</cos:ObjectID>
</cos:VersionObject>
```

#### Response

```
<cos:VersionObjectResponse xmlns:cos="http://www.emc.com/cos">
  <cos:ObjectID>499ad542a2a8bc200499ad5a7099940499d7bf6e4ee0</cos:ObjectID>
</cos:VersionObjectResponse>
```

### **Namespace Interface**

#### Request

```
<cos:VersionObject xmlns:cos="http://www.emc.com/cos">
  <cos:FileName>/photos/mypicture.jpg</cos:FileName>
</cos:VersionObject>
```

#### Response

```
<cos:VersionObjectResponse xmlns:cos="http://www.emc.com/cos">
  <cos:ObjectID>499ad542a1a8bc200499ad5a6b05580499d7c109f443</cos:ObjectID>
</cos:VersionObjectResponse>
```



**PART IV:**

**ADVANCED TOPICS**

# 6 Reserved Namespace for Extended Attributes

For each file/object, there is a protected namespace — `user.mau.*` — for extended attributes. The namespace can be accessed via the file system using the Atmos installable file system and through the Linux extended-attribute command-line utilities, `getfattr` and `setfattr`. When the installable file system is used, Atmos layers user-metadata access across POSIX extended attributes; some system metadata also can be accessed through the extended-attribute mechanism (see below).

The `user.mau` extended-attribute namespace is reserved; i.e., EMC controls the contents of the namespace and the format of its fields. Some of the `xattrs` are exposed to applications, as defined below (see the table in “[Atmos Extended Attributes](#)” on page 111.” As noted in the table, some `xattrs` can be only queried, others can be queried and modified. Applications cannot create new `xattrs` in this namespace. Failure to follow the defined contents and format of the namespace results in undefined behavior and may lead to future failures or inconsistencies.

---

## Linux Extended Attributes

Extended attributes are name:value pairs associated permanently with files and directories, similar to the environment strings associated with a process. An attribute may be defined or undefined. If it is defined, its value may be empty or non-empty.

Extended attributes are extensions to normal attributes. Often, they are used to provide additional functionality to a file system.

Users with search access to a file or directory may retrieve a list of attribute names defined for that file or directory.

Extended attributes are accessed as atomic objects. Reading retrieves the whole value of an attribute and stores it in a buffer. Writing replaces any previous value with the new value.

For more information, see the extended-attribute manual page. On a Linux system, this can be queried with `man 5 attr`.

---

## Atmos Extended Attributes

The protected namespace contains the following attributes:

Attribute	Can be Queried?	Can be Set?	See ...
capability	X		<a href="#">“Capability” on page 112</a>
expirationEnable	X	X	<a href="#">“Expiration of Objects” on page 112</a>
expirationEnd	X	X	<a href="#">“Expiration of Objects” on page 112</a>
lso	X		<a href="#">“Layout Storage Object” on page 113</a>
mdsmaster	X		<a href="#">“MDS (Metadata Service)” on page 114</a>
mdsreplicas	X		<a href="#">“MDS (Metadata Service)” on page 114</a>
nlink	X		<a href="#">“Number of Links” on page 114</a>
objectid	X		<a href="#">“Object ID” on page 114</a>
objState	X		objState is an internal field and not relevant for users.
queues	X		<a href="#">“Queues” on page 114</a>
refCount	X		<a href="#">“Reference Count” on page 115</a>
retentionEnable	X	X	<a href="#">“Retention of Objects” on page 115</a>
retentionEnd	X	X	<a href="#">“Retention of Objects” on page 115</a>
stats	X	X	<a href="#">“Statistics” on page 116</a>
tracer	X	X	<a href="#">“Log Tracing” on page 116</a>
updateNum	X		<a href="#">“updateNum” on page 116</a>
version	X		<a href="#">“Version” on page 116</a>

---

## Capability

Generically, a *capability* is an unforgeable token of authority. A capability is granted to an application by an MDS when the application successfully opens an object for access. Subsequently, the capability can be passed by the application to storage servers, to prove to the storage server that the MDS has authorized the application to access the object. The capability transfers notice of the MDS's authorization to the storage servers in a secure manner through the client.

When this is queried, "unavailable" is returned if the client does not have a capability.

### Get Example

```
# getfattr -n user.mauif.capability /mnt/mauifs/bar
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/bar
user.mauif.capability="unavailable"
```

---

## Expiration of Objects

An *expiration* period is a period after which the data is deleted.

Object expiration is controlled by policies. You can change the policy parameter value in the object directly. The parameters are accessible as if they were user-metadata attributes of the object. The policy attributes have Atmos-specific reserved names to distinguish them from user-defined attributes. The reserved names are:

- ▶ **user.mauif.expirationEnable** — Of type string ("true" or "false")
- ▶ **user.mauif.expirationEnd** — Of type xsd:dateTime (e.g., 2008-04-16T10:00:00Z)

You can get/set these attributes through either the file-system interface (the `getfattr/setfattr` examples shown below) or the object interface (`GetUserMetadata/SetUserMetadata`).

NOTE: Expiration applies only to files, not directories.

NOTE: These policy attributes cannot be *created* in an object using the calls to `setfattr/MauifClientSetUserMetadata()`. The attributes must exist as a result of policy application, to be retrieved or updated.

### Get and Set Examples

```
# getfattr -n user.mauif.expirationEnd /mnt/mauifs/CIFS/boat1.jpg
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/CIFS/boat1.jpg
user.mauif.expirationEnd="2009-04-04T23:22:14Z"

# getfattr -n user.mauif.expirationEnable /mnt/mauifs/CIFS/boat1.jpg
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/CIFS/boat1.jpg
user.mauif.expirationEnable="true"

# setfattr -n user.mauif.expirationEnd -v 2009-05-04T23:22:14Z /mnt/mauifs/CIFS/boat1.jpg
```

```
# getfattr -n user.maui.expirationEnd /mnt/mauifs/CIFS/boat1.jpg
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/CIFS/boat1.jpg
user.maui.expirationEnd="2009-05-04T23:22:14Z"

# setfattr -n user.maui.expirationEnable -v false /mnt/mauifs/CIFS/boat1.jpg

# getfattr -n user.maui.expirationEnable /mnt/mauifs/CIFS/boat1.jpg
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/CIFS/boat1.jpg
user.maui.expirationEnable="false"
```

---

## Layout Storage Object

A *Layout Storage Object (LSO)* is a data structure that describes how the data in an object is allocated on one or more SSs (e.g., replication, striping, and chunking into extents).

### Get Example

```
# getfattr -n user.maui.lso /mnt/mauifs/bar
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/bar
user.maui.lso="<?xml version=\"1.0\" encoding=\"UTF-8\"
standalone=\"no\"?>\012<maui:Lso
xmlns:maui=\"http://www.emc.com/maui\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xsi:schemaLocation=\"http://www.emc.com/maui_lso.xsd\"
xsi:type=\"maui:LsoReplica\">\012 <type>Replica</type>\012
<id>1</id>\012 <refcnt>1</refcnt>\012 <replica>\012
<type>sync</type>\012 <current>>true</current>\012 <queryStr>for $h
in CLUSTER/HOST where
$h/METRIC[@NAME=\"mauiss_status\"]/@VAL=\"up\"</queryStr>\012
<revision>2</revision>\012 <child xsi:type=\"maui:LsoExtent\">\012
<type>Extent</type>\012 <id>3</id>\012 <refcnt>1</refcnt>\012
<extent>\012 <offset>0</offset>\012 <length>0</length>\012
<child xsi:type=\"maui:LsoPhysical\">\012
<type>Physical</type>\012 <id>2</id>\012
<refcnt>1</refcnt>\012 <ssaddr>\012 <service>SS</service>\012
<host>indy-003</host>\012 <port>10301</port>\012
<location>Indy</location>\012 </ssaddr>\012
<capacity>0</capacity>\012 <osdid>89</osdid>\012 </child>\012
</extent>\012 </child>\012 </replica>\012 <replica>\012
<type>sync</type>\012 <current>>true</current>\012 <queryStr>for $h
in CLUSTER/HOST where
$h/METRIC[@NAME=\"mauiss_status\"]/@VAL=\"up\"</queryStr>\012
<revision>2</revision>\012 <child xsi:type=\"maui:LsoExtent\">\012
<type>Extent</type>\012 <id>5</id>\012 <refcnt>1</refcnt>\012
<extent>\012 <offset>0</offset>\012 <length>0</length>\012
<child xsi:type=\"maui:LsoPhysical\">\012
<type>Physical</type>\012 <id>4</id>\012
<refcnt>1</refcnt>\012 <ssaddr>\012 <service>SS</service>\012
<host>indy-001</host>\012 <port>10301</port>\012
<location>Indy</location>\012 </ssaddr>\012
<capacity>0</capacity>\012 <osdid>88</osdid>\012 </child>\012
</extent>\012 </child>\012 </replica>\012 <revision>1</revision>\012
<creatLoc>Indy</creatLoc>\012</maui:Lso>\012"
```

---

## MDS (Metadata Service)

The MDS is where metadata is stored and managed.

`mdsmaster` is the MDS that is hosting the database master for the object.

`mdsreplicas` are the MDS(s) that are hosting the database slave(s) for the object.

### Get Examples

```
# getfattr -n user.maui.mdsmaster /mnt/mauifs/bar
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/bar
user.maui.mdsmaster="indy-001:10401:Indy"

# getfattr -n user.maui.mdsreplicas /mnt/mauifs/bar
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/bar
user.maui.mdsreplicas="indy-002:10401"
```

---

## Number of Links

`nlink` is the number of hard links to a file. This is a system-metadata field, generally not relevant to a user application. Hard links are not currently supported, so this always returns 1.

### Get Example

```
# getfattr -n user.maui.nlink /mnt/mauifs/bar
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/bar
user.maui.nlink="1"
```

---

## Object ID

`objectid` is the object ID; for example, `4924264aa10573d404924281caf51f049242d810edc8`.

### Get Example

```
# getfattr -n user.maui.objectid /mnt/mauifs/bar
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/bar
user.maui.objectid="49a660fb0000000000000000000000049a7d8ea59701"
```

---

## Queues

`queues` reports the length of the event queues inside the client library. This is for developer debugging and not relevant for users.

---

## Reference Count

refCount is not currently used. It always returns 0.

### Get Example

```
# getfattr -n user.mauif.refCount /mnt/mauifs/bar
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/bar
user.mauif.refCount="0"
```

---

## Retention of Objects

A *retention* period is a period during which the data cannot be modified.

Object retention is controlled by policies. You can change the policy parameter value in the object directly. The parameters are accessible as if they were user-metadata attributes of the object. The policy attributes have Atmos-specific reserved names to distinguish them from user-defined attributes. The reserved names are:

- ▶ **user.mauif.retentionEnable** — Of type string ("true" or "false")
- ▶ **user.mauif.retentionEnd** — Of type xsd:dateTime (e.g., 2008-04-16T10:00:00Z)

You can get/set these attributes through either the file-system interface (the `getfattr/setattr` examples shown below) or the object interface (`GetUserMetadata/SetUserMetadata`).

**NOTE:** These policy attributes cannot be *created* in an object using the calls to `setfattr/MauifClientSetUserMetadata()`. The attributes must exist as a result of policy application, to be retrieved or updated.

### Get and Set Examples

```
# getfattr -n user.mauif.retentionEnable /mnt/mauifs/CIFS/boat1.jpg
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/CIFS/boat1.jpg
user.mauif.retentionEnable="true"

# getfattr -n user.mauif.retentionEnd /mnt/mauifs/CIFS/boat1.jpg
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/CIFS/boat1.jpg
user.mauif.retentionEnd="2009-03-05T23:22:14Z"

# setfattr -n user.mauif.retentionEnd -v 2009-03-06T23:22:14Z /mnt/mauifs/CIFS/boat1.jpg

# getfattr -n user.mauif.retentionEnd /mnt/mauifs/CIFS/boat1.jpg
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/CIFS/boat1.jpg
user.mauif.retentionEnd="2009-03-06T23:22:14Z"

# setfattr -n user.mauif.retentionEnable -v false /mnt/mauifs/CIFS/boat1.jpg
```

```
# getfattr -n user.maui.retentionEnable /mnt/mauifs/CIFS/boat1.jpg
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/CIFS/boat1.jpg
user.maui.retentionEnable="false"

# getfattr -n user.maui.retentionEnd /mnt/mauifs/CIFS/boat1.jpg
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/CIFS/boat1.jpg
user.maui.retentionEnd="NONE"
```

---

## Statistics

`stats` enables querying to return the performance metrics collected for the internal client library. Valid values to set are `reset/clear`, `enable`, and `disable`. This is for developer debugging and not relevant for users.

---

## Log Tracing

`tracer` can be set but not queried. When it is set (to any value), the logging configuration file is re-read. This is for developer debugging and not relevant for users.

---

## updateNum

`updateNum` is used by the asynchronous-replication mechanism to determine when a replica is current. This is internal metadata and not relevant to users.

---

## Version

A *version* is a point-in-time views of an object, which may provide protection in the event of data corruption, for example. In contrast to replicas, *versions* are represented by their own, discrete, object IDs, the same as the parent object to which they refer.

### ***Get Example for an Object without Versioning***

```
# getfattr -n user.maui.version /mnt/mauifs/foo
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/foo
user.maui.version
```

## Get Example for an Object with Versioning

```
# getfattr -n user.maui.version /mnt/mauifs/baz
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/baz
user.maui.version="<?xml version="1.0" encoding="UTF-8"
standalone="no"?>\012<maui:VersionAttr
xmlns:maui="http://www.emc.com/maui"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.emc.com/maui mauiobj.xsd">\012
<lineageVersion>49a660fb0000000000000000000000049ae98be112a1</lineageVersion>\012
<next>49a660fb0000000000000000000000049ae98cc385d6</next>\012
<verType>UNDELETABLE</verType>\012
<timestamp>2009-03-04T15:05:48Z</timestamp>\012
<counter>1</counter>\012 <sysePOCH>1236179148</sysePOCH>\012
<prevsysePOCH>1236179134</prevsysePOCH>\012</maui:VersionAttr>\012"
```

# 7

## Security

Security for Web services consists of:

- ▶ *Authentication* using an encrypted signature model. See [“Managing Authentication” on page 118](#).
- ▶ *Authorization* through access-control lists (ACLs) at the user (UID) level. See [“Access Control Lists” on page 122](#).

An Atmos user may construct a “pre-authenticated” URL to a specific object that they may then share with anyone. This allows an Atmos user to let a non-Atmos user download a specific object. See [“Shareable URLs” on page 124](#).

---

### Managing Authentication

The Web service uses a combination of the UID and other request headers to produce a signature that authenticates the user accessing the Web service. It uses a combination of various pieces of the message to validate the identity of the sender, integrity of the message, and non-repudiation of the action.

The UID is a unique, static value that identifies your application to the Web service. To complete the operation, you must generate a signature using the shared secret associated with the UID. Without this information, your Web-service application cannot be authenticated by the server. For the UID and shared secret corresponding to your application, contact your system administrator.

**NOTE:** The shared secret is in base64-encoded form and needs to be base64 decoded before it can be used. See the detailed explanation below in [“REST Authentication: Algorithm for Securing REST Messages with Signatures” on page 119](#).

The server retrieves the UID from the request and retrieves the shared secret associated with that UID, stored on the server lockbox. The server then regenerates the signature using the same algorithm as the client. If this signature matches the one in the request, the Web service processes the request and returns the response payload.

#### Time-stamps

Atmos also uses timestamps to enforce a request-validity window. Each request is valid for only a certain window of time from when the request was created on the client; the request must arrive at the server within this window. This request-validity window is designed to protect against replay attacks. If a request is received after this window, the server rejects the request and returns an error to the client. The creation and expiration times of the request are part of the header and are used for signature computation. This ensures that any alteration to these values is detected by the server, and the request is rejected. By default, this time window is plus or minus 5 minutes from the server time, which is in UTC.

## REST Authentication: Algorithm for Securing REST Messages with Signatures

A client using the REST API composes the request and computes a hash of the request using the algorithm for securing REST messages. The UID is stored in a custom HTTP header which is `x-emc-uid` and is a part of the request. Then, a signature is computed by applying HMAC-SHA1 on the hash and using the shared secret that maps to the UID in the request. This signature is appended to the request and sent to the Web service for comparison.

### Signature

The header has the following format:

```
x-emc-signature : signature
```

The *signature* is defined as:

```
signature = Base64(HMACSHA1(HashString))
```

where *Base64* is the base64 encoding of the argument and *HMACSHA1* is the keyed hash of the argument. The shared secret is used for computing *HMACSHA1*. The actual shared secret is in binary format. This binary array of bytes is converted to a human-readable format by base64-encoding it, and this encoded format is what a user receives from the SysAdmin. Make sure the shared secret is base64-decoded before using it as an input to the HMAC-SHA1 algorithm to generate the signature.

For example, here is some Ruby code:

```
digest = HMAC.digest(Digest.new(SHA1), Base64.decode64(key), HashString)
return Base64.encode64(digest.to_s()).chomp()
```

*SHA1* is defined above. *key* is the base64-encoded shared secret that the user receives. When you base64-encode a string, the resulting string may look like this: `xxxxxxxxxxxx\n`. You must call the `chomp()` function to remove the `\n` character at the end of the result string.

### HashString

*HashString* is computed as follows:

```
HTTPRequestMethod + '\n' +
ContentType + '\n' +
Range + '\n' +
Date + '\n' +
CanonicalizedResource + '\n' +
CanonicalizedEMCHheaders
```

where `+` is the concatenation operator.

Components of *HashString* are described in the following table.

Field	Description
HTTPRequestMethod	One of the five HTTP method types, in uppercase: GET, POST, PUT, DELETE, HEAD.
Content-Type	Content type. Only the value is used, not the header name. If a request does not include an HTTP body, this is an empty string.
Range	Range header. Only the value is used, not the header name. If a request does not include the range header, this is an empty string.
Date	(Optional: Date and/or x-emc-date must be in the request.) Standard HTTP header, in UTC format. Only the date value is used, not the header name. If a request does not include the date header, this is an empty string, and the x-emc-date header is then required.
CanonicalizedResource	Path and Query portions of the HTTP request URI, in lowercase. For example, when using the ACL operation (where the Query is ?acl), the value of CanonicalizedResource would be:  /rest/objects/5ca1ab1e0a05737604847ff1f7a26d04848167b63d9f?acl  When reading an object (where there is no Query), the value of CanonicalizedResource would be:  /rest/objects/5ca1ab1e0a05737604847ff1f7a26d04848167b63d9f
CanonicalizedEMCHheaders	Refer to the process below for canonicalizing EMC headers.

### **Canonicalization of Headers**

Canonicalization of EMC headers is done as follows:

- 1 Remove any white space before and after the colon and at the end of the metadata value. Multiple white spaces embedded within a metadata value are replaced by a single white space. For example:

Before canonicalization:      x-emc-meta: title=Mountain    Dew

After canonicalization:        x-emc-meta:title=Mountain Dew

- 2 Convert all header names to lowercase.
- 3 Sort the headers alphabetically.
- 4 For headers with values that span multiple lines, convert them into one line by replacing any newline characters and extra embedded white spaces in the value.
- 5 Concatenate all headers together, using newlines (\n) separating each header from the next one. There should be no terminating newline character at the end of the last header.

## **REST Example Request**

```
POST /rest/objects HTTP/1.1
x-emc-listable-meta: part4/part7/part8=quick
x-emc-meta: part1=buy
accept: */*
x-emc-useracl: john=FULL_CONTROL,mary=WRITE
date: Thu, 05 Jun 2008 16:38:19 GMT
content-type: application/octet-stream
x-emc-date: Thu, 05 Jun 2008 16:38:19 GMT
x-emc-groupacl: other=NONE
host: 10.5.115.118
content-length: 4286
x-emc-uid: 6039ac182f194e15b9261d73ce044939/user1
```

## **REST Example HashString**

```
POST
application/octet-stream

Thu, 05 Jun 2008 16:38:19 GMT
/rest/objects
x-emc-date:Thu, 05 Jun 2008 16:38:19 GMT
x-emc-groupacl:other=NONE
x-emc-listable-meta:part4/part7/part8=quick
x-emc-meta:part1=buy
x-emc-uid:6039ac182f194e15b9261d73ce044939/user1
x-emc-useracl:john=FULL_CONTROL,mary=WRITE
```

Note that there is a blank line included in the above example to account for the missing Range header.

If you use the following key:

```
LJLuryj6zs8ste6Y3jTGQp71xq0=
```

on the hash string above, you will generate the following signature:

```
WHJo1MFevMnK4jCthJ974L3YHoo=
```

## **SOAP Authentication**

The Atmos SOAP Web service uses the WS-Security standard for securing communication between client and server. It implements security using symmetric key authentication. Atmos uses a signature policy to authenticate the UID making the request. The requestor's UID and the subtenant ID to which the UID belongs are part of the SOAP header. The signature is generated by computing the HMAC-SHA1 of a digest string. The digest is a combination of various pieces of the SOAP message. The shared secret used for the HMAC-SHA1 algorithm is shared between the client and the server.

The following policy, which is part of the WSDLs, is used to enforce compliance of incoming SOAP request with WS-Security standards. This policy describes the details of the WS-Security implementation, like the algorithm used for signing, parts of the SOAP message that are used for signature computation, and the canonicalization algorithm used. This policy is meant to be interpreted by any WSDL-to-code generation tool that supports WS-Security standards.

```

<wsp:Policy wsu:ID="SymmetricKeyPolicy">
  <wsp:ExactlyOne>
    <wsp>All>
      <sp:SymmetricBinding>
        <wsp:Policy>
          <sp:ProtectionToken>
            <wsp:Policy>
              <sp:SecurityContextToken
                sp:IncludeToken="http://schemas.xmlsoap.org/ws/
                2005/07/securitypolicy/IncludeToken/Never">
                <wsp:Policy>
                  </wsp:Policy>
                </wsp:Policy>
              </sp:SecurityContextToken>
            </wsp:Policy>
          </sp:ProtectionToken>

          <sp:AlgorithmSuite>
            <wsp:Policy>
              <sp:Basic256/>
            </wsp:Policy>
          </sp:AlgorithmSuite>
          <sp:Layout>
            <wsp:Policy>
              <sp:Strict/>
            </wsp:Policy>
          </sp:Layout>
          <sp:IncludeTimestamp/>
          <sp:EncryptBeforeSigning/>
          <sp:OnlySignEntireHeadersAndBody/>
        </wsp:Policy>
      </sp:SymmetricBinding>
      <sp:Wss10>
        <wsp:Policy>
          <sp:MustSupportRefKeyIdentifier/>
          <sp:MustSupportRefEmbeddedToken/>
          <sp:MustSupportRefIssuerSerial/>
        </wsp:Policy>
      </sp:Wss10>
    </wsp>All>
  </wsp:ExactlyOne>
</wsp:Policy>
<wsp:Policy wsu:ID="SignaturePolicy">
  <sp:SignedPartgs>
    <sp:Body/>
  </sp:SignedParts>
</wsp:Policy>

```

---

## Access Control Lists

UIDs are used for both authentication and controlling access to objects using ACLs. By default, no UID except the owner of an object has any access to the object. The owner may choose to grant access to any UID under the same subtenant as himself. The access level can be `READ`, `WRITE`, or `FULL_CONTROL`. ACLs also can be used to revoke permission to specific UIDs. Note that ACLs cannot be used to grant access to UIDs across different subtenants.

For details on user ACLs for your application, contact your Atmos system administrator.

## REST ACLs

You set user-level authorization with the `x-emc-groupacl` or `x-emc-useracl` custom headers, which define access control for objects (see “[Custom Headers](#)” on page 24). Access control for files and directories is done with standard file-system commands like `chmod`.

### **REST Example Request**

The following example shows a request for the `SetACL` method.

- ▶ The `x-emc-useracl: fred=FULL_CONTROL` header specifies full access control for one user, `fred`.
- ▶ The `x-emc-groupacl: other=READ` header specifies group read attributes for the object.

```
POST /rest/objects/5scalable0a05737604847ff1f7a26d04848167b63d9f?acl HTTP/1.1
accept: */*
x-emc-useracl: fred=FULL_CONTROL
date: Thu, 05 Jun 2008 16:38:23 GMT
content-type: application/octet-stream
x-emc-date: Thu, 05 Jun 2008 16:38:23 GMT
x-emc-groupacl: other=READ
host: 10.5.115.118
x-emc-uid: 6039ac182f194e15b9261d73ce044939/user1
x-emc-signature: MDaCy5+1t7ZYdglRxpIOrF4K1hU=
```

### **REST Example Response**

```
HTTP/1.1 200 OK
Date: Thu, 05 Jun 2008 16:38:23 GMT
Server: Apache/2.0.61 (rPath)
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

## SOAP ACLs

User-level authorization is achieved by using the `ACL` element of the WSDL to set the access control for objects. Access can be granted or revoked to one or more UIDs at the same time.

---

## Shareable URLs

An Atmos user (UID) can construct a pre-authenticated URL to an object, which may then be used by anyone to retrieve the object (e.g., through a browser). This allows an Atmos user to let a non-Atmos user download a specific object. The entire object/file is read.

### URL Syntax

The URL has the following syntax:

```
http://MyAtmosServer/location?uid=uid&expires=expires&signature=signature
```

Where the parameters are defined as follows:

Parameter	Description
<i>location</i>	<i>/objects /object_ID</i> — OR — <i>/namespace/pathname</i>  For example:  <i>/rest/objects/496cbaada2a8bc200496cbb0dd04a004970ce8be68a6</i> — OR — <i>/rest/namespace/videos/mycoolvideo</i>
<i>uid</i>	The UID (and optional subtenant). The UID must have read access to the requested object.
<i>expires</i>	The expiration date/time, specified in seconds since Jan 1 1970 UTC 00:00:00. For example, to expire the object at Fri Feb 20 09:34:28 -0500 2009, <i>expires</i> would be 1235140468. Requests made after this time will fail.
<i>signature</i>	Base64-encoded HMAC-SHA1 of the hash string. See <a href="#">“Calculating the Signature” on page 125</a> . The URL is signed using the UID’s secret key, to prevent tampering.

### Example

The following example is one line. For readability, however, it is shown here on several lines.

```
http://MyAtmosServer/rest/objects/5ca1ab1ec0a8bc1b049412d09a510804941767490dde?  
uid=64dbbc37bef04889b175c9ee21b0517b%2Fuser1&  
expires=1235140468&  
signature=GJdwY1D1ex2CCyuPIyGmc5HdSzw%3D
```

## Calculating the Signature

The signature is defined and calculated as described in [“Signature” on page 119](#).

*HashString* is computed as follows:

```
GET + '\n' +  
requested-resource + '\n' +  
uid + '\n' +  
expires
```

where + is the concatenation operator, and *requested-resource* is lowercase.

**IMPORTANT:** When computing *HashString*, the values for *uid* and *signature* should not be URL-encoded. (They should be URL-encoded when piecing together the final URL.) For example:

```
This UID:          64dbbc37bef04889b175c9ee21b0517b/user1  
Becomes:          64dbbc37bef04889b175c9ee21b0517b%2Fuser1
```

```
This signature:    GJdwY1D1ex2CCyuPIyGMc5HdSzw=  
Becomes:          GJdwY1D1ex2CCyuPIyGMc5HdSzw%3D
```

Here is a sample *HashString* computation:

```
GET\n  
/rest/objects/5ca1ab1ec0a8bc1b049412d09a51080494167490dde\n  
64dbbc37bef04889b175c9ee21b0517b/user1\n  
1235140468
```

In this case, the base64-encoded key that was used is LjLuryj6zs8ste6Y3jTGQp71xq0=.

# 8 Error Messages and Status Codes

This chapter lists the codes that are trapped and returned during Web-service operations.

## REST Information

When the operations are invoked using the REST interface and an exception occurs, the server returns an HTTP error, along with a detailed error message in the response body, which contains the error code and error description.

```
HTTP/1.1 404 Not Found
Date: Thu, 31 Jan 2008 20:03:24 GMT
Server: Apache/2.0.61 (rPath)
Content-Length: 131
Connection: close
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<Error>
  <Code>1003</Code>
  <Message>The requested object was not found.</Message>
</Error>
```

## SOAP Information

When the operations are invoked using the SOAP interface and an exception occurs, the server returns a SOAP fault in the following format:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <soapenv:Fault>
      <faultcode>soapenv:Client|Server.ErrorCode</faultcode>
      <faultstring>Error Description</faultstring>
    </soapenv:Fault>
  </soapenv:Body>
</soapenv:Envelope>
```

Note the parsing in the `<faultcode>` field:

- ▶ The first part specifies whether the error occurred because of an error in the request from the client or a processing error on the server side. In case of a client error, taking corrective action to address the error and resending the request generally results in a successful response. Server errors generally mean there is no corrective action the client can take to resolve the error. In such cases, the client should try submitting a new request; if the problem persists, contact the Atmos system administrator.
- ▶ The second part provides the error code.

An actual error message follows. The <faultcode> field indicates there was an error in the request.

```
<soapenv:Envelope xmlns: soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <soapenv:Fault>
      <faultcode>SOAPenv:Client.1004</faultcode>
      <faultstring>The specified range cannot be satisfied.</faultstring>
    </soapenv:Fault>
  </soapenv:Body>
</soapenv:Envelope>
```

## Error Codes

In the following table, the HTTP status codes and descriptions apply only to REST.

Error Code	Error Message	HTTP Status Code	HTTP Status Description
1001	The server encountered an internal error. Please try again.	500	Internal Server Error
1002	One or more arguments in the request were invalid.	400	Bad Request
1003	The requested object was not found.	404	Not Found
1004	The specified range cannot be satisfied.	416	Requested Range Not Satisfiable
1005	One or more metadata tags were not found for the requested object.	400	Bad Request
1006	Operation aborted because of a conflicting operation in process against the resource.	409	Conflict
	<b>NOTE:</b> This error code may indicate that the system temporarily is too busy to process the request. This is a non-fatal error; you can re-try the request later.		
1007	The server encountered an internal error. Please try again.	500	Internal Server Error
1008	The requested resource was not found on the server.	400	Bad Request
1009	The method specified in the Request is not allowed for the resource identified.	405	Method Not Allowed
1010	The requested object size exceeds the maximum allowed upload/download size.	400	Bad Request
1011	The specified object length does not match the actual length of the attached object.	400	Bad Request

<b>Error Code</b>	<b>Error Message</b>	<b>HTTP Status Code</b>	<b>HTTP Status Description</b>
1012	There was a mismatch between the attached object size and the specified extent size.	400	Bad Request
1013	The server encountered an internal error. Please try again.	500	Internal Server Error
1014	The maximum allowed metadata entries per object has been exceeded.	400	Bad Request
1015	The request could not be finished due to insufficient access privileges.	401	Unauthorized
1016	The resource you are trying to create already exists.	400	Bad Request
1019	The server encountered an internal error. Please try again.	500	Internal Server Error
1020	The requested resource is missing or could not be found.	500	Internal Server Error
1021	The requested resource is not a directory.	400	Bad Request
1022	The requested resource is a directory.	400	Bad Request
1023	The directory you are attempting to delete is not empty.	400	Bad Request
1024	The server encountered an internal error. Please try again.	500	Internal Server Error
1025	The server encountered an internal error. Please try again.	500	Internal Server Error
1026	The server encountered an internal error. Please try again.	500	Internal Server Error
1027	The server encountered an internal error. Please try again.	500	Internal Server Error
1028	The server encountered an internal error. Please try again.	500	Internal Server Error
1029	The server encountered an internal error. Please try again.	500	Internal Server Error
1031	The request timestamp was outside the valid time window.	403	Forbidden
1032	There was a mismatch between the signature in the request and the signature as computed by the server.	403	Forbidden
1033	Unable to retrieve the secret key for the specified user.	403	Forbidden

<b>Error Code</b>	<b>Error Message</b>	<b>HTTP Status Code</b>	<b>HTTP Status Description</b>
1034	Unable to read the contents of the HTTP body.	400	Bad Request
1037	The specified token is invalid.	400	Bad Request

## HTTP Success Codes

<b>HTTP Status Code</b>	<b>HTTP Status Description</b>	<b>Description</b>
200	OK	The request has succeeded.
201	Created	The request has been fulfilled and resulted in a new object being created. This applies to <code>CreateObject</code> and <code>VersionObject</code> requests.
204	No Content	The request has been fulfilled, and no content is being sent with the response. This applies to <code>DeleteObject</code> and <code>DeleteUserMetadata</code> requests.
206	Partial Content	The server has fulfilled the partial <code>GET</code> request for the object. This applies to <code>ReadObject</code> requests that include the <code>Range</code> header).