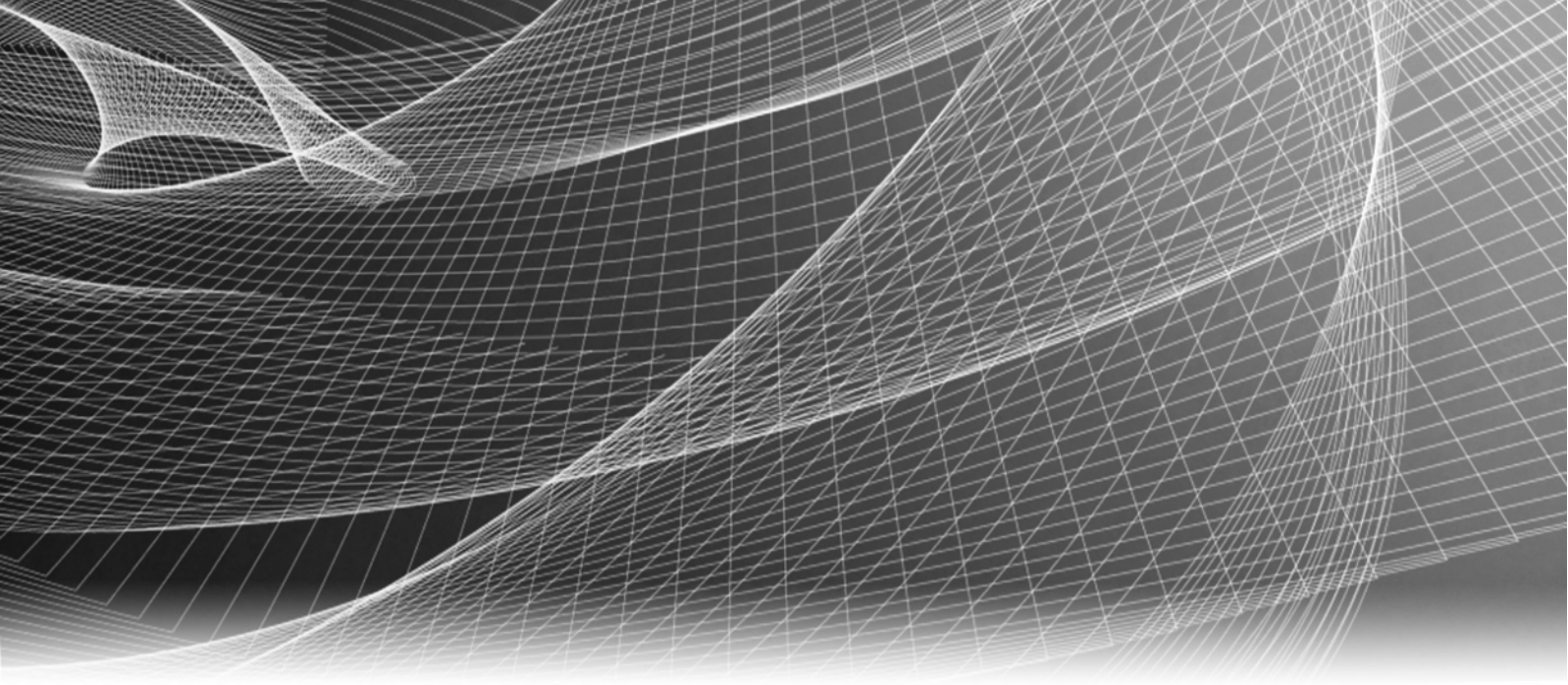


EMC®
Atmos™

Version 2.4

Programmer's Guide

P/N 302-002-655
REV 01

EMC²

Published March, 2016

EMC Corporation
Hopkinton, Massachusetts 01748-9103
1-508-435-1000 In North America 1-866-464-7381
www.EMC.com

CONTENTS

PREFACE

As part of an effort to improve its product lines, EMC periodically releases revisions of its software and hardware. Therefore, some functions described in this document might not be supported by all versions of the software or hardware currently in use. The product release notes provide the most up-to-date information on product features.

Contact your EMC representative if a product does not function properly or does not function as described in this document.

Note: This document was accurate at publication time. New versions of this document might be released on the EMC online support website. Check the EMC online support website to ensure that you are using the latest version of this document.

Audience

This document is part of the Atmos documentation set, and is intended for use by developers who want to programmatically read and write data to Atmos.

Related documentation

The EMC Atmos documentation set includes the following titles:

- *EMC Atmos Release Notes*
- *EMC Atmos Support Matrix*
- *EMC Atmos Administrator's Guide*
- *EMC Atmos Programmer's Guide*
- *EMC Atmos System Management API Guide*
- *EMC Atmos Security Configuration Guide*
- *EMC Atmos CAS Programmer's Guide*
- *EMC Atmos CAS API Reference Guide*
- *EMC Atmos Installable File System (IFS) Installation and Upgrade Guide*
- *EMC Atmos Open Source License and Copyright Information*

Conventions used in this document

EMC uses the following conventions for special notices:

Note: A note presents information that is important, but not hazard-related.

IMPORTANT

An important notice contains information essential to software or hardware operation.

Typographical conventions

EMC uses the following type style conventions in this document:

Normal	Used in running (nonprocedural) text for: • Names of interface elements, such as names of windows, dialog boxes, buttons, fields, and menus • Names of resources, attributes, pools, Boolean expressions, buttons, DQL statements, keywords, clauses, environment variables, functions, and utilities • URLs, pathnames, filenames, directory names, computer names, links, groups, service keys, file systems, and notifications
Bold	Used in running (nonprocedural) text for names of commands, daemons, options, programs, processes, services, applications, utilities, kernels, notifications, system calls, and man pages Used in procedures for: • Names of interface elements, such as names of windows, dialog boxes, buttons, fields, and menus • What the user specifically selects, clicks, presses, or types
Italic	Used in all text (including procedures) for: • Full titles of publications referenced in text • Emphasis, for example, a new term • Variables
`Courier`	Used for: • System output, such as an error message or script • URLs, complete paths, filenames, prompts, and syntax when shown outside of running text
`Courier bold`	Used for specific user input, such as commands
`Courier italic`	Used in procedures for: • Variables on the command line • User input variables
< >	Angle brackets enclose parameter or variable values supplied by the user
[ ]	Square brackets enclose optional values
|	Vertical bar indicates alternate selections — the bar means “or”
{}	Braces enclose content that the user must specify, such as x or y or z
...	Ellipses indicate nonessential information omitted from the example

Where to get help

EMC support, product, and licensing information can be obtained as follows:

Product information - For documentation, release notes, software updates, or information about EMC products, go to EMC Online Support at:

`https://support.emc.com`

Technical support - Go to EMC Online Support and click Service Center. You will see several options for contacting EMC Technical Support. Note that to open a service request, you must have a valid support agreement. Contact your EMC sales representative for details about obtaining a valid support agreement or with questions about your account.

CHAPTER 1
About the Atmos API

This chapter includes the following topics:

Overview

EMC® Atmos™is an object-storage system with enormous scalability and extensibility. It uses metadata-driven policies to manage data placement and data services.

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 that include creating versions and tagging objects with user-defined metadata (to form user-defined collections of objects). APIs are available for REST Web services.

You can use the API to create and manipulate objects and metadata. Applications can associate metadata with the objects they store. Metadata can be used to trigger policies (defined by the system administrator) that meet goals for performance, data protection, content delivery, archiving, and so on.

The Web services APIs support both an *object interface* and a file-system-like *namespace interface* for addressing content.

In addition to the storage API, Atmos includes:

- An API for storing and accessing fixed content (information in its final form). For more information, see the *EMC Atmos CAS Programmer's Guide* and the *EMC Atmos CAS API Reference Guide*.
- A system-management API that lets you implement a subset of system management functions in client applications. For more information, see the *EMC Atmos System Management API Guide*.

Atmos natively supports the Amazon Simple Storage Service (S3) API as an access method for data operations between S3-based applications and Atmos.

EMC recommends that you follow the standard programming practice of checking the return codes when programming with the Atmos API. Always check the return status of the API calls and handle any errors appropriately. This is especially crucial for operations that change the state of the data, such as write and append. If the write operation fails with an error, it should be re-tried. A failed write operation can result in inconsistent data.

About the object interface

In the **object interface**:

- Atmos assigns the object a unique object ID (OID).
- The service endpoint is /rest/objects.
- If you create the object using this interface, you can access it only using the OID.
- The application layer might need to persist OIDs and perform a translation for an application's end users (if necessary).
- Typically a more performant and scalable interface. It does not require any special structuring of data or applications.

About the namespace interface

In the **namespace interface**:

- The object's unique identifier is the object's pathname. Atmos also assigns it an OID.
- The service endpoint is /rest/namespace.
- The file system path provides a natural structuring of data and follows a very familiar paradigm.
- You can use the namespace interface to transition from existing file-based applications to web services.
- Because the object has both an OID and a namespace path, you can use either the object interface or the namespace interface to perform reads.
- Do not use the object interface or the namespace interface interchangeably on creates, updates, or deletes.
- Typically less performant and scalable than the object interface because the system has to traverse a namespace to locate an object.
- Requires careful planning of the namespace layout to ensure efficient performance and scalability. See "Namespace interface dos and don'ts" on page 12 for details.

Note: The Atmos Web Services API does not support symlinks.

System metadata

Atmos supports system metadata and user metadata (see "User metadata" on page 10). System metadata is generated automatically and updated by the system based on a predefined schema.

Table 1 Atmos system metadata (page 1 of 2)

Name	Description	Example
atime	Last access time (write to data or metadata)	2007-10-29T18:19:57Z
ctime	Last metadata modification time	2007-10-29T18:19:56Z
gid	Group ID	Apache
itime	Inception (create) time	2007-10-29T18:19:57Z
mtime	Last user-data 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	Atmos Object ID	4924264aa10573d404924281caf51f049242d810edc8

Table 1 Atmos system metadata (page 2 of 2)

Name	Description	Example
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.	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
x-emc-wschecksum	String containing the checksum value and other related information. *algorithm* — Represents the hashing algorithm used. Values can be: SHA0, SHA1, or md5 *offset* — Represents the offset at which the checksum was calculated. *checksumValue* — Represents the hash of the object at the offset.	sha0/1037/87hn7kkdd9d982f031qwe9ab224abjd6h1276nj9

User metadata

User metadata is a collection of text name-value pairs that are not validated by the system. User metadata allows you the flexibility to create custom tags for data specific to your applications. Atmos supports two kinds of user-defined metadata tags:

- **Non-listable** — Allows you to store user-defined key-value pairs.
- **Listable** — Also allows you to store user-defined key-value pairs, and you can enumerate objects that have the same tag. This ability adds processing overhead that does impact performance.

A user can store up to 127 user-metadata pairs.

Non-listable user metadata

Non-listable metadata tags (also just called metadata tags) are a way of classifying an object. Often, metadata tags are used to trigger policies; for example, 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 more information about policies, see the *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.

Listable user metadata

Listable metadata tags are metadata tags that can be used to index and retrieve objects. Listable tags are private to the user who creates them; tags created by one user cannot be seen by another user.

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. 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`.

Listable tags provide a mechanism 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.

Object tagging guidelines

For listable tags:

- Use listable tags carefully and **only** when necessary.
- Limit the number of unique tags to 50,000.
- A single tag can be associated with a maximum of 6.5 million objects. For applications where more than 6.5 million objects are desired, maximize the number of objects by using multiple tags in a round robin methodology as a single tag. For example, for a listable tag “Application tag=‹tag value›”, use hashed application tags as follows:

```
Application_1 = <tag value> for the first 6,500,000 objects
Application_2 = <tag value> for the second 6.500,000 objects
…
Application_n = <tag value> for the nth 6,500,000 objects
```

- Do not create a deeply nested hierarchy of listable tags.

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.
- 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.
- For information about constructing commands to specify files instead of objects, see "Specifying objects/files in REST commands" on page 66.
- Examples of using the namespace interface for each operation are in Chapter 5 and Chapter 7.

Namespace interface dos and don'ts

To ensure that the namespace interface performs efficiently, plan the namespace according to these rules:

- Design a well-balanced directory structure.
  - Design the directory structure so that the directory tree has *breadth*. The top 2 levels of the directory structure should have tens to hundreds of directories.
  - Do not design the directory structure to be narrow and deep.
- Limit the number of objects in each directory or subdirectory to 100,000 or less.
- Create objects in the Atmos location where they will be accessed most often for update.

Checksum protection

Atmos supports end-to-end SHA-0, SHA-1, md5 checksum protection for objects created with the REST interface that are stored in replicas that use GeoParity. For objects not stored in GeoParity replicas, Atmos uses checksum to ensure the data was not corrupted during HTTP transit.

Atmos requires that you use checksum protection for applications that must conform to SEC 17a-4f standards.

To invoke checksum protection on a create or update request include the x-emc-wschecksum custom header. The x-emc-checksum header includes:

`x-emc-wschecksum: algorithm/offset/checksumValue`

Where:

- *algorithm* — Is set to SHA0, SHA1, or md5.
- *offset* — Specifies the offset where the algorithm was calculated.
- *checksumValue* — Specifies the hash of the object to create or update.

On a create request, you must pass in the checksum value for the complete object. The data flow for an object create request that includes the checksum header is as follows:

1. A web services application invokes a create request passing in the x-emc-wschecksum custom header.
2. The Atmos web services node validates the checksum, and if it is valid, it uses the client API to request the appropriate storage services operations on the object create request. If the checksum does not validate with the request header, Atmos rejects the request.
3. The Atmos storage services writes the new object/fragment to the drives.
4. Atmos returns the success/failure response to the web services application.

You can obtain the checksum value for an object by performing a read operation (using the GET or HEAD HTTP methods) or a Get System Metadata request. The x-emc-wschecksum header is returned in the response.

For more information about checksum operations, see:

- "x-emc-wschecksum"
- "Creating an object"
- "Updating an object"
- "Getting system metadata"

Checksum and system metadata

When an Atmos object is successfully created with a checksum value, Atmos creates checksum metadata for the object. The checksum metadata includes:

```
x-maui-wschecksum =
   Algorithm/Offset/LibraryName/LibraryVersion/ChecksumValue/Context
```

Where:

- Algorithm — The name of the hashing algorithm (SHA0, SHA1, or md5).
- Offset — The offset at which the checksum was computed.
- LibraryName — The name of the library used to compute the checksum.
- LibraryVersion — The version of the library used to compute the checksum.
- ChecksumValue — The hash value for the object at the Offset specified above.
- Context — The serialized context at the offset. This allows the checksum computation to be resumed at the offset.

By including the offset and context in the metadata, Atmos enables applications to resume failed uploads from the point of failure instead of uploading the entire object from the beginning.

If you have an object create or update that fails, you can:

- Issue a HEAD request on the object that failed to upload completely.
- The server returns x-emc-wschecksum header with a value containing the algorithm/offset/checksumValue.
- Your application can then resume the upload (with checksum validation) from this offset instead of from the beginning.

The version object API

The Atmos version object API lets you easily create and manage immutable snapshots of Atmos objects. The object you copy is called the *top-level object*, and the copy is called the *versioned object*.

- You can use versioned objects to:
- Recover a top-level object (in case of accidental deletions or updates).
- Audit changes to an object over time.

An Atmos versioned object is a point-in-time copy that includes the object's data and metadata. The policy applied to the top-level object applies to the versioned object (except for retention or expiration rules.)

When creating versions, Atmos performs a full-range copy-on-write. So if your application:

- Creates one 10MB top-level object and a versioned object. At this time, the versioned object will point to the same data as the top-level object.
- Replaces 1MB of data on the top-level object and creates another versioned object. This versioned object will point to the copied data, which contains the full 10M not just the 1MB difference.

Versioning large objects (those greater than 5GB) might impact system performance depending on your specific performance requirements and the system's overall load.

Once the version is created, it is not affected by changes (including policy changes) to the top-level object. The top-level object is always available for changes. Versioning operations are applied to one object at a time.

Figure 1 illustrates how a versioned object is created.

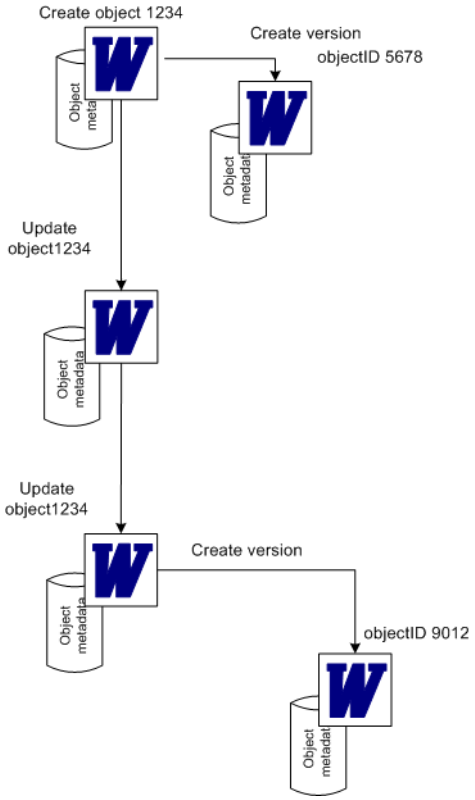


Figure 1 Create versioned object lifecycle

- The user stores a word document as an Atmos object (a top-level object) using the standard object API. It gets an object ID to uniquely identify it, and it has metadata associated with it.
- The user wants a snapshot of the top-level object. To create it, they use the version API create operation (HTTP POST with the /rest/objects/‹*ObjectID*›?versions URI.) Atmos creates an exact copy of the top-level object and the object's metadata. Atmos gives it a unique object ID, and applies the same policy to the copy as it applied to the top-level object.
- The user modifies the top-level object. No changes are made to the version, and no new version is created.
- The user modifies the top-level object, and wants a snapshot of the updated object. They call the version API create again. Atmos creates the versioned object (with metadata), and applies the policy to that object.

The version object API includes the operations described in Table 2.

Table 2 Atmos Version Object API Operations

Operation	Description
Create	Creates a snapshot of the top-level object and returns the object ID for the version. Any subsequent operations (list, delete, restore, get) for the specific version must reference this Object ID.
Delete	Deletes a specific version of the object, and returns capacity to the system upon successful completion of the delete.
List	Enumerates the versions (by Object ID) for the named top-level object. The list is sorted by date. To paginate the results, use the x-emc-token and x-emc-limit headers.
Read	To retrieve the contents of a versioned object, use the standard object API, passing in the OID of the version you want to retrieve.
Restore	Reverts the top-level object to the specified point-in-time copy.
Update	You cannot modify a versioned object. To update the top-level object, use the standard API to perform any updates to the top-level object.

You can version objects in the namespace, but you must reference those objects using the object's OID. You cannot reference them using the namespace path.

Versioned objects with other Atmos features

Table 3 describes how versioned objects work with other Atmos features.

Table 3 Versioning and Atmos Feature Interaction

Atmos Feature	Description
Policy	The policy applied to the top-level object is also applied to the versioned object except for retention or expiration rules.
Replica types	You can create versioned objects when any of the following replicas are defined for the top-level objects: synchronous, asynchronous, striping, GeoParity, compressed and deduplicated.
Security/ACLs	A versioned object is owned by the same UID and is assigned the same ACLs as the top-level object from which it was created.

Restrictions

You can version:

- Top-level objects.
- You cannot create a version of a directory, and you cannot create a version of a versioned object.
- An object by object ID.
- You can version objects in the namespace, but you must reference them by object ID, and not by the namespace path.

You cannot:

- Group objects in collections for version management.
- Manage versions using the standard Atmos API, you must use the version API.
- Use the version feature for consistency groups.
- Use object versioning to provide added data durability or disaster recovery.

Unicode Support

By default, the Atmos REST API uses UTF-8 encoding for Unicode characters, however, the HTTP standard supports only Latin-1 characters in HTTP headers. To support Unicode characters in data sent via an HTTP header, Atmos requires that you use the "x-emc-utf8" header, and percent-encode the Unicode data by following these rules:

- **Requests** — When sending requests that include Unicode characters in headers (such as metadata names or values), percent-encode the Unicode values and pass the x-emc-utf8:true header on the request so that Atmos knows to unencode the characters before it performs any processing. For example, to set the listable metadata to περιοχή=βόρεια (region=north), the request would look like this:

  ```
  x-emc-listable-meta:
  %CF%80%CE%B5%CF%81%CE%B9%CE%BF%CF%87%CE%AE=%CE%B2%CF%8C%CF%81%CE%B5
  %CE%B9%CE%B1
  ```

- **Responses** — Atmos will percent-encode Unicode characters on responses when x-emc-utf8:true is included in the request. The clients that receive the data must then unencode the data. For example, a response to a get user metadata request that includes x-emc-utf8:true will send "key1" and "val one" as "key1=val%20one".
- **Object names** — You can use Unicode characters when naming Atmos objects. Atmos accepts percent-encoded utf8 names. For example, the request to create the object named "images/υπολογιστή.jpg" would look like:

  ```
  POST
  /rest/namespace/images/%CF%85%CF%80%CE%BF%CE%BB%CE%BF%CE%B3%CE%B9%C
  F%83%CF%84%CE%AE.jpg HTTP/1.1
  ```

- **Signatures** — When the client generates the signature, and x-emc-utf8 is being used, it should use the encoded version for each applicable header, for example, use "my%20value" instead of "my value"
- **Characters to separate metadata entries** — The equal character '=' and comma ',' that separate metadata entries should not be encoded. For example: key1=val%20one,key2=val%20two
- Atmos will only encode/unencode characters when the request includes the "x-emc-utf8" header and it is set to true. If the "x-emc-utf8" header is not included (or if it is set to false), Atmos will not perform any encoding/unencoding of characters, and will return specific errors.

The following operations pass data (such as, metadata tag names) and thus might require the use of the "x-emc-utf8" header:

- "Renaming a file or directory in the namespace"
- "Getting system metadata"/"Getting user metadata"
- "Setting user metadata"
- "Listing objects"
- "Listing user metadata tags"
- "Getting listable tags"
- "Creating an object", "Reading an object", and "Updating an object"

Note: All metadata name/values will be percent-encoded, whether or not their encodings represent non-ASCII characters. For example, the space character is a character normally included in percent-encoding, so will be encoded by Atmos.

You can use the "Getting service information" operation to determine if the current version of Atmos supports the x-emc-utf8 header.

Contact your service provider to determine if your service provider supports Unicode characters.

Percent Encoding

To encode a value, you first convert it to the corresponding utf8 byte string, then percent-encode the utf8 encoded value. Percent-encoding represents each byte in the value as a pair of hexadecimal digits, preceded by the '%' symbol.

For example, υπολογιστή would be encoded as:

```
"%CF%85%CF%80%CE%BF%CE%BB%CE%BF%CE%B3%CE%B9%CF%83%CF%84%CE%AE"
```

Table 4 describes how each character is encoded.

Table 4 Example for each character (page 1 of 2)

Character	Unicode point	Hexadecimal	Name
υ	U+03C5	%CF%85	GREEK SMALL LETTER UPSILON
π	U+03C0	%CF%80	GREEK SMALL LETTER PI
ο	U+03BF	%CE%BF	GREEK SMALL LETTER OMICRON
λ	U+03BB	%CE%BB	GREEK SMALL LETTER LAMDA
ο	U+03BF	%CE%BF	GREEK SMALL LETTER OMICRON
γ	U+03B3	%CE%B3	GREEK SMALL LETTER GAMMA
ι	U+03B9	%CE%B9	GREEK SMALL LETTER IOTA

Table 4 Example for each character (page 2 of 2)

Character	Unicode point	Hexadecimal	Name
σ	U+03C3	%CF%83	GREEK SMALL LETTER SIGMA
τ	U+03C4	%CF%84	GREEK SMALL LETTER TAU
ή	U+03AE	%CE%AE	GREEK SMALL LETTER ETA WITH TONOS

Getting better write performance

To get the best possible performance:

- Write as many objects in parallel as possible.
- Use only one request per object for creates and updates. Concurrent access to the same object (on writes) may lead to locking or serialization overhead.
- Use the **Expect: 100-continue** request header when creating or updating namespace objects. This technique is especially useful when using the namespace to create or update data because oftentimes the filename will already exist. By using the expect header, the requesting application can avoid sending data when it would fail. Here's how it works:

  1. The application requesting the create/update sends the expect: 100-continue header in the request, along with the other normal request headers. But they do not send any data.
  2. The system receives the request, validates the headers, and checks to see if an object with that name exists.
  3. If everything is ok, the system sends back the 100-continue response.
  4. If there is an error, (for example the signature is invalid or the object name already exists) the system sends back the normal error response, and the application never sends the request body.
  5. The application will only send data if it gets the 100-continue header back.

CHAPTER 2
Getting started with the Atmos REST API

This chapter includes examples for using the REST API. It includes the following topics:

REST commands

Atmos supports the following HTTP methods:

- POST — Creates objects and sets user metadata and ACLs for specified objects.
- GET — Retrieves object data, including metadata and ACLs.
- HEAD — Corresponds to each GET method. HEAD looks exactly like a GET request except the method name is HEAD instead of GET.

  The response for a HEAD request includes only headers; it does not include a response body. This is useful for ReadObject requests to retrieve the object's user metadata, system metadata, and access-control list but not the object itself.
- PUT — Updates object attributes.
- DELETE — Removes objects and metadata from the system.

Applications written using the Atmos REST API must be coded to handle HTTP1.1 responses. For example, if a request results in Atmos returning a message length greater than 1MB, Atmos uses the Transfer-Encoding:chunked header instead of the Content-Length header.

Object interface examples

This section includes the following object interface examples:

- “Example: Creating an object”
- “Example: Creating an object with non-listable user metadata”
- “Example: Setting (non-listable) user metadata”
- “Example: Creating an object with listable metadata tags”
- “Example: Setting listable metadata tags”

Example: Creating an object

The following example shows how to create 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+3Ini1W+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.

Line #	Description	See...
1	Identifies the command and, where needed, the object being acted on. In this line, `/rest/objects` indicates an object is being referenced; alternately, you can specify a filename.	"Specifying objects/files in REST commands" on page 66
2	Standard HTTP header.	N/A
3	Sets access rights to the object for the specified user IDs (UIDs). In this case, Mary is granted read access and John is granted full access (read and write).	Chapter 4, "Common REST Headers"
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.	Chapter 4, "Common REST Headers"
5	Specifies the type of object being stored.	Chapter 4, "Common REST Headers"
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.	Chapter 4, "Common REST Headers"
7	Sets access rights to the object ID for the user group.	Chapter 4, "Common REST Headers"
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.	Chapter 4, "Common REST Headers"
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.	Chapter 4, "Common REST Headers"
11	Specifies the signature, which is a means for the system to authenticate the UID making the request.	Chapter 4, "Common REST Headers" and "Managing authentication" on page 140

Example: Creating an object with non-listable user metadata

This example builds on the one in "Object interface examples" on page 22, 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+CS6YwJEAWYWvIlIs=
```

Line #	Description	See...
2	Sets the value of non-listable metadata tags. This might be used to trigger a policy. For example, 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 `part1`, with a value of `buy`.	Chapter 4, "Common REST Headers" "Non-listable user metadata" on page 10

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=
```

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 24, 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+CS6YwJEAWYWvIlIs=
```

Line #	Description	See...
2	Sets the value of listable metadata tags. In this example, in the file system, `part8` is a subdirectory of `part7`, and `part7` is a subdirectory of `part4`. A symbolic link to the object, named `quick`, is under `part8`.	Chapter 4, "Common REST Headers" and "Listable user metadata" on page 11

Example: Setting listable metadata tags

This example builds on the one in "Example: Setting (non-listable) user metadata" on page 24, 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=
```

Namespace interface examples

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 (for example, `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/rwMcQh74Pl8W4JkyTJiPZW4=
```

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: onk0Z8dvgqxKk6wDhlDznKrZqfM=
   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 ReadObject 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 Read Object operation, 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 Read 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: vv7reSLatse4u7WxoO7FPSjJCpY=
```

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=

   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
```

Using HTML forms to create and update content

Atmos provides the ability to upload binary content by using HTML forms. The HTML form must include the elements as described in Table 5. If the form includes the set of required elements, Atmos translates the HTML form into a standard REST request, and it treats the form fields as REST headers.

If the form does not meet the requirements, Atmos treats the request as though it were a request to upload multipart/form-data.

Table 5 Form element requirements (page 1 of 2)

Element	Name	Required	Requirement/Description
Form	`action`	Yes	Set to the Atmos endpoint URI for a POST (create) or PUT (update) request. If you specify a value for the x-http-method-override header, then Atmos transforms this URI to the URI appropriate to the HTTP operation specified.
	`method`	Yes	Set to POST.
	`enctype`	Yes	Set to multipart/form-data.
Form fields	`x-emc-uid`	Yes	If a field of this name is not present, then Atmos assumes that the object is actually multipart/form-data and does not perform the other processing described above. For example: `<input type="hidden" name="x-emc-uid" value="6039ac182f194e15b9261d73ce044939/user1"></input>`
	`x-emc-date`	Yes	This field is required by Atmos for all REST operations. For example: `<input type="hidden" name="x-emc-date" value="Thu, 05 Jun 2008 16:38:19 GMT"></input>`
	`x-emc-signature`	Yes	This field is required by Atmos for all REST operations. Calculate the signature for HTML forms in the same way as you do for any Atmos requests, and store them in a form field of this name. For example: `<input type="hidden" name="x-emc-signature" value="WHJo1MFevMnK4jCthJ974L3YHoo="></input>`
	`x-http-method-override`	Optional	Use this field to change the HTTP request verb from POST to PUT if you want to perform an update and not a create. For example: `<input type="hidden" name="x-http-method-override" value="PUT"></input>`

Table 5 Form element requirements (page 2 of 2)

Element	Name	Required	Requirement/Description

	`x-http-inject-response-headers`	Optional	When set to true, the response code and any response headers are injected into the response body.
			The format of this response is one line that contains the length of the headers (integer) followed by HTTP status line and any HTTP headers. For example:
			`128`
			`HTTP 201 Created`
			`Content-Length: 0`
			`Location: /rest/objects/4d773b6ca10574f404d773bd3bedfc04d776693243b8`
			`x-emc-policy: default`
			`x-emc-delta: 502323`
			Note: After this block, the response body, if any, will be sent.

	`x-emc-xxx`	Optional	Where xxx represents a valid Atmos custom header. For example, to set the x-emc-meta header: `<input type="hidden" name="x-emc-meta" value="part1=buy"></input>`
	`input type = file name =data`	Required	This must be the last element in the form. • When input type = file and name = data, the data is streamed as the body of the request. If the name does not = data, an HTTP 400 is returned. • When this form field is encountered, it is assumed that from this point until the end of the request (except for the last boundary) is the content to upload. • The request's Content-Length is modified to this new size. • If this is not true, the Content-Length is incorrect and the Atmos operation fails. • The Content-Type of this field override the request's Content-Type. • The file's encoding must be: Content-Transfer-Encoding: binary. Base-64 is not supported. For example: `<input type="file" name="data"></input>`

Sample form

This example shows how Atmos processes an HTML form that meets the form requirements described in Table 5:

```
<form action="http://10.5.116.244/rest/objects" method="POST" enctype="multipart/form-data">
<input type="hidden" name="x-emc-uid" value="6039ac182f194e15b9261d73ce044939/user1"></input>
<input type="hidden" name="x-emc-date" value="Thu, 05 Jun 2008 16:38:19 GMT"></input>
<input type="hidden" name="x-emc-signature" value="WHJo1MFevMnK4jCthJ974L3YHoo="></input>
<input type="hidden" name="x-emc-meta" value="part1=buy"></input>
<input type="file" name="data"></input>
<input type= "submit"></input>
</form>
```

The resulting POST would look like this:

```
POST /rest/objects HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
   application/x-shockwave-flash, application/xaml+xml,
   application/vnd.ms-xpsdocument, application/x-ms-xbap,
   application/x-ms-application, */*
Accept-Language: en-us
Content-Type: multipart/form-data;
   boundary=---------------------------7db212f1101d8
```

```
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET
   CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30; .NET CLR
   3.0.04506.648; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; MS-RTC LM
   8)
Host: lciga093.lss.emc.com
Content-Length: 306677
Connection: Keep-Alive
Cache-Control: no-cache

-----------------------------7db212f1101d8
Content-Disposition: form-data; name="x-http-method-override"

PUT
-----------------------------7db212f1101d8
Content-Disposition: form-data; name="x-emc-uid"

6039ac182f194e15b9261d73ce044939/user1
-----------------------------7db212f1101d8
Content-Disposition: form-data; name="x-emc-date"

Thu, 05 Jun 2008 16:38:19 GMT
-----------------------------7db212f1101d8
Content-Disposition: form-data; name="x-emc-signature"

WHJo1MFevMnK4jCthJ974L3YHoo=
-----------------------------7db212f1101d8
Content-Disposition: form-data; name="x-emc-meta"

part1=buy
-----------------------------7db212f1101d8
Content-Disposition: form-data; name="data";
   filename="Z:\cwikj\iomega\atmosAppSet_1.0_armel.tgz"
Content-Type: application/x-gzip-compressed

<<<file data>>>
```

Because the request meets the requirements for an HTML form upload, Atmos processes the request as:

```
PUT /rest/objects/ HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
   application/x-shockwave-flash, application/xaml+xml,
   application/vnd.ms-xpsdocument, application/x-ms-xbap,
   application/x-ms-application, */*
Accept-Language: en-us
Content-Type: multipart/form-data
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET
   CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30; .NET CLR
   3.0.04506.648; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; MS-RTC LM
   8)
Host: lciga093.lss.emc.com
Content-Length: 305821
Connection: Keep-Alive
Cache-Control: no-cache
X-Emc-Uid: 6039ac182f194e15b9261d73ce044939/user1
X-Emc-Date: Thu, 05 Jun 2008 16:38:19 GMT
X-Emc-Signature: WHJo1MFevMnK4jCthJ974L3YHoo=
X-Emc-Meta: part1=buy

<<file data>>
```

In the processed request, note that:

- The Content-Type was adjusted to "multipart/form-data".
- The Content-Length was adjusted to the real content length of the file.

Limits/Restrictions/Recommendations

- A POST or PUT request can include a single upload.
- The file element must be the last element in the form.
- EMC recommends that you use the x-http-inject-response-headers so that response headers are injected into the body content of the response. If you do not use this header, you are not able to read response headers from HTML form POST operations.
- For error responses less than 512 bytes, Atmos might pad the response so that the friendly, but uninformative error page used by some browsers is not substituted.
- Atmos might set the "X-Content-Type-Options" response header to "nosniff" to prevent browsers from preemptively changing the content type. For example, if you attempted to upload a file named mypic.gif, some browsers would interpret the response as an image, and try to render it.
- Atmos detects requests from Internet Explorer, and automatically sets an "Expires: 0" response header to prevent caching.
- Form fields are not required to be input type = hidden (as shown in the examples), but they must contain valid values. The hidden input type is useful for preventing users from tampering with these values.

Providing anonymous access

You can let non-authenticated users to upload data to or download data from Atmos by:

- "Using shareable URLs"— Allows anonymous users to download of one file/object. The URL expires at a predetermined time.
- "Using access tokens for anonymous upload and download" — Allows anonymous users to upload or download one file/object as defined by an access token.

Using shareable URLs

Atmos lets you provide a non-Atmos user with a pre-constructed, pre-authenticated URL that lets them download a specific object. The entire object/file is read.

The URL has the following syntax:

```
http://MyAtmosServer/location?uid=uid&expires=expires&signature=signature&disposition=attachment&filename="myfile.txt"
```

Table 6 defines the parameters.

Table 6 URL parameters (page 1 of 2)

Parameter	Description

location	`/objects/object_ID`
	— OR —
	`/namespace/pathname`
	
	For example:
	`/rest/objects/496cbaada2a8bc200496cbb0dd04a004970ce8be68a6`
	
	— OR —
	`/rest/namespace/videos/mycoolvideo`

uid	The UID (and optional subtenant). The UID must have read access to the requested object.
expires	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, expires would be 1235140468. Requests made after this time will fail.
signature	Base64-encoded HMAC-SHA1 of the hash string. See "Calculating the signature" on page 35. The URL is signed using the UID's secret key, to prevent tampering.
disposition	Optional. Use to force the browser to download the content. Use the filename parameter in conjunction with disposition to set the name of the object being downloaded. The content of disposition should be URL-encoded (and UTF-8 filenames should be double-encoded). Atmos will unencode the data once, and the disposition value will be appended to the end of the signature string in lowercase for signing purposes. For backwards compatibility, if disposition is not specified, the signature string will not have anything appended including the typical \n for an empty value. For example: `disposition: attachment` This parameter corresponds to the HTTP Content-Disposition header described in RFC 2183, 5987, and 6266.

Table 6 URL parameters (page 2 of 2)

Parameter	Description

filename	Optional. Use with disposition to set the name attribute of the object being downloaded. Specify a quoted string. For example:
	`filename="filename.txt"`
	
	The following example shows how to specify the filename value when using UTF-8 encoded characters. To download the file named:
	
	6ö°.txt
	
	You'd specify the value like this:
	`disposition: attachment; filename*=UTF-8 "%D0%B1%C3%B6%EF%BD%BC.txt"`
	
	Note: Not all browsers support UTF-8 encoding.
	
	If you are not sure whether your customers are using browsers that support UTF-8, then you can supply filename in both ways. For example:
	`disposition: attachment; filename="no UTF support.txt"; filename*=UTF-8 "%D0%B1%C3%B6%EF%BD%BC.txt"`
	
	When downloading namespace objects, typically set the disposition to attachment without the filename option because the browser can use the file name supplied by the namespace URL.
	When downloading objects from the object space, use filename to supply a more meaningful name for users.

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 141.

HashString is computed as follows:

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

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

If disposition is included in the URL, the hashstring is computed including the disposition like this:

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

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=.
```

Filename parameter and UTF-8 considerations

Newer implementations of URL encoding (based on updated RFCs) encode spaces as "%20". Older implementations might encode the space as a plus sign (+). Java's URLEncoder still encodes spaces as "+". You should verify that your encoding is correct. In Java, the code looks like this:

```
if(disposition != null) {
            disposition = URLEncoder.encode(disposition, "UTF-8");
            // For some reason, Java uses the old-style + encoding for
            // spaces, but Apache expects the newer %20.
            disposition = disposition.replace("+", "%20");
   query += "&disposition=" + disposition;
            }
```

Using access tokens for anonymous upload and download

Atmos provides an API for creating and managing access tokens that enable non-authenticated users to directly upload or download a single Atmos object using a web browser and an HTML form. You can program the HTML form to pass in user metadata and many of the other parameters normally passed in on Atmos Create or Get operations.

The access token defines what the user can do. Specifically, it controls:

- How long the access token can be used.
- The IP addresses the upload or download request can originate from.
- The number of times the token can be used to upload or download content.
- The content upload size (in bytes).
- The valid HTML form field elements that can exist on an upload request including any validation for those fields.

To learn more about managing access tokens see:

Table 7 Access token management operations

Management operation	See
Create	“Creating an access token” on page 69
Delete	“Deleting an access token” on page 75
Get details for one access token	“Getting access token info” on page 82
List all available tokens for a subtenant	“Listing access tokens” on page 100

For more information about anonymous download, see “Downloading content anonymously” on page 81.

About the HTML form for anonymous upload

To upload content using an access token, you use an HTML form displayed in a browser. The HTML form must include:

- **A form declaration** — The form declaration which must meet these requirements:
  - Encoding — UTF-8 encoded.
  - HTTP method— POST.
  - enctype — Set to multipart/form-data (see RFC 1867) for file uploads and text area uploads.
  - form action — Set to:

    ```
    http://<Atmos_endpoint>/rest/accesstokens/<token_id>
    ```

- **One or more form fields** — The form field validation is defined by the access token policy. For more information, see “About the access token policy document” on page 38.

Table 8 describes additional form fields.

Table 8 Form fields

Name	Description	Required?
x-emc-file	Specifies the file to be uploaded to Atmos. • The file content must be the last field in the form. • The user can upload one file at a time. • If this access token points to a directory, the uploaded filename will be created in that directory; otherwise the uploaded file name is ignored. • The Content-Disposition part header requires a specific order. The name parameter must appear before the filename parameter (for example, 'form-data; name="data"; filename="foo.txt"'). Browsers typically provide this order by default.	Yes
x-emc-redirect-url	Specifies the URL the requesting application is redirected to on successful upload. This field must be specified in the token policy or the upload will be rejected. The URL must be absolute and if it can not be interpreted, error 1002 (HTTP 400) is returned. The redirected request will include these query string parameters: • status — true (indicating a successful upload). • tokenId — the token ID. • id — the object ID of the newly created object.	No
x-emc-listable-meta or x-emc-meta	Specifies the non-listable and listable metadata tags to associate with the file/object being uploaded. This field must be specified in the token policy or the upload will be rejected.	No

About the access token policy document

The access token policy defines rules about how a specific access token can be used — whether it can be used only for upload, only for download, or for both. It also defines rules about how the anonymous user must complete the form during an upload request — including field validation rules. If any of the form's field fail the policy evaluation, the upload request fails.

Table 9 describes the access token policy document elements. They must appear in the order shown in the table.

Table 9 Access token policy document elements

Element name	Description
expiration	(Optional). The expiration date of the policy in ISO8601 GMT date format. If not specified, the access token expires 24 hours from the time it was created.
max-uploads	Defines whether the token can be used to upload content. Values are: • 1 — The token can be used to upload content one time. If you specify a value greater than 1, Atmos resets it to 1. • 0 — The token cannot be used to upload content.
max-downloads	Defines the number of times that an unauthenticated user can download a file using the access token. Values are: • 0 — The token cannot be used for downloads (the default). • › 0— The number of times it can be downloaded. **Note:** If a download request returns an HTTP 404 error, the request is counted as a download and will reduce the value of max-downloads.
source	Container for the collection of rules that define the IP addresses where uploads can originate from. Values are: • allow — An IP address or group of addresses in CIDR format from which user can access given access token. • deny — An IP address or group of addresses in CIDR format from which user can not access given access token.
content-length-range	Restricts the upload size to a specific length, in bytes. You must specify the minimum and maximum number of bytes of uploaded content from and to attributes.
form-field elements (zero or more)	The form-field elements in the policy document are used to validate the contents of the uploaded form. Each form-field element in a policy imposes a restriction to a corresponding field of the form using a condition for the form-field element. You must specify one condition for each form field, and you can create more complex matching criteria by specifying multiple conditions for a form field. If an uploaded form contains a field not specified here, it will be rejected. This includes x-emc-meta and x-emc-listable-meta. The only mandatory attribute is name which defines the name of the form field that the restriction applies to. The name element has one mandatory boolean attribute called optional. Use the optional attribute to specify that the field can be absent from a form.

Table 10 describes the condition matches for form fields.

Table 10 Condition matching for form fields

Condition	Description

Exact Matches	Use when a form field must match specific values. In this example, the user must enter log_file_10_20_2011.log in the x-emc-form-filename field:
	`<form-field name="x-emc-form-filename">`
	`<eq>log_file_10_20_2011.log</eq>`
	`</form-field>`

Starts With	Use when the field must start with a certain value. In this example, the x-emc-form-filename must start with log_file.txt:
	`<form-field name="x-emc-form-filename">`
	`<starts-with>log_file_</starts-with>`
	`</form-field>`

Ends With	Use when the field must end with a specific value. In this example, the x-emc-form-filename field must end with .log:
	`<form-field name="x-emc-form-filename">`
	`<ends-with>.log</ends-with>`
	`</form-field`

Contains	Use to specify a string that must be present in the field. In this example, the x-emc-groupacl field must contain the string FULL_CONTROL:
	`<form-field name="x-emc-groupacl">`
	`<contains>FULL_CONTROL</contains>`
	`</form-field>`

Matches	Use to specify a regular expression that will be used to check validity of a field. In this example, the form field x-emc-meta is optional, and it must be populated with a comma-separated list of values.
	`<form-field name="x-emc-meta" optional="true">`
	`<matches>^(\w+=\w+)|((\w+=\w+),(\w+, \w+))$</matches>`
	`</form-field>`

Example

The access token:

- Expires after December 2, 2012.
- Allows 1 upload. Allows 0 downloads. (The policy does not specify this value so it uses the default.)
- Can only be used by the IP 127.0.0./24.
- The upload/download can be in the range of 10 to 11000 bytes.
- The field with name x-emc-redirect-url must be present
- The field x-emc-meta must contain a comma-separated list of key-value pairs.

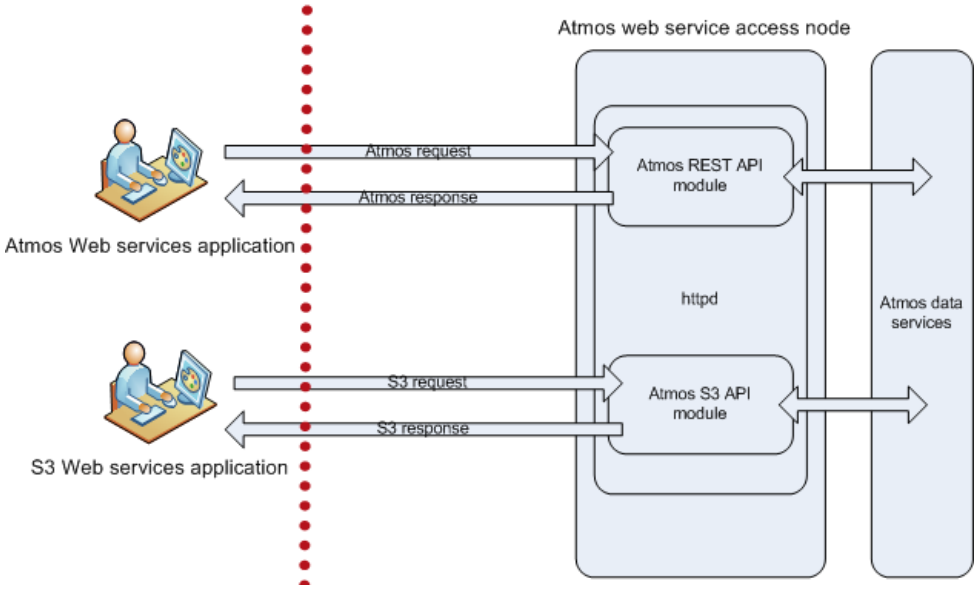
```
<?xml version="1.0" encoding="UTF-8"?>
<policy>
<expiration>2012-12-01T12:00:00.000Z</expiration>
<max-uploads>1</max-uploads>
<source>
<allow>127.0.0.0/24</allow>
</source>
<content-length-range from="10" to="11000"/>
<form-field name="x-emc-redirect-url"/>
<form-field name="x-emc-meta" optional="true">
<matches>^(\w+=\w+)|((\w+=\w+),(\w+, \w+))$</matches>
</form-field>
</policy>
```

CHAPTER 3
Using Amazon S3 Applications with Atmos

Atmos natively supports the Amazon Simple Storage Service (S3) API as an access method for data operations between S3-based applications and Atmos. This chapter describes the Atmos support for Amazon S3 applications. It includes the following topic:

Using S3 with Atmos

Atmos natively supports the most common operations of the Amazon S3 API (version 2006-03-01) through a web services module that runs on Atmos access nodes as shown in Figure 2.



You can find the list of supported S3 operations in Table 11 and the list of unsupported operations in Table 12.

Figure 2 S3 applications running against an Atmos system

The service endpoints for Atmos S3 applications are:

```
http://Atmos_accessnode_IP:8080
https://Atmos_accessnode_IP:8443
```

Existing S3 applications can read and write data to Atmos without requiring additional application development — except that you supply Atmos credentials (UID/shared secret), and not S3 credentials (Access Key ID/Secret Access Key) on the authentication request.

Contact your Atmos system administrator to obtain the Atmos UID/shared secret and the access node IP address.

Table 11 lists the Amazon S3 operations supported by Atmos.

Table 11 Supported S3 operations

Category	Operation
Bucket operations	Delete Bucket
	GET Bucket
	GET Bucket ACL
	GET Bucket Location
	GET Bucket Versioning
	GET Service
	List Multipart Uploads
	PUT Bucket
	PUT Bucket ACL
Object operations	DELETE Object
	DELETE Multiple Objects
	GET Object
	GET Object ACL
	HEAD Object
	POST Object
	PUT Object
	PUT Object ACL
	Multipart Upload – Initiate, Upload, Complete, Abort, List parts

Table 12 lists the unsupported S3 operations.

Table 12 S3 operations not supported by Atmos

Category	Operation
Bucket operations	Delete Bucket CORS
	Delete Bucket lifecycle
	Delete Bucket policy
	Delete Bucket tagging
	Delete Bucket website
	Get Bucket CORS
	Get Bucket lifecycle
	Get Bucket tagging
	Get Bucket policy
	Get Bucket notification
	Get Bucket logging
	Get Bucket requestPayment
	Get Bucket website
	Put Bucket CORS
	Put Bucket lifecycle
	Put Bucket logging
	Put Bucket notification
	Put Bucket policy
	Put Bucket requestPayment
	Put Bucket tagging
	Put Bucket versioning
	Put Bucket website
Object operations	Get Object torrent
	Get Object versions
	Post Object restore

When one of the unsupported operations is used, Atmos return a 501 (Not Implemented) HTTP status code, and the following message: “This S3 operation is currently not implemented.”

S3 Bucket configuration and performance

S3 buckets can be configured in either flat mode or one-to-one mode. Choosing the mode to use depends on whether the keys are hierarchical or not. One-to-one mode is the default.

IMPORTANT

Set the bucket's mode immediately after creating the bucket. If you change the mode after objects have been ingested, those objects become inaccessible.

Flat mode provides the best performance for applications that store millions of keys and the keys are not hierarchical. An example of a flat key is a UUID. Applications that store objects without any hierarchy generally put all of their keys in the root of the bucket without a prefix such as a directory. Flat mode's limitation is that listing buckets using the prefix option is not supported. A key using flat mode looks like this:

```
/bucket/a1e8eb5c-dc2a-4a02-a71b-0fab8bb28c3b
```

One-to-one mode supports applications that treat S3 keys like a traditional directory structure with prefixes separated by slashes. In one-to-one mode, prefix searching performs well as long as you use the standard delimiter ("/"). One-to-one mode's limitation is that, like a directory in Atmos, you are limited to 64,000 objects per directory (prefix). A key in one-to-one mode (or hierarchical mode) looks like this

```
/bucket/articles/2013-09/document234/images/header.jpg
```

The bucket is stored as a directory in the Atmos namespace, you can also optionally change the bucket mode by setting the same metadata on the Atmos directory. Atmos manages bucket mode configuration through the user metadata key called **bucket-mapping-type**. By default, **bucket-mapping-type** is set as one-to-one. You can set or get this value by using PUT/GET bucket-tagging operation.

To programmatically set the bucket-mapping-type to flat, use the PUT bucket-tagging operation like this:

```
PUT /?tagging HTTP/1.1
Host: BucketName.s3.amazonaws.com
Date: date
Authorization: authorization string (see Authenticating Requests (AWS
   Signature Version 4))

<Tagging>
  <TagSet>
     <Tag>
       <Key>bucket-mapping-type</Key>
       <Value>flat</Value>
     </Tag>
  </TagSet>
</Tagging>
```

To determine the bucket's current setting, use the "Getting user metadata" operation. If bucket-mapping-type is not one-to-one, it is set to flat. Otherwise it is set to one-to-one.

Through the Atmos REST API, a bucket's directory path is:

```
/s3/<bucket name>
```

For example, for a bucket called “mybucket”, the directory to modify would be:

```
/s3/mybucket
```

S3 bucket addressing

You can address an S3bucket in these ways:

Bucket style	Bucket address	Example	Works by default with Atmos
Path style bucket	`http://hostname/bucket`	`http://s3.sampleorg.org/samplebucket`	Yes
Virtual host style bucket	`http://bucket.hostname/`	`http://samplebucket.s3.sampleorg.org/`	No

For the virtual host style bucket, the following host names are supported by default:

- s3.amazonaws.com
- localhost
- localhost.localdomain

To enable the virtual host style bucket for additional host names, the DomainNames key in the Atmos S3 configuration file must include the additional host names.

Configuration errors

If the Atmos system is not configured to support your hostname, you might see the errors described in this section.

Missing DNS wildcard entry

When the DNS does not include the wildcard entry, for example: *.s3.domain.com, the DNS resolution of mybucket.s3.domain.com fails, and you get an error like the following:

```
Amazon.S3.AmazonS3Exception: Maximum number of retry attempts reached
   : 3
System.Net.WebException: The remote name could not be resolved:
   'mybucket.s3.domain.com' at System.Net.HttpWebRequest.GetResponse()
   at Amazon.S3.AmazonS3Client.getResponseCallback[T](IAsyncResult
   result)
```

Missing DomainNames key

When the DomainNames key has not been set by the Atmos system administrator, Atmos does not interpret the resource correctly, and you get an error like the following:

```
Amazon.S3.AmazonS3Exception: The specified method is not allowed
   against this resource.
at Amazon.S3.AmazonS3Client.processRequestError(String actionName,
   HttpWebRequest request, WebException we, HttpWebResponse
   errorResponse, String requestAddr, WebHeaderCollection& respHdrs,
   Type t, Exception& cause) at
   Amazon.S3.AmazonS3Client.handleHttpWebErrorResponse(S3Request
   userRequest,WebException we, HttpWebRequest request,
   HttpWebResponse httpResponse, Exception& cause, HttpStatusCode&
```

```
statusCode) at
Amazon.S3.AmazonS3Client.getResponseCallback[T](IAsyncResult
result)
```

CHAPTER 4
Common REST Headers

This chapter describes the common REST headers. For a list of the headers related to specific requests and responses, refer to each operation.

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

Standard HTTP headers

This section describes the standard HTTP headers typically used in Atmos requests/responses. They include:

Content-Length

The length of the request/response body, in bytes. It must be >= 0.

Syntax

```
content-length: <num-bytes>
```

For example:

```
content-length: 211
```

Content-Type

Optional. Used to get and set the content type of the object. The default is application/octet-stream. Any value can be entered here, but only valid HTTP content types are understood when data is retrieved; for example, by a browser.

Syntax

```
content-type: <HTTP content-type>
```

For example:

```
content-type: application/octet-stream
```

Date

(Optional: Date and/or x-emc-date must be in the request.)

Date in UTC format, as defined in RFC 2616, section 3.3.1; for example, Thu, 31 Jan 2008 19:37:28 GMT. Many HTTP clients set this header automatically.

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 x-emc-date header takes priority over this header. The Web server first checks for the x-emc-date header and uses its timestamp. If the x-emc-date header is not present, the Web server checks for the Date header and uses its timestamp.

This date also is used for signature computation; see "REST authentication: securing REST messages with signatures" on page 141.

Syntax

```
Date : date_in_UTC_format
```

See Also

"x-emc-date"

Expect

Optional. May be used with the 100-continue expectation.

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.

Syntax

For example:

```
Expect: 100-continue
```

See Also

"Getting better write performance" on page 19

Location

Included in responses when creating objects or creating versions. It contains the object ID for the newly created objects.

Syntax

```
Location: path/objectID
```

For example:

```
location: /rest/objects/4d773b6ca10574f404d773bd3bedfc04d776693243b8
```

Range

When *updating* 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.

For *reading* 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."

Range header formats:

- **Format:** bytes=first-last
  - **Example:** range: bytes=10-20
  - **Description:** From first byte index to last byte index, inclusive.
  - **Use:** Reading or updating an object

- **Format:** bytes=first-
  - **Example:** range: bytes=10-
  - **Description:** From first byte index until the end of the object (for example, object size - 1)
  - **Use:** Reading an object
- **Format:** bytes=-length
  - **Example:** range: bytes=-30
  - **Description:** The last `length` bytes.
  - **Use:** Reading an object

Syntax

```
Range: Bytes=begin_offset-end_offset
```

See Also

HTTP 1.1 specification at
http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.35

Atmos custom headers

The headers in this section are proprietary to the Atmos REST API. They include:

x-emc-accept

Optional. Use with a list objects request to define the format of the returned object IDs list. Values can be:

- text/plain— Returns the object list as plain text. Each object ID is returned on a separate line separated by the \n. Use this option if you want better performance (because the response body is smaller), and do not want Atmos to return any metadata associated with the objects. Requesting metadata when this header is present results in an invalid argument error (HTTP code 1002).

- text/xml — (default). Returns the object list as an XML document. Use this option if you want to request object metadata with the object list.

Syntax

```
x-emc-accept:text/xml |text/plain
```

For example:

```
x-emc-accept:text/xml
```

See Also

"Listing objects", "x-emc-include-meta"

x-emc-auth-ver

Optional. Makes the URI signature algorithm case sensitive.

By default, the Atmos signature algorithm is case insensitive. As a result, two different objects - for example, one with file name like 'FiLe naMe' and the other with a file name like 'file name' - may have the same signature. Making the signature algorithm case sensitive avoids this problem. Also, making the signature algorithm case sensitive increases security.

The client specifies the **x-emc-auth-ver** header to indicate which authentication version it wants to use.

- If the **x-emc-auth-ver** header is not specified, the case insensitive approach is used, providing backwards compatible. By default, the authentication approach of REST requires no customer application change. Existing REST clients can still work (using the case insensitive authentication version) without any modification
- If **x-emc-auth-ver** header is equal to 1, use case insensitive authentication.
- If **x-emc-auth-ver** header is equal to 2, use case sensitive authentication.

Syntax

```
x-emc-auth-ver:1|2
```

For example:

```
x-emc-auth-ver:2
```

x-emc-date

Optional. Date and/or x-emc-date must be in the request.

Specify the date UTC format, as defined in RFC 2616, section 3.3.1; for example, Thu, 31 Jan 2008 19:37:28 GMT. This is set by the user.

Atmos uses this date 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 `Date` header. The web server first checks for the x-emc-date header and uses its timestamp. If the x-emc-date header is not present, the Web server checks for the Date header and uses its timestamp.

This date also is used for signature computation.

This header is provided because some development frameworks set the standard HTTP Date 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.

Syntax

```
x-emc-date : date_in_UTC_format
```

See Also

"REST authentication: securing REST messages with signatures"

x-emc-delta

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.

Syntax

```
x-emc-delta: <num-bytes>
```

For example:

```
x-emc-delta: 211
```

x-emc-force

(Optional). Specifies whether to overwrite a file or directory when performing a rename operation in the namespace and an object of that type already exists with the specified name.

When not specified, the value is false and the file or directory is not overwritten. In this case the rename operation will fail.

Syntax

```
x-emc-force: true|false
```

For example:

```
x-emc-force:true
```

See Also

"Renaming a file or directory in the namespace", "x-emc-force"

x-emc-force-overwrite

When specified on a POST request, the object will be created if it does not exist, and will be overwritten if it does exist.

Syntax

```
x-emc-force overwrite: true|false
```

For example:

```
x-emc-force overwrite: true
```

x-emc-groupacl

Sets the access rights to this object for the specified user group(s). Valid values are:

- READ
- WRITE
- NONE
- FULL_CONTROL

These values are not case-sensitive when specified but are always returned in uppercase. Only the OTHER group is supported; this applies to everyone other than the object owner.

Syntax

```
x-emc-groupacl:other=permission
```

x-emc-include-meta

(Optional) Use with list object or read of directory object requests. If true, the request returns an object list that includes system and user metadata. Use with list object or read of directory object requests.

When this header is present, the "x-emc-accept" cannot be set to text/plain.

Syntax

```
x-emc-include-meta: true|false
For example:
x-emc-include-meta: true
```

See Also

"Reading an object", "Listing objects", "x-emc-accept"

x-emc-limit

(Optional) Specifies the maximum number of objects that should be returned. The response might include fewer items. If false or not specified, there is no limit.

The Atmos system might impose a limit on the number of items returned to ensure that system performance is not impacted if a very large resultset is requested. For listing versions the upper limit is 4096.

Syntax

```
x-emc-limit: <integer>
```

For example:

```
x-emc-limit: 2
```

See also

"x-emc-token""Listing objects,""Listing versions,", and "Reading an object."

x-emc-listable-meta

Used in requests to set listable metadata tags and their values.

There can be only one of these headers per request, with up to 127 comma-separated, name-value pairs.

If you are using both listable (x-emc-listable-meta) and non-listable (x-emc-meta) metadata tags, the combined total of name-value pairs cannot exceed 127. For example, if you define 50 listable name-value pairs, you have 77 available for use as non-listable tags.

Metadata names and values passed through the REST interface can use any characters from the iso-8859-1 character set.

This header can be used in the following requests: creating an object, updating an object, and setting user metadata. It can also occur in responses for getting user metadata and reading an object.

To use this header in an anonymous upload form, it must be included in the token policy or the upload will be rejected.

Syntax

```
x-emc-listable-meta: tag_name1=value1 [,tag_name2=value2...]
```

See Also

"Creating an object", "Updating an object", "Setting user metadata""Getting user metadata", "Reading an object"

x-emc-listable-tags

Occurs in responses to return listable metadata tags for an object (which are set with the x-emc-listable-meta header).

When writing data in Unicode format, the data must be percent-encoded, and the request must include the "x-emc-utf8" header or the request fails.

Special characters: — If a metadata tag name contains a character that is not in the iso-8859-1 character set, that character is replaced with a mark question mark (?) character for display purposes. For example, consider a metadata tag name βeta (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:

```
x-emc-listable-tags: mykey1, mykey2, ?eta
```

Syntax

```
x-emc-listable-tags: tag_name1 [,tag_name2...]
```

See Also

"x-emc-listable-meta"

x-emc-meta

Used in requests and responses, to set and get non-listable metadata tags and their values.

When used to write data in Unicode format, the data must be percent-encoded, and the request must include the "x-emc-utf8" header or the request fails.

When returned on a create response, it includes the retention and deletion values for the policy applied to the object just created.

Limitation: — There can be only one of these headers per request, with up to 127 comma-separated, name-value pairs. Metadata names and values passed via the REST interface can use any characters from the iso-8859-1 character set.

If you are using both listable (x-emc-listable-meta) and non-listable (x-emc-meta) metadata tags, the combined total of name-value pairs cannot exceed 127. For example, if you define 50 listable name-value pairs, you have 77 available for use as non-listable tags.

To use this header in an anonymous upload form, it must be included in the token policy, or the upload will be rejected.

Syntax

```
x-emc-meta: tag_name1=value1 [,tag_name2=value2...]
```

For example:

```
x-emc-meta: part1=order
```

See Also

"x-emc-listable-meta"

x-emc-objectid

Use when creating an access token for an anonymous download. It must be a valid Atmos object ID. The object must exist. If this header is specified, the x-emc-path cannot also be specified.

Syntax

```
x-emc-objectid: <object ID>
```

For example:

```
x-emc-objectid: 499ad542a1a8bc200499ad5a6b05580499c3168560a4
```

See Also

create access token, "x-emc-path"

x-emc-path

Use when:

- Submitting a request to rename a file or directory — This header specifies the full path to the new directory or file name within the same namespace. If you specify a parent directory that does not exist, the operation creates it.

Note: You cannot use this to move a file or directory to a different namespace.

- Creating an access token for use with anonymous upload or download — This header must be a valid Atmos namespace path to the location where the access token uploads or downloads content. The value can be a folder or a full path that includes the file name. If the path is a directory, it must end with a forward slash '/' and an uploaded file will be created under that directory. On an upload operation, Atmos does not create any directories that do not already exist. If this header is specified, the x-emc-objectid cannot also be specified. When not specified, the object interface is assumed. On an upload operation, if an object of the same name or Object ID exists, Atmos returns an already exists error — it does not overwrite it.

If the data passed on this header is in Unicode format, you must also pass the "x-emc-utf8" header on the request.

Syntax

```
x-emc-path: <path-expression>
```

For example:

```
x-emc-path: full/path/to/new/name
```

See Also

"Renaming a file or directory in the namespace", "x-emc-force", create access token

x-emc-policy

Occurs in all responses. The value depends on the type of request:

- For requests that deal with the actual content of an object (for example, creating, deleting, reading, and versioning an object), the value is the name of the policy applied to the object.
- For other operations (for example, metadata or ACL operations), the value of x-emc-policy is set to the reserved word _int.

Syntax

```
x-emc-policy: <policy-name| _int>
```

For example:

```
x-emc-policy: default
```
or
```
x-emc-policy: _int
```

x-emc-redirect-url

(Optional.) When using the anonymous upload feature, this is the absolute URL the requesting application is redirected to on successful upload. This field must be specified in the token policy or the upload is rejected.

The redirected request includes these query string parameters:

- status — true (indicating a successful upload).
- tokenId — the token ID.

- id — the object ID of the newly created object.

If x-emc-redirect-url is not specified, upon success a 201 HTTP response code is returned and the object id is returned in the Location header.

Table 13 describes the HTTP codes returned for various error conditions when this header is not specified.

Table 13 HTTP codes returned if x-emc-redirect-url is not specified

HTTP Code	Condition
301	The content were successfully uploaded.
500	A server-side error occurred.
400	The URL cannot be interpreted.
403	An authentication error occurred.

Syntax

```
x-emc-redirect-url: http://<IPAddress>/<pagename.htm>
```

x-emc-signature

Use to authenticate the UID making the request.

See "REST authentication: securing REST messages with signatures" on page 141 for details on constructing this header.

Syntax

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

x-emc-system-tags

(Optional) Use to specify system metadata tags to be returned for each object that is retrieved. Can be used in combination with "x-emc-user-tags".

If the data passed on this header is in Unicode format, you must also pass the "x-emc-utf8" header on the request.

Syntax

```
x-emc-system-tags: tag_name1 [,tag_name2...]
```

For example:

```
x-emc-system-tags: atime, size
```

See Also

"Reading an object", "Listing objects", "x-emc-user-tags"

x-emc-tags

Use in requests to retrieve user metadata or system metadata by tag name.

Occurs in responses to get non-listable metadata tags for an object (which are set with the "x-emc-meta" header).

This header has the following limits:

- Some operations accept only one tag name; others accept multiple tag names, separated by commas. For correct usage, see the documentation for the operation.
- There can be only one of these headers per request or response.
- For limits on the character set, see the description of special characters on page "Special characters:" on page 56.

Syntax

```
x-emc-tags: tag_name1 [,tag_name2...]
```

For example:

```
x-emc-tags: color
```

See Also

"Listing objects", "x-emc-meta"

x-emc-token

(Optional) When present in a response, this header indicates that more data exists than was returned, and it provides an identifier to use to retrieve the next item. You use the identifier in a subsequent request to specify the item where data retrieval should start for the next (page) set of results.

This header might be returned in the response headers at any time when using "Listing objects", "Listing versions", or when "Reading an object" to return a list of directories.

When x-emc-token is not returned in the response, there are no more results.

If x-emc-token is specified and "x-emc-limit" is set to 0, all objects from that point on are requested.

Note: The x-emc-token is used to maintain state and should not be interpreted.

If the object that the x-emc-token points to is no longer indexed under the given tag, (either because the object has been deleted or because it's listable metadata has been removed), the operation might fail with the 1037 error code.

Syntax

```
x-emc-token: <token>
```

For example:

```
x-emc-token: file3
```

See Also

"Getting listable tags", "Listing objects", "Listing versions", "Reading an object", "x-emc-limit"

x-emc-uid

Use to specify the UID (user_id) of an application that is using the API, and the subtenant (subtenant_id) to which the UID (user_id) belongs.

If the subtenant ID is missing, Atmos uses the default subtenant of the tenant who has access to the node where the REST call is made. Only one UID is allowed per request. The shared secret associated with this UID is used for signature generation; see "Managing authentication" on page 140.

Syntax

```
x-emc-uid: subtenant_id/user_id
```

x-emc-unencodable-meta

Occurs only in responses. Specifies a list of metadata tags that have names and/or values that are unencodable for REST.

Character set limits. For more on limits on the character set, see the description of Special Characters on page 56.

Syntax

```
x-emc-unencodable-meta: tag_name1 [,tag_name2...]
```

x-emc-user-tags

(Optional) Use in list object, read object, read directory requests to specify the selected user metadata tags to be returned as key/value pairs for each object retrieved.

Can be used in combination with "x-emc-system-tags".

If the data passed on this header is in Unicode format, you must also pass the "x-emc-utf8" header on the request.

Syntax

```
x-emc-user-tags: <tag-name>
```

For example:

```
x-emc-user-tags: state,color
```

See Also

"Listing objects", "Reading an object", "x-emc-system-tags"

x-emc-useracl

Use to set the access rights to an object for the specified UID(s) or access token. Valid values are:

- READ
- WRITE
- NONE

- FULL_CONTROL;

These values are not case sensitive when specified but always are returned in uppercase.

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.

Syntax

```
x-emc-useracl: uid1=permission [,uid2=permission...]
```

x-emc-utf8

Use for:

- On requests to notify Atmos that header data is in Unicode format and has been percent-encoded. (HTTP restricts header data from using Unicode).
- On metadata create requests where the metadata values include more than one equal sign (=) or a comma (,).

When Atmos receives requests with this header, Atmos will:

- Unencode the percent-encoded header data before processing the request.
- Re-encode the data it returns in the response.

Client applications that submit headers that contain data in Unicode format or the special characters (comma or more than one equal sign) must percent-encode the data before submitting the request, and unencode the returned data.

If this header is present, and Atmos receives Unicode data that is not percent-encoded, then

If this header is not present, but Atmos receives header data that includes non-Latin characters, Atmos adds the header name to a a list of unencodable items, for example: "x-emc-unencodable-meta: objname, mdkey1, ..".

Consider using the "x-emc-utf8" header with the following operations if your applications use data in Unicode format:

- "Renaming a file or directory in the namespace"
- "Setting user metadata"
- "Getting user metadata"
- "Getting system metadata"
- "Listing objects"
- "Listing user metadata tags"
- "Getting listable tags"
- "Creating an object", "Reading an object", and "Updating an object"

Because Atmos will percent-encode the responses, make sure the client applications receiving the responses are expecting to unencode them

Note: All metadata name/values will be percent-encoded, whether or not their encodings represent non-ASCII characters. For example, the space character is normally percent-encoded, so Atmos will percent-encode it.

Syntax

```
x-emc-utf8
```

x-emc-wschecksum

Use in create or update requests when:

- Your application must conform to SEC 17a-4f standards.
- You want end-to-end checksum protection of GeoParity replicas.

This header occurs in response documents in the following circumstances:

- When the create or update request is successful, the x-emc-wschecksum header is returned to the requesting program with the same values sent with the request. If the create or update request is not successful, the response does not include this header.
- When performing an object read (via HTTP GET and HEAD methods).
- When performing a GET System Metadata request.

Client applications are responsible for performing checksum verifications on object reads. The values are case-sensitive.

Syntax

```
x-emc-wschecksum: algorithm/offset/checksumValue
```

where:

- *algorithm* — Represents the hashing algorithm used. Valid values: SHA0, SHA1, or md5.
- *offset* — The offset at which the checksum was calculated.
- *checksumValue* — The hash of the object the user is creating or updating.

See Also

"Creating an object", "Updating an object"

CHAPTER 5
REST API Reference

This chapter describes the Atmos REST operations that act on objects and metadata.

Specifying objects/files in REST commands

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

To create an object 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.

To delete, update, read, or version an object, include the object ID (if you use the object interface) or filename (if you use the namespace interface).

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 the 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, Atmos 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.

Namespace file name rules

The characters allowed in file names are governed by both Atmos and HTTP URI rules.

- Atmos allows any character in the printable ASCII character set in a filename, including ? and @.
- HTTP Request URIs allow: A-Z and a-z, 0-9, and the following special characters:

hyphen (-)	period (.)	underscore (_)	tilde (~)
exclamation point (!)	dollar sign ($)	ampersand (&)	double quotes ("'")
parentheses (())	asterisk (*)	plus sign (+)	comma (,)
semi-colon (;)	equal sign (=)	colon (:)	

For HTTP request URIs all other ASCII characters must be URL-encoded (also referred to as percent-encoded). For example, the space character is an ASCII character that must be encoded. The representation of the space as an URL-encoded character is %20.

URL-encoding is only required for the Request URI in the HTTP request itself. Do not URL-encode characters when computing the HashString to sign the request for the CanonicalizedResource.

Suppose you request the file `pictures/my profile picture`. You would encode the file name `pictures/my profile picture` as `pictures/my%20profile%20picture`. But since the CanonicalizedResource should not be URL-encoded, the HashString would look similar to:

```
GET
application/octet-stream
Wed, 16 Dec 2009 21:15:51 GMT
/rest/namespace/pictures/my profile picture
x-emc-date:Wed, 16 Dec 2009 21:15:51 GMT
x-emc-uid:47cadb22de2e46328e49bafc02f64637/user1
```

Because the path (filename) in the Request-URI must be URL-encoded, the request would look similar to:

```
GET /rest/namespace/pictures/my%20profile%20picture HTTP/1.1
date: Wed, 16 Dec 2009 21:15:51 GMT
x-emc-date: Wed, 16 Dec 2009 21:15:51 GMT
x-emc-uid: 47cadb22de2e46328e49bafc02f64637/user1
x-emc-signature: W6rNZOSD7YMWaUEOHW6jNqIVYCg=
```

REST commands

Atmos supports these methods:

- **POST** — Creates objects, creates versions of existing objects, and sets user metadata and ACLs for specified objects.
- **GET** — Retrieves object data, including metadata and ACLs.
- **HEAD** — 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** — Updates object attributes.
- **DELETE** — Removes 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.

Table 14 Data Management Operations

HTTP Method	Operation	URI

POST	"Creating an object"	`/objects`
		— OR —
		`/namespace/pathname`

	"Renaming a file or directory in the namespace"	`/namespace/pathname?rename`

	"Setting an ACL"	`/objects/objectID?acl`
		— OR —
		`/namespace/pathname?acl`

GET/ HEAD	"Getting an ACL"	`/objects/objectID?acl`
		— OR —
		`/namespace/pathname?acl`

	"Getting object info"	`/objects/objectid?info`
		— OR —
		`/namespace/pathname/myfile?info`

	"Listing objects"	`/objects`

	"Reading an object"	`/objects/objectID`
		— OR —
		`/namespace/pathname`

PUT	"Updating an object"	`/objects/objectID`
		— OR —
		`/namespace/pathname`

DELETE	"Deleting an object"	`/objects/objectID`
		— OR —
		`/namespace/pathname`

Table 15 Service Operations

HTTP Method	Operation	URI
GET/HEAD	"Getting service information"	`/rest/service`

Table 16 Versioning Operations

HTTP Method	Operation	URI
POST	"Creating a version" on page 74	/rest/objects/<*ObjectID*>?versions
DELETE	"Deleting a version" on page 80	/rest/objects/‹*objectID*›?versions
PUT	"Restoring a version" on page 129	/rest/objects/‹*objectID*›?versions
GET	"Listing versions" on page 111	/rest/objects/‹*objectID*›?versions

Creating an access token

Creates and returns an access token that can be used by anonymous users to upload and download content. Each access token is governed by an access token policy that you specify when you create the access token. The access token policy defines:

Table 17 Access token policy elements

Element	Description
expiration	A specific date when the access token expires. By default, the access token expires 24 hours after it is created.
max-uploads	Defines whether the token can be used to upload content. Values are: • 1 — The token can be used to upload content one time. If you specify a value greater than 1, Atmos resets it to 1. • 0 — The token cannot be used to upload content.
max-downloads	Defines whether the token can be used to download content. Values are: • 0 — The token cannot be used for downloads (the default). • › 0— The token can be used for the specified number of downloads. **Note:** If a download request returns an HTTP 404 error, the request is counted as a download and will reduce the value of max-downloads.
source	Container for the collection of rules that define the IP addresses where uploads can originate from. • allow — An IP address or group of addresses in CIDR format from which user can access given access token. • deny — An IP address or group of addresses in CIDR format from which user can not access given access token.
content-length-range	Defines the content upload size minimum and maximum (in bytes). This must be the entire object size. Appends are not allowed to this object.
form-field elements (zero or more)	Content is uploaded via an HTML form as an HTTP POST operation. The policy must define the validation for each element on the form.

For more information on access token policies, see "Using access tokens for anonymous upload and download" on page 37.

HTTP Method

POST

Object interface URI

```
/rest/accesstokens
```

Request parameters

Required request header:

- "x-emc-uid"
- "x-emc-signature"
- "x-emc-date"

Optional:

- "x-emc-path" or "x-emc-objectid"
- "x-emc-useracl", "x-emc-groupacl" on upload requests, this ACL is applied to the uploaded/created object. The token itself does not have ACLs assigned to it.
- "x-emc-listable-meta" or "x-emc-meta" on upload requests, sets non-listable and listable metadata tags for objects created by this access token.

Request body element (optional)

- policy —An XML document that describes attributes of the access token. If a policy is not specified, Atmos applies default values.

Object interface examples

Request

```
POST /rest/accesstokens HTTP/1.1
accept: */*
x-emc-useracl: john=FULL_CONTROL,mary=READ
date: Wed, 18 Feb 2011 16:03:52 GMT
x-emc-date: Wed, 18 Feb 2011 16:03:52 GMT
host: 192.168.0.1
content-length: 211
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: KpT+3Ini1W+CS6YwJEAWYWvIlIs=
```

Request Body

```
<policy>
<expiration>2011-11-01T23:59:59.000Z</expiration>
<source>
<allow>192.168.0.0/24</allow>
</source>
<form-field="x-emc-form-filename">
<starts-with>log_</starts-with>
</form-field>
</policy>
```

Response

```
HTTP/1.1 201 Created
Date: Wed, 18 Feb 2011 16:03:52 GMT
Server: Apache
Content-Length: 0
Connection: close
Content-Type: text/xml; charset=UTF-8
location:/rest/accesstokens/4ef1ed17a1a8000f04ef1ed776c0f104f16a71e576
   a3
```

See also

"About the access token policy document"

Creating an object

Use this operation to create and populate an object. You can optionally add ACLs and user metadata. (Atmos does not validate user metadata passed in on this request.)

On a successful creation, Atmos:

- Returns the object ID as URI on the "Location" header.
- Automatically generates the object's system metadata.
- Returns any retention or deletion values on the "x-emc-meta" header when the policy applied to the object on create includes these values.

For performance, Atmos recommends that you create and populate the object in one request, but for larger objects, you can consider creating the object and passing a chunk of the data on the create request, then using the update operation (see "Updating an object" on page 134) to append the rest of the data.

For the namespace interface, you can also use this operation to create directories. You can create directories not files directly under the / directory. While creating a file, if you specify a directory that does not exist, Atmos creates it. You can create directories in these ways:

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

For applications that must conform to SEC 17a-4f standards, you must specify the "x-emc-wschecksum" header. When you use this header, you must send the checksum of the entire object that is part of the request. For more information, see "x-emc-wschecksum" on page 63.

If the metadata tags that you pass in this request are Unicode, you must percent-encode the data before submitting the request, and include the "x-emc-utf8" header on the request. Atmos will percent-encode the values that it returns.

Permissions

For the namespace interface, you need write on the directory where you create the object.

HTTP method

POST

Object interface URI

`/rest/objects`

Namespace interface URI

`/rest/namespace/pathname`

Request headers

Required:

- "Content-Length"
- "x-emc-date"or "Date"
- "x-emc-uid"
- "x-emc-signature"
- "x-emc-utf8" (only required if metadata values are in Unicode)
- "x-emc-wschecksum" (required if the application must conform to SEC 17a-4f standards)

Optional:

- To set user metadata tags: "x-emc-listable-meta", "x-emc-meta"
- To set ACLs: "x-emc-groupacl","x-emc-useracl"

Object interface examples

Create an object with listable and non-listable user metadata

This example shows how to create an object with listable and non-listable user metadata and user and group ACLs. The default policy has been modified to include retention and deletion periods so the response includes those values in the "x-emc-meta" header.

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: 21
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: KpT+3Ini1W+CS6YwJEAWYWvIlIs=

object test data here
```

Response

```
HTTP/1.1 201 Created
Wed, 18 Feb 2009 16:03:52 GMT
Server: Apache
x-emc-policy: default
location: /rest/objects/499ad542a1a8bc200499ad5a6b05580499c3168560a4
x-emc-delta: 21
x-emc-meta: user.maui.expirationEnable=NONE,
   user.maui.expirationEnd=NONE, user.maui.retentionEnable=false,
   user.maui.retentionStart=2012-04-10T12:38:56Z,
   user.maui.retentionEnd=2014-06-12T14:40:58Z
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Namespace interface examples

Create an object and a directory

In this example, if the photos directory does not exist, Atmos creates it.

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: 21
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: GTOC1GqFELjMMH9XIKvYRaHdyrk=

object test data here
```

Response

```
HTTP/1.1 201 Created
Date: Wed, 09 Mar 2011 11:37:54 GMT
Server: Apache
x-emc-policy: default
x-emc-delta: 21
location: /rest/objects/499ad542a1a8bc200499ad5a6b05580499c326c2f9
84
x-emc-meta: user.maui.expirationEnable=NONE,
   user.maui.expirationEnd=NONE, user.maui.retentionEnable=false,
   user.maui.retentionStart=2012-04-10T12:38:56Z,
   user.maui.retentionEnd=2014-06-12T14:40:58Z
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Create object using checksum

Request

```
POST /rest/namespace/file1.txt HTTP/1.1
accept: */*
date: Thu, 06 May 2010 16:02:25 GMT
```

```
content-type: application/octet-stream
x-emc-date: Thu, 06 May 2010 16:02:25 GMT
x-emc-uid: f390a44a03bd4a80be49c373c17725f7/user1
x-emc-signature: Or/7Unux65EzA//oBpbSKGWW+4o=
x-emc-wschecksum: sha0/1037/6754eeaad9d752f079dcb9ab224ab716720b9dda
```

Response

```
HTTP/1.1 201 Created
Date: Thu, 06 May 2010 16:11:00 GMT
Server: Apache
location: /rest/objects/4be15814a205737304be158919f49104be2ea14d06a9
x-emc-wschecksum: sha0/1037/6754eeaad9d752f079dcb9ab224ab716720b9dda
x-emc-meta: user.maui.expirationEnable=NONE,
   user.maui.expirationEnd=NONE, user.maui.retentionEnable=false,
   user.maui.retentionStart=2012-04-10T12:38:56Z,
   user.maui.retentionEnd=2014-06-12T14:40:58Z
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Creating a version

Creates a point-in-time copy of the referenced object. Returns the object ID of the versioned object in the location header. If the object ID of the referenced object is open for writing, the create version operation fails.

The create request must meet the following requirements:

- The object being versioned must be a top-level, mutable object. It cannot be a version, and it cannot be a directory.
- The object being versioned must be referenced by its object ID, and not its namespace path.

Permissions

Write permissions to the object being versioned (the top-level object).

HTTP method

POST

Object interface URI

/rest/objects/‹*ObjectID*›?versions

Namespace interface URI

Not supported

Request headers

Required:

- “x-emc-date”or “Date”
- “x-emc-signature”

- "x-emc-uid"

Object interface examples

Request

```
POST
   /rest/objects/491abe33a105736f0491c2088492430491c5d0d67efc?versions
   HTTP/1.1
accept: */*
date: Thu, 13 Nov 2008 16:59:59 GMT
content-type: application/octet-stream
x-emc-date: Thu, 13 Nov 2008 16:59:59 GMT
host: 168.159.116.51
x-emc-uid: 6039ac182f194e15b9261d73ce044939/user1
x-emc-signature: krsCbUPZexw5AM2ZnBfd9pjtDHM=
```

Response

```
HTTP/1.1 201 Created
Date: Thu, 13 Nov 2008 16:59:59 GMT
Server: Apache/2.0.63 (rPath)
location: /rest/objects/491abe33a105736f0491c2088492430491c5d0f0daa8
x-emc-delta: 7584
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Deleting an access token

Removes the specified access token.

HTTP Method

DELETE

Object interface URI

```
/rest/accesstokens/<token_id>
```

Request parameters

Required headers:

- "x-emc-uid"
- "x-emc-signature"
- "x-emc-date"

Object interface examples

Request

```
DELETE /rest/accesstokens/499ad542a1a8bc200499ad5a6b05580499c3168560a4
   HTTP/1.1
accept: */*
date: Wed, 18 Nov 2009 14:02:00 GMT
x-emc-date: Wed, 18 Feb 2009 16:03:52 GMT
```

```
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: KpT+3Ini1W+CS6YwJEAWYWvIlIs=
```

Response

```
HTTP/1.1 204 No Content
Date: Wed, 18 Nov 2011 14:02:00 GMT
Server: Apache
Content-Length: 0
Connection: close
Content-Type: text/xml; charset=UTF-8
x-emc-policy: default
```

Deleting an object

Deletes the specified object and its associated metadata.

Permissions

For the namespace interface, you need write (execute) on the parent directory of the object to remove. For the object interface, you need write on the object.

HTTP method

DELETE

Object interface URI

`/rest/objects/<objectID>`

Namespace interface URI

`/rest/namespace/pathname`

Request headers

Required:

- "x-emc-date" or "Date"
- "x-emc-signature"
- "x-emc-uid"

Object interface examples

Request

```
DELETE /rest/objects/499ad542a2a8bc200499ad5a7099940499c3e6fbbcc3
   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: AHnsdoK6vmIEP8mt97O8S8j7TKY=
```

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 examples

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

Deletes user metadata (listable or non-listable) for the specified object. Pass the name of the metadata tag to delete on the "x-emc-tags" on header in the request. To delete more than one tag, pass the tag names in a comma-separated list.

If the request does not include the "x-emc-tags" header, the operation returns an error.

You cannot directly delete or modify system metadata.

Permissions

For both the namespace interface and the object interface, you need write on the object.

HTTP method

DELETE

Object interface URI

`/rest/objects/<objectID>?metadata/user`

Namespace interface URI

```
/rest/namespace/<pathname>?metadata/user
```

Request headers

Required:

- "x-emc-tags"
- "x-emc-date"or "Date"
- "x-emc-signature"
- "x-emc-uid"

Optional

- "x-emc-utf8" (if the user metadata tag name/value pairs are in Unicode format).

Object interface examples

- "Delete multiple tags"
- "Delete request failure"
- "Delete Unicode metadata tags"

Delete multiple tags

This request example deletes the user metadata tags state and color.

Request

```
DELETE
   /rest/objects/4dc19958a10574f404dc199e64fc7204dcbdf1e02269?metadata
   /user HTTP/1.1
accept: */*
date: Thu, 12 May 2011 14:31:15 GMT
content-type: application/octet-stream
x-emc-date: Thu, 12 May 2011 14:31:15 GMT
x-emc-tags: state,color
host: 10.238.112.140:1234
x-emc-uid: 66371ac3bd8148348c0f3f1545e2da69/test-uid
x-emc-signature: CV/gR6WIcPH9ug3TucLBpNbpdpg=
```

Response

```
HTTP/1.1 204 No Content
Date: Thu, 12 May 2011 14:31:18 GMT
Server: Apache
x-emc-policy: _int
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Delete request failure

This request does not pass in the "x-emc-tags" header on the request, so it fails.

Request

```
DELETE
   /rest/objects/4dc19958a10574f404dc199e64fc7204dcbdf1e02269?metadata
   /user HTTP/1.1
accept: */*
date: Thu, 12 May 2011 14:38:27 GMT
content-type: application/octet-stream
x-emc-date: Thu, 12 May 2011 14:38:27 GMT
host: 10.238.112.140:1234
x-emc-uid: 66371ac3bd8148348c0f3f1545e2da69/test-uid
x-emc-signature: 1OGkLeGt8QNA55Xh12Ck0J/c4GA=
```

Response

```
HTTP/1.1 400 Bad Request
Date: Thu, 12 May 2011 14:38:31 GMT
Server: Apache
Content-Length: 145
Connection: close
Content-Type: text/xml
<?xml version='1.0' encoding='UTF-8'?>
<Error>
   <Code>1002</Code>
   <Message>One or more arguments in the request was invalid.</Message>
</Error>
```

Delete Unicode metadata tags

This examples shows how to delete the metadata tag χρώμα (color). The "x-emc-tags" value is percent-encoded, and the "x-emc-utf8" (true) is included because of the Unicode value.

Request

```
DELETE
   /rest/objects/4ef49feaa106904c04ef4a066e778104f071a5ff0c85?metadata
   /user HTTP/1.1
date: Fri, 06 Jan 2012 16:50:31 GMT
content-type: application/octet-stream
x-emc-date: Fri, 06 Jan 2012 16:50:31 GMT
x-emc-tags: %CF%87%CF%81%CF%8E%CE%BC%CE%B1
x-emc-utf8: true
x-emc-uid: 071464e3e8fb4cce8609a623fd9df025/user1
x-emc-signature: zFBmEK/zLzvBQ9cH1ZX+015vXQU=
```

Response

```
HTTP/1.1 204 No Content
Date: Fri, 06 Jan 2012 16:50:31 GMT
Server: Apache
x-emc-policy: _int
x-emc-utf8: true
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8

DELETE
application/octet-stream

Fri, 06 Jan 2012 16:50:31 GMT
/rest/objects/4ef49feaa106904c04ef4a066e778104f071a5ff0c85?metadata/us
   er
x-emc-date:Fri, 06 Jan 2012 16:50:31 GMT
```

```
x-emc-tags:%CF%87%CF%81%CF%8E%CE%BC%CE%B1
x-emc-uid:071464e3e8fb4cce8609a623fd9df025/user1
x-emc-utf8:true
```

Namespace interface examples

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
```

Deleting a version

Deletes a specific version of the object, and returns capacity to the system once the delete is successful. Returns an HTTP 204 — No Content error code. Once a version is deleted, it's object ID is no longer returned by the "Listing versions" operation.

This operation does not delete the top-level object. To delete the top-level object, use the standard delete object. API.

Permission

Write permission to the top-level object

HTTP method

DELETE

URI

/rest/objects/‹*objectID*›?versions

Request headers

Required:

- "x-emc-date"or "Date"
- "x-emc-signature"
- "x-emc-uid"

Object interface examples

Request

```
DELETE
   /rest/objects/491abe33a105736f0491c2088492430491c5d0f0daa8?versions
   HTTP/1.1
accept: */*
date: Thu, 13 Nov 2008 17:00:03 GMT
content-type: application/octet-stream
x-emc-date: Thu, 13 Nov 2008 17:00:03 GMT
host: 168.159.116.51
x-emc-uid: 6039ac182f194e15b9261d73ce044939/user1
x-emc-signature: 29AQTcYe428b0p0I/xI2X9oJyfM=
```

Response

```
HTTP/1.1 204 No Content
Date: Thu, 13 Nov 2008 17:00:04 GMT
Server: Apache/2.0.63 (rPath)
x-emc-delta: -7584
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Downloading content anonymously

Allows anonymous users to download content from Atmos using a browser if they have a valid access token. The Atmos content is returned in the response and includes the content-disposition header. When the namespace interface is used, the content-disposition filename is the namespace path. When the object interface is used the filename is the object ID.

The number of times the file can be downloaded is determined by the access token's policy. If a maximum number is not specified, you cannot download content.

HTTP Method

GET

Object interface URI

/rest/accesstokens/‹token_id›

Namespace interface URI

/rest/accesstokens/‹token_id›

Object interface example

Request

```
GET /rest/accesstokens/224e6a7b98104dddb3f3d650f1105476/
4ef1ed17a1a8000f04ef1ed776c0f104f15bef991f91
HTTP/1.1
accept: */*
date: Wed, 18 Feb 2011 16:03:52 GMT
host: 192.168.0.1
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2011 16:03:52 GMT
Content-Disposition: attachment; filename=log_10_11_2011.log
Server: Apache
Content-Length: 23049
Connection: close
```

Getting access token info

Returns an XML document that describes the specified access token.

HTTP Method

GET

Object interface URI

```
/rest/accesstokens/<token_id>?info
```

Request parameters

Required:

- "x-emc-uid"
- "x-emc-date"
- "x-emc-signature"

Optional:

- "x-emc-token"
- "x-emc-limit"

Object interface examples

Request

```
GET
   /rest/accesstokens/T2lbAqGoAB4E9py78d+fBPmS5HT19jx+a5E3oEk7j7Mg3AVi
   GaY?info HTTP/1.1
Accept-Encoding: identity
X-Emc-Signature: u3rwJdh2NRnsB2XMlMldzt4qEm8=
Connection: close
User-Agent: Python-urllib/2.7
Host: 192.168.0.30:80
Date: Thu, 26 Apr 2012 11:19:29 GMT
X-Emc-Uid: 3c7e6b9137a0493b8fb320dc056219a6/test
X-Emc-Date: Thu, 26 Apr 2012 11:19:29 GMT
```

Response header

```
HTTP/1.1 200 OK
Date: Thu, 26 Apr 2012 11:19:28 GMT
Server: Apache
Content-Length: 318
Connection: close
```

Response body

```
Content-Type: text/xml
<access-token>
<access-token-id>T2lbAqGoAB4E9py78d+fBPmS5HT19jx+a5E3oEk7j7Mg3AViGaY</
   access-token-id>
<expiration>2012-04-27T11:15:19+0000</expiration>
<max-uploads>1</max-uploads>
<max-downloads>-1</max-downloads>
<content-length-range from="0" to="20971520"/>
</access-token>
```

Table 18 describes the XML elements of the response document.

Table 18 Response body elements

Element	Description
access-token	Container element for the details of a specific access token.
access-token-id	The identifier for a specific access token.
expiration	The expiration date of the access token's policy. In ISO8601 format.
max-uploads	The maximum number of times that the same token can be used for uploading a file.
max-downloads	The maximum number of times the same token can be used to download a file.
source	Container element for the set of rules that define where uploads can come from.
source/allow	An IP address or group of addresses in CIDR format from which user can access given access token.
source/deny	An IP address or group of addresses in CIDR format from which user can not access given access token.
content-length-range	The minimum and maximum size of uploaded content (in bytes).
form-field	Form field validation for uploads. See "About the access token policy document" on page 38.
path	The namespace path of the target object for the access token.
object-id	The object ID of the target object for the access token.
useracl	The user ACL for uploaded objects. See the response examples in "Getting an ACL" on page 83.
groupacl	The group ACL for uploaded objects. See the response examples in "Getting an ACL" on page 83.

Getting an ACL

Returns the ACL details associated with the specified object ID.

Permissions

Any UID within the same subtenant can perform this operation.

HTTP method

GET

Object interface URI

`/rest/objects/objectID?acl`

Namespace interface URI

`/rest/namespace/pathname?acl`

Request parameters

Required:

- "x-emc-date" or "Date"
- "x-emc-signature"
- "x-emc-uid"

Object interface examples

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 examples

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

Retrieves a user's listable tags. Use the "x-emc-tags" header to specify the tags to retrieve. The listable tags are returned as a comma-separated list on the "x-emc-listable-tags" header.

To get:

- All top-level listable tags, omit the "x-emc-tags" header on the request.
- The tags within a hierarchy, specify the tag name (including path) within the "x-emc-tags" header.

If the response includes the "x-emc-token" header, it means that there might be more tags to retrieve. To request the next set of tags, pass the value of the "x-emc-token" header in subsequent requests.When the "x-emc-token" header is not included in the response, it means that you have retrieved the full set of tags. For more information, see the "x-emc-token example" on page 86.

Permissions

Any UID within the same subtenant can perform this operation.

HTTP method

GET

Object interface URI

```
/rest/objects?listabletags
```

Namespace interface URI

```
/rest/namespace?listabletags
```

Request parameters

Required:

- "x-emc-date"or "Date"
- "x-emc-signature"
- "x-emc-uid"

Optional:

- "x-emc-tags"
- "x-emc-token"
- "x-emc-utf8" (if the tag name/value pairs are in Unicode format).

Object interface examples

- "Get top-level tags"
- "x-emc-token example"

Get top-level tags

In this example, the request retrieves all of the listable tags that are defined under the top-level tag called continent. They are returned in the response on the "x-emc-listable-tags" header.

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: continent
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:asia, africa, australia, antarctica
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
x-emc-policy: _int
```

x-emc-token example

The following example shows how to use the "x-emc-token" header. The request asks for all of the sub-tags under the pictures/vacation tag specified by the"x-emc-tags" header.

Request 2

```
GET /rest/objects?listabletags HTTP/1.1
date: Fri, 16 Apr 2010 17:15:19 GMT
x-emc-date: Fri, 16 Apr 2010 17:15:19 GMT
x-emc-tags: pictures/vacation
x-emc-uid: f6639b0790634733bdf56e1223908224/user1
x-emc-signature: MSeOcmDQzcJkQIc/iy7NQXmndN0=
```

Response 2

This response includes the "x-emc-token" header to indicate there are more results.

```
HTTP/1.1 200 OK
Date: Fri, 16 Apr 2010 17:15:19 GMT
Server: Apache
x-emc-policy: _int
x-emc-token: 4bb5fa58a1a8482004bb5faf0d12f804bc89a4c5ddb7
```

```
x-emc-listable-tags: boston, newyork, chicago, miami, losangeles,
   sandiego, sanfrancisco, paris, london, rome
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

To continue retrieving the tags under pictures/vacation, include the "x-emc-token" in the subsequent request.

Request 2a

This requests the next set of tags under `pictures/vacation`. It includes the "x-emc-token" with a value of `4bb5fa58a1a8482004bb5faf0d12f804bc89a4c5ddb7` from the previous response.

```
GET /rest/objects?listabletags HTTP/1.1
x-emc-token: 4bb5fa58a1a8482004bb5faf0d12f804bc89a4c5ddb7
date: Fri, 16 Apr 2010 17:15:29 GMT
x-emc-date: Fri, 16 Apr 2010 17:15:29 GMT
x-emc-tags: pictures/vacation
x-emc-uid: f6639b0790634733bdf56e1223908224/user1
x-emc-signature: U8/d6IWL2fa/gfsWPXXSHdM06GM=
```

Response 2a

This response returns the next set of tags. It is also the final set of tags as indicated by the absence of the "x-emc-token" header in the response.

```
HTTP/1.1 200 OK
Date: Fri, 16 Apr 2010 17:15:29 GMT
Server: Apache
x-emc-policy: _int
x-emc-listable-tags: sydney, athens, barcelona, milan, madrid
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Namespace interface examples

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

Returns details about the replicas for an object. Performing this operation on a directory returns error code 1022 because directories do not have storage.

Permissions

Any UID within the same subtenant can perform this operation.

HTTP method

GET

Object interface URI

```
/rest/objects/objectid?info
```

Namespace interface URI

```
/rest/namespace/pathname/myfile?info
```

Request parameters

Required:

- "x-emc-date" or "Date"
- "x-emc-signature"
- "x-emc-uid"

Object interface examples

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:e103f726a87d45abbd8d5f189a8cecfc/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

<?xml version='1.0' encoding='UTF-8'?>
   <GetObjectInfoResponse xmlns='http://www.emc.com/cos/'>
   <objectId>4b00fffea12059c104b00ffca1f8e804b040c4d911
      c9</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 examples

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>4b4502a5a2a8482004b4503232663404b45fe98a5e
      c1</objectId>
   <selection>geographic</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>
```

Table 19 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.
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, ErasureCode for GeoParity replicas, and dedup.
retention	Container element for retention values.
enabled	A Boolean value (true/false) that defines whether retention is enabled for the replica.
endAt	When enabled is true, specifies the dateTime when the data retention period expires. When enabled is false, this element is empty. dateTime has this format: YYYY— year MM—month DD — day hh — hour mm — minute ss — second

Table 19 Response XML Elements

XML Element	Description
expiration	Container element for expiration values.
enabled	A Boolean value that specifies if expiration is enabled (true) or not (false)
endAt	When enabled is true, specifies the dateTime at when the deletion expiration ends. When enabled is false, this element is empty. dateTime has this format: YYYY— year MM—month DD — day hh — hour mm — minute ss — second

Getting service information

Returns the version of Atmos software in use in the following form:

major.minor.servicepack.patch.

For example:

```
2.1.4.0
```

Also lists the features supported by the version on the x-emc-features header and the authentication method (sensitive or case-insensitive) specified by the `x-emc-auth-ver` header.

Permissions

No special permissions required.

HTTP method

GET

URI

/rest/service

Request parameters

Required:

- "x-emc-date" or "Date"
- "x-emc-signature"
- "x-emc-uid"

Examples

Request

```
GET /rest/service HTTP/1.1
Connection: Keep-Alive
Date: Tue, 01 Oct 2013 18:15:21 GMT
Accept: */*
Host: 10.5.116.244
x-emc-date: Tue, 01 Oct 2013 18:15:21 GMT
x-emc-uid: 0e2997b7dc1940eda38270155e2f3136/test
x-emc-signature: mHQZTbVaCdRUKh8ijQRXDJRAhe0=
```

Response

```
HTTP/1.1 200 OK
Date: Tue, 01 Oct 2013 18:15:18 GMT
Server: Apache
x-emc-policy: _int
Content-Length: 140
x-emc-support-utf8: true
x-emc-features: object, namespace, utf-8, browser-compat, versioning
x-emc-auth-ver: 2
Connection: close
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<Service xmlns='http://www.emc.com/cos/'>
<Version>
<Atmos>2.1.5.0</Atmos>
</Version>
</Service>
```

Getting system metadata

Returns the system metadata for the specified object. To get:

- A subset of system metadata, specify the tag names in a comma-separated list on the "x-emc-tags" header on the request.
- All of the object's system metadata, omit the "x-emc-tags" in the request.

If the object was created with a checksum, the "x-emc-wschecksum" header is returned in the response.

If the metadata tags that you pass in this request are Unicode, you must percent-encode the data before submitting the request, and include the "x-emc-utf8" header on the request. Atmos will percent-encode the values that it returns.

To learn more about system metadata tags, see "Example: Creating an object with non-listable user metadata" on page 24.

Permissions

Any UID within the same tenant can perform this operation.

HTTP method

GET

Object interface URI

```
/rest/objects/<objectID>?metadata/system
```

Namespace interface URI

/rest/namespace/‹pathname›?metadata/system

Request parameters

Required:

- "x-emc-date"or "Date"
- "x-emc-signature"
- "x-emc-uid"
- "x-emc-utf8" (only required metadata values are in Unicode.)

Optional:

- "x-emc-tags"

Object interface examples

- "Get all system metadata"
- "Get a subset of system metadata"

Get all system metadata

In this 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

```
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

```
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
```

Get a subset of system metadata

In this example, the "x-emc-tags"s header includes two tags, atime and uid, so only those system-metadata pairs are returned.

Request

```
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

```
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
```

Request metadata for object with Unicode objname

In this example, the objname of the associated object is in Unicode, so the request must include the x-emc-utf8:true header. Atmos percent-encodes the objname.

Request

```
GET
   /rest/objects/4ef49feaa106904c04ef4a066e778104f071a5ff0c85?metadata
   /system HTTP/1.1
date: Fri, 06 Jan 2012 16:32:42 GMT
content-type: application/octet-stream
x-emc-date: Fri, 06 Jan 2012 16:32:42 GMT
x-emc-utf8: true
x-emc-uid: 071464e3e8fb4cce8609a623fd9df025/user1
x-emc-signature: ZQReJ4DrvynvjPv+hQ5B3ZW/Yfk=
```

Response

```
HTTP/1.1 200 OK
Date: Fri, 06 Jan 2012 16:32:42 GMT
Server: Apache
x-emc-policy: _int
x-emc-utf8: true
x-emc-meta: atime=2012-01-06T16:16:00Z, mtime=2012-01-06T15:59:28Z,
   ctime=2012-01-06T16:16:00Z, itime=2012-01-06T15:59:27Z,
   type=regular, uid=user1, gid=apache,
objectid=4ef49feaa106904c04ef4a066e778104f071a5ff0c85,
   objname=%cf%85%cf%80%ce%bf%ce%bb%ce%bf%ce%b3%ce%b9%cf%83%cf%84%ce%a
   e.jpg, size=459, nlink=1, policyname=default
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Request metadata for object with Unicode objname without using x-emc-utf8 header

In this example, the objname of the associated object is in Unicode, but the request does not include the x-emc-utf8:true header. In this case, Atmos does not percent-encode the objname, so the response returns the x-emc-unencodable-meta header with the name of the field (objname) that it was not able to return.

Request

```
GET
   /rest/objects/4ef49feaa106904c04ef4a066e778104f071a5ff0c85?metadata
   /system HTTP/1.1
date: Fri, 06 Jan 2012 16:31:41 GMT
content-type: application/octet-stream
x-emc-date: Fri, 06 Jan 2012 16:31:41 GMT
x-emc-uid: 071464e3e8fb4cce8609a623fd9df025/user1
x-emc-signature: K2uM592a2z9RHBFPL83klmS0U3w=
```

Response

```
HTTP/1.1 200 OK
Date: Fri, 06 Jan 2012 16:31:41 GMT
Server: Apache
x-emc-policy: _int
x-emc-meta: atime=2012-01-06T16:16:00Z, mtime=2012-01-06T15:59:28Z,
   ctime=2012-01-06T16:16:00Z, itime=2012-01-06T15:59:27Z,
   type=regular, uid=user1,
 gid=apache, objectid=4ef49feaa106904c04ef4a066e778104f071a5ff0c85,
   size=459, nlink=1, policyname=default
x-emc-unencodable-meta: objname
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Namespace interface examples

Request

```
GET /rest/namespace/dir561/file14.txt?metadata/system HTTP/1.1
   accept: */*
date: Mon, 05 Jul 2010 19:51:30 GMT
content-type: application/octet-stream
x-emc-date: Mon, 05 Jul 2010 19:51:30 GMT
host: 168.159.116.112:2345
x-emc-uid: ebd858f829114dfabbcf069637a07cfe/user1
x-emc-signature: vMyNLeg/ja208OwCPYlwjMt/MW4=
```

Response

```
HTTP/1.1 200 OK
Date: Mon, 05 Jul 2010 19:51:30 GMT
Server: Apache
x-emc-policy: _int
x-emc-meta: atime=2010-07-05T19:51:19Z, mtime=2010-07-05T19:51:19Z,
   ctime=2010-07-05T19:51:19Z, itime=2010-07-05T19:51:19Z,
   type=regular, uid=user1, gid=apache,
   objectid=4bf520e2a105737304bf52170a4e6204c3237b7c1b16,
   objname=test14.txt4, size=1037, nlink=1, policyname=default
x-emc-wschecksum: sha0/1037/87hn7kkdd9d982f031qwe9ab224abjd6h1276nj9
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Getting user metadata

Returns the user metadata associated with the specified object. To request:

- One or more user metadata tags, pass the tag names on the "x-emc-tags" header. When specifying more than one, supply the tag names in a comma-separated list.
- All user metadata, omit the "x-emc-tags".

Listable metadata is returned on the "x-emc-listable-meta" header, and non-listable metadata is returned on the "x-emc-meta" header.

If the metadata tags that you pass in this request are in Unicode format, you must:

- Percent-encode the data before submitting the request
- Include the "x-emc-utf8" header on the request

Atmos will percent-encode the values that it returns.

Permissions

Read permissions

HTTP method

GET

Object interface URI

```
/rest/objects/<objectID>?metadata/user
```

Namespace interface URI

/rest/namespace/‹pathname›?metadata/user

Request parameters

Required:

- "x-emc-date"or "Date"
- "x-emc-signature"
- "x-emc-uid"
- "x-emc-utf8" (only required when the x-emc-path data is in Unicode format)

Optional:

- "x-emc-tags"

Object interface examples

- "Request non-listable user metadata"
- "Request listable user metadata"
- "Request all user metadata"

- "Request user metadata in Unicode"

Request non-listable user metadata

This example retrieves the user metadata tags, city and state, by passing them on the "x-emc-tags" header. Atmos returns the tags and their values on the "x-emc-meta" header in the response.

Request

```
GET
   /rest/objects/4dc19958a10574f404dc199e64fc7204dcbdf1e02269?metadata
   /user HTTP/1.1
accept: */*
date: Thu, 12 May 2011 13:53:24 GMT
content-type: application/octet-stream
x-emc-date: Thu, 12 May 2011 13:53:24 GMT
x-emc-tags: city,state
host: 10.238.112.140:1234
x-emc-uid: 66371ac3bd8148348c0f3f1545e2da69/test-uid
x-emc-signature: 24kFFIxX8DqRUPv8Ca/n+7KFRUw=
```

Response

```
HTTP/1.1 200 OK
Date: Thu, 12 May 2011 13:53:27 GMT
Server: Apache
x-emc-policy: _int
x-emc-meta: city=boston, state=MA
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Request listable user metadata

This example requests the listable user metadata tag, color, by passing it in on the "x-emc-tags" header. Atmos returns the listable user metadata tag and its value on the "x-emc-listable-meta"in the response.

Request

```
GET
   /rest/objects/4dc19958a10574f404dc199e64fc7204dcbdf1e02269?metadata
   /user HTTP/1.1
accept: */*
date: Thu, 12 May 2011 13:53:04 GMT
content-type: application/octet-stream
x-emc-date: Thu, 12 May 2011 13:53:04 GMT
x-emc-tags: color
host: 10.238.112.140:1234
x-emc-uid: 66371ac3bd8148348c0f3f1545e2da69/test-uid
x-emc-signature: r/ht0y2f+WDeYDpsVJ40JRRVG1Y=
```

Response

```
HTTP/1.1 200 OK
Date: Thu, 12 May 2011 13:53:08 GMT
Server: Apache
x-emc-policy: _int
x-emc-listable-meta: color=blue
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Request all user metadata

This request omits the "x-emc-tags" header so all of the object's user metadata tags are returned. Listable metadata is returned on the "x-emc-listable-meta" header, and non-listable metadata is returned on the "x-emc-meta" header.

Request

```
GET
   /rest/objects/4dc19958a10574f404dc199e64fc7204dcbdf1e02269?metadata
   /user HTTP/1.1
accept: */*
date: Thu, 12 May 2011 13:48:02 GMT
content-type: application/octet-stream
x-emc-date: Thu, 12 May 2011 13:48:02 GMT
host: 10.238.112.140:1234
x-emc-uid: 66371ac3bd8148348c0f3f1545e2da69/test-uid
x-emc-signature: xtzLToS1M6Jg8AInhZsAZi1D2ck=
```

Response

```
HTTP/1.1 200 OK
Date: Thu, 12 May 2011 13:48:06 GMT
Server: Apache
x-emc-policy: _int
x-emc-meta: city=boston, state=ma
x-emc-listable-meta: color=blue
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Request user metadata in Unicode

This request omits the "x-emc-tags" header so all of the object's user metadata tags are returned. Listable metadata is returned on the "x-emc-listable-meta" header, and non-listable metadata is returned on the "x-emc-meta" header.

Because the request includes the "x-emc-utf8" header, Atmos percent-encodes the values returned on the "x-emc-listable-meta" and "x-emc-meta" headers.

Request

```
GET
   /rest/objects/4ef49feaa106904c04ef4a066e778104f071a5ff0c85?metadata
   /user HTTP/1.1
date: Fri, 06 Jan 2012 16:43:03 GMT
content-type: application/octet-stream
x-emc-date: Fri, 06 Jan 2012 16:43:03 GMT
x-emc-utf8: true
x-emc-uid: 071464e3e8fb4cce8609a623fd9df025/user1
x-emc-signature: nwwPnmAVEgjF4ycYtWDULWTlYPk=
```

Response

```
HTTP/1.1 200 OK
Date: Fri, 06 Jan 2012 16:43:03 GMT
Server: Apache
x-emc-policy: _int
x-emc-utf8: true
x-emc-meta: %cf%87%cf%81%cf%8e%ce%bc%ce%b1=%ce%bc%cf%80%ce%bb%ce%b5,
   %ce%bc%ce%ad%ce%b3%ce%b5%ce%b8%ce%bf%cf%82=%ce%bc%ce%b9%ce%ba%cf%81
   %cf%8c
```

```
x-emc-listable-meta:
   %cf%80%ce%b5%cf%81%ce%b9%ce%bf%cf%87%ce%ae=%ce%b2%cf%8c%cf%81%ce%b5
   %ce%b9%ce%b1
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Namespace interface examples

This request omits the "x-emc-tags" header so all of the object's user metadata tags are returned. Listable metadata is returned on the "x-emc-listable-meta" header, and non-listable metadata is returned on the "x-emc-meta" header.

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 access tokens

Returns the set of access tokens available for use by a specific UID.

HTTP Method

GET

Object interface URI

```
/rest/accesstokens
```

Request parameters

Required:

- "x-emc-uid"
- "x-emc-signature"
- "x-emc-date"

Optional:

- "x-emc-token"
- "x-emc-limit"

Object interface examples

Request

```
GET /rest/accesstokens HTTP/1.1
accept: */*
date: Wed, 18 Nov 2011 14:02:00 GMT
x-emc-date: Wed, 18 Feb 2009 16:03:52 GMT
host: 168.159.116.96
x-emc-uid: 33115732f3b7455d9d2344ddd235f4b9/user1
x-emc-signature: KpT+3Ini1W+CS6YwJEAWYWvIlIs=
```

Response header

```
HTTP/1.1 200 OK
Date: Wed, 18 Nov 2011 14:02:00 GMT
Server: Apache
Content-Length: 310
Connection: close
Content-Type: text/xml; charset=UTF-8
```

Response body

```
<?xml version="1.0" encoding="UTF-8"?>
<list-access-tokens-result>
 <access-tokens-list>
  <access-token>
   <access-token-id>4ef1ed17a1a8000f04ef1ed776c0f104f15bef991f92</Acce
   ssTokenID>
   <expiration>2011-12-01T12:00:00.000Z</expiration>
   <max-uploads>1</max-uploads>
   <source>
     <allow>127.0.0.0/24</allow>
   </source>
   <content-length-range from="10" to="11000"/>
  </access-token>
  <access-token>
   <access-token-id>4ef1ed17a1a8000f04ef1ed776c0f104f15bef98f36a</Acce
   ssTokenID>
   <expiration>2012-01-01T12:00:00.000Z</expiration>
   <max-uploads>1</max-uploads>
   <max-downloads>32768</max-uploads>
   <source>
     <allow>192.168.0.0</allow>
   </source>
   <content-length-range from="10" to="11000"/>
  </access-token>
 </access-tokens-list>
</list-access-tokens-result>
```

Table 20 describes the XML elements of the response document.

Table 20 Response body elements (page 1 of 2)

Element	Description
list-access-tokens-result	Container element for the response document.
access-tokens-list	Container element for the collection of access tokens.
access-token	Container element for the details of a specific access token.
access-token-id	The identifier for a specific access token.
expiration	The expiration date of the access token's policy. In ISO8601 format.
max-uploads	The maximum number of times that the same token can be used for uploading a file.
max-downloads	The maximum number of times the same token can be used to download a file.
source	Container element for the set of rules that define where uploads can come from.
source/allow	An IP address or group of addresses in CIDR format from which user can access given access token.
source/deny	An IP address or group of addresses in CIDR format from which user can not access given access token.
content-length-range	Specifies the minimum and maximum size of uploaded content (in bytes).
form-field	form field validation for uploads. See "About the access token policy document" on page 38.

Table 20 Response body elements (page 2 of 2)

Element	Description
path	The namespace path of the target object for the access token.
object-id	The object ID of the target object for the access token.
useracl	The user ACL for uploaded objects. See the response examples in "Getting an ACL" on page 83.
groupacl	The group ACL for uploaded objects. See the response examples in "Getting an ACL" on page 83.

Listing objects

Retrieves all object IDs indexed by a tag. You can specify only one tag name/hierarchy on each operation.

To get:

- No metadata in the response, omit the "x-emc-include-meta" or set it to 0 or false on the request.
- All system and user metadata in the response, set "x-emc-include-meta" to 1 or true.
- A subset of metadata, pass in the tag names on as a comma-separated list on the "x-emc-system-tags" and "x-emc-user-tags". Use "x-emc-utf8" set to true when the "x-emc-system-tags" or "x-emc-user-tags" are in Unicode.

By default, the response contains an XML document listing the object IDs that meet the criteria. Use "x-emc-accept" to specify a different format.

Object IDs are 44 characters long. 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 operation returns the object IDs from a list-objects operation into the XML body, not the header.

Listable tags are created in a user's own namespace, they are private to that user. Only objects belonging to the requesting UID are returned.

Permissions

No special permissions apply.

HTTP method

GET

Object interface URI

```
/rest/objects
```

Namespace interface URI

Not supported

Request parameters

Required:

- "x-emc-date"or "Date"
- "x-emc-signature"
- "x-emc-uid"
- "x-emc-utf8" (only required if the request includes metadata tags in Unicode format).

Optional:

- "x-emc-include-meta"
- "x-emc-system-tags"
- "x-emc-tags"
- "x-emc-user-tags"
- "x-emc-accept"

Object interface examples

This section includes the following examples:

- "List object IDs — No metadata"
- "List objects — Using the x-emc-limit header"
- "List objects — All metadata"
- "List objects — Selected metadata"
- "List objects — with Unicode tag names"

List object IDs — No metadata

This example retrieves object IDs — without metadata by setting the "x-emc-include-meta" header to 0.

Request

```
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: ZllFtIyYe6kvqibS9eqcIBpiQ7I=
x-emc-include-meta: 0
```

Response

```
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>499ad542a2a8bc200499ad5a7099940499b44f779a5
      4</ObjectID>
 </Object>
</ListObjectsResponse>
```

List objects — Using the x-emc-limit header

In this example, 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>4a0d6e22a1a8482004a0d6ecd1247804a0d7337c
         89fd
      </ObjectID>
   </Object>
   <Object>
     <ObjectID>4a0d6e22a2a8482004a0d6ecd85daf04a0d73390
         2a93</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>4a0d6e22a2a8482004a0d6ecd85daf04a0d733b2
         8892</ObjectID>
   </Object>
   <Object>
      <ObjectID>4a0d6e22a1a8482004a0d6ecd1247804a0d733c
         19b14</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>4a0d6e22a2a8482004a0d6ecd85daf04a0d733d
         f3eea</ObjectID>
   </Object>
   <Object>
      <ObjectID>4a0d6e22a2a8482004a0d6ecd85daf04a0d733e
         b2d85</ObjectID>
   </Object>
</ListObjectsResponse>
```

List objects — All metadata

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.

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
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>499ad542a2a8bc200499ad5a7099940499b44f51e97
         d</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>policyname</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>
```

List objects — Selected metadata

This example shows how to get a set of tags using 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>499ad542a2a8bc200499ad5a7099940499b44f51e9
       7d</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>
```

List objects — with Unicode tag names

This example shows how to get the objects with the tag περιοχή (region). Because the tag name is Unicode, the value in "x-emc-tags" is percent-encoded and the "x-emc-utf8" is set to true. The response also includes "x-emc-utf8".

Request

```
GET /rest/objects HTTP/1.1
accept: */*
date: Fri, 06 Jan 2012 16:46:21 GMT
content-type: application/octet-stream
x-emc-date: Fri, 06 Jan 2012 16:46:21 GMT
x-emc-tags: %CF%80%CE%B5%CF%81%CE%B9%CE%BF%CF%87%CE%AE
x-emc-utf8: true
x-emc-uid: 071464e3e8fb4cce8609a623fd9df025/user1
x-emc-signature: ApiTlJQW7gTZ5M5xBa3p01kx7L0=
```

Response

```
HTTP/1.1 200 OK
Date: Fri, 06 Jan 2012 16:46:21 GMT
Server: Apache
x-emc-policy: _int
x-emc-utf8: true
Content-Length: 204
Connection: close
Content-Type: text/xml
<?xml version='1.0' encoding='UTF-8'?>
<ListObjectsResponse xmlns='http://www.emc.com/cos/'>
        <Object>
   <ObjectID>4ef49feaa106904c04ef4a066e778104f071a5ff0c85</ObjectID>
        </Object>
</ListObjectsResponse>
```

Listing user metadata tags

Returns the user metadata tags assigned to the specified object.

Regular user metadata is returned using the "x-emc-tags" header, and listable metadata is returned using the "x-emc-listable-tags" header.

Include "x-emc-utf8" header set to true when the values returned include Unicode characters.

Permissions

Read

HTTP method

GET

Object interface URI

/rest/objects/‹objectID›?metadata/tags

Namespace interface URI

```
/rest/namespace/photos/<pathname>?metadata/tags
```

Request parameters

Required:

- "x-emc-date"or "Date"
- "x-emc-signature"
- "x-emc-uid"

Optional

- "x-emc-utf8" (when the data returned is in Unicode)

Object interface examples

Request

```
GET
   /rest/objects/4dc19958a20574f604dc1a3a1ec8cb04dcc1679d6609?metadata
   /tags HTTP/1.1
accept: */*
date: Thu, 12 May 2011 17:42:32 GMT
content-type: application/octet-stream
x-emc-date: Thu, 12 May 2011 17:42:32 GMT
host: 10.238.112.140:1234
x-emc-uid: 66371ac3bd8148348c0f3f1545e2da69/test-uid
x-emc-signature: cNLgr0oBkiIy24+5OgdeLZCjVy0=
```

Response

```
HTTP/1.1 200 OK
Date: Thu, 12 May 2011 17:42:36 GMT
Server: Apache
x-emc-policy: _int
x-emc-tags: city, country, state
x-emc-listable-tags: color, pattern
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

List user metadata tags with Unicode characters

In this example, the tags that will be returned are in Unicode so the request includes the "x-emc-utf8" header set to true so that Atmos will percent-encode the values it returns on the "x-emc-tags" and "x-emc-listable-tags" headers.

Request

```
GET
   /rest/objects/4ef49feaa106904c04ef4a066e778104f071a5ff0c85?metadata
   /tags HTTP/1.1
date: Fri, 06 Jan 2012 16:47:33 GMT
content-type: application/octet-stream
x-emc-date: Fri, 06 Jan 2012 16:47:33 GMT
x-emc-utf8: true
x-emc-uid: 071464e3e8fb4cce8609a623fd9df025/user1
x-emc-signature: lMgIiA1XbLod3knlkUMqbFwZhmM=
```

Response

```
HTTP/1.1 200 OK
Date: Fri, 06 Jan 2012 16:47:33 GMT
Server: Apache
x-emc-policy: _int
x-emc-utf8: true
x-emc-tags: %ce%bc%ce%ad%ce%b3%ce%b5%ce%b8%ce%bf%cf%82,
   %cf%87%cf%81%cf%8e%ce%bc%ce%b1
x-emc-listable-tags: %cf%80%ce%b5%cf%81%ce%b9%ce%bf%cf%87%ce%ae
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Namespace interface examples

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

Returns the list of all versions for the specified objectID. The list includes the version number, the object ID, and the version creation date.

This list has as many entries as there are versions of the specified object. They are sorted based on their create time. This list includes all versioned objects that have been created unless they have been deleted.

The list request must include the top-level object's object ID. You cannot request the list by the top-level object's namespace path.

Use the "x-emc-limit" header to limit the resultset for a request. Atmos limits the maximum number of versions it can return to 4096. If you set the x-emc-limit value to higher, it will only return 4096.

Permissions

Read access to the top-level object.

HTTP method

GET

Object interface URI

`/rest/objects/<objectID>?versions`

Namespace interface URI

Not supported

Request parameters

Required:

- "x-emc-date"or "Date"
- "x-emc-signature"
- "x-emc-uid"

Object interface

Request

```
GET
   /rest/objects/491abe33a105736f0491c2088492430491c5d0d67efc?versions
   HTTP/1.1
accept: */*
date: Thu, 13 Nov 2008 16:59:59 GMT
content-type: application/octet-stream
x-emc-date: Thu, 13 Nov 2008 16:59:59 GMT
host: 168.159.116.51
x-emc-uid: 6039ac182f194e15b9261d73ce044939/user1
x-emc-signature: tKLhz275+l8SMxoVnzoo/TNgbu8=
```

Response

```
HTTP/1.1 200 OK
Date: Thu, 13 Nov 2008 16:59:59 GMT
Server: Apache/2.0.63 (rPath)
Content-Length: 252
Connection: close
Content-Type: text/xml
<?xml version='1.0' encoding='UTF-8'?>
<ListVersionsResponse xmlns='http://www.emc.com/cos/'>
 <Ver>
  <VerNum>0</VerNum>
  <OID>491abe33a105736f0491c2088492430491c5d0fbaf74</OID>
  <itime>2008-11-12T16:00:00Z</itime>
  </Ver>
 <Ver>
  <VerNum>1</VerNum>
  <OID>491abe33a105736f0491c2088492430491c5d0f0daa8</OID>
  <itime>2008-11-13T16:59:59Z</itime>
 </Ver>
</ListVersionsResponse>
```

Reading an object

Use this operation to:

- **Return the contents of an object.** The contents include the associated user metadata, system metadata, and access-control lists.
  - Use the optional "Range" header to read only part of the object. The value of the Range header should be the byte ranges to retrieve, in the form Bytes=begin_offset-end_offset. The byte offsets are 0 based: 0 is the first byte, 1 is the second byte, and so on.

- **List the contents of a directory.** By default, the operation returns a list of directory entries, and each entry includes the object ID, the filename, and file type. The operation also allows you to return metadata for each of the entries in the directory by using the following headers:
  - To get all system and user metadata for each entry in the directory, specify the request header "x-emc-include-meta": true.
  - To get a subset of system metadata tags, specify the tag names as a comma-separated list on the "x-emc-system-tags" header.
  - To get a subset of user metadata tags, specify the tag names on as a comma-separated list on the "x-emc-user-tags".

    You can combine the "x-emc-system-tags" and "x-emc-user-tags".

    See "Namespace interface — Directory listing examples" on page 123 for examples of how to use each of these headers.

  In some cases, Atmos might force the pagination of the resultset if the number of entries is too large. To ensure that your application can handle forced pagination, it should be prepared to handle an "x-emc-token" in the response.

To define pagination, use the "x-emc-limit" header.

If the object was created with a checksum, the "x-emc-wschecksum" header is returned in the response.

If the object's metadata (user or system) includes Unicode data, include the "x-emc-utf8" (true) on the request and Atmos will return them as percent-encoded values.

Permissions

Read permission is required (for both the namespace interface and the object interface).

HTTP method

GET

Object interface URI

`/rest/objects/objectID`

Namespace interface URI

`/rest/namespace/pathname`

Request parameters

Required:

- "x-emc-date"or "Date"
- "x-emc-signature"
- "x-emc-uid"
- "x-emc-utf8" (use if metadata names/values are in Unicode)

Optional:

- "Range"
- "x-emc-include-meta"
- "x-emc-limit"
- "x-emc-system-tags"
- "x-emc-token"
- "x-emc-user-tags"

Object interface examples

- "Basic read object"
- "Read a directory containing one file"
- "Read object using range header"
- "Read object containing Unicode metadata name/value pairs"

Basic read object

Request

```
GET /rest/objects/4dc19958a20574f604dc1a3a1ec8cb04dcc1679d6609
   HTTP/1.1
accept: */*
date: Thu, 12 May 2011 17:53:21 GMT
content-type: application/octet-stream
x-emc-date: Thu, 12 May 2011 17:53:21 GMT
host: 10.238.112.140:1234
x-emc-uid: 66371ac3bd8148348c0f3f1545e2da69/test-uid
x-emc-signature: OGB4xNR6CoIUscfl1b7DdatQMV8=
```

Response

```
HTTP/1.1 200 OK
Date: Thu, 12 May 2011 17:53:25 GMT
Server: Apache
x-emc-policy: default
x-emc-meta: city=boston, state=ma, country=united states,
   atime=2011-05-12T17:18:50Z, mtime=2011-05-12T17:18:50Z,
   ctime=2011-05-12T17:18:50Z, itime=2011-05-12T17:18:49Z,
   type=regular, uid=test-uid, gid=apache,
   objectid=4dc19958a20574f604dc1a3a1ec8cb04dcc1679d6609, size=110076,
   nlink=0, policyname=default
x-emc-listable-meta: color=blue, pattern=stripes
x-emc-useracl: anne=FULL_CONTROL, test-uid=FULL_CONTROL
x-emc-groupacl: other=NONE
Content-Length: 110076
Connection: close
Content-Type: application/octet-stream
```

Read a directory containing one file

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

Request

```
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

```
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>
```

Read object using 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.

Request 1

This example requests the entire object

```
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
```

Request 2

Requests bytes 4-8.

```
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
```

Request 3

Requests bytes 4-8 and 41-44.

```
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--
```

Request 4

Requests from byte 32 until the end of the object.

```
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
```

Request 5

Requests the last 9 bytes.

```
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
```

Request 6

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

```
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--
```

Request 7

Requests a range that is valid but not satisfiable.

```
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>
```

Request 8

Requests one range that is not satisfiable and one range that is satisfiable.

```
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
```

Request 9

Requests an invalid byte range. The entire object is returned.

```
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
```

Read object containing Unicode metadata name/value pairs

This example passes in the "x-emc-utf8" header on the request. The data returned on the x-emc-meta and x-emc-listable headers are percent-encoded because they are in Unicode format.

Request

```
GET /rest/objects/4ef49feaa106904c04ef4a066e778104f071a5ff0c85
   HTTP/1.1
date: Fri, 06 Jan 2012 16:49:24 GMT
content-type: application/octet-stream
x-emc-date: Fri, 06 Jan 2012 16:49:24 GMT
x-emc-utf8: true
x-emc-uid: 071464e3e8fb4cce8609a623fd9df025/user1
x-emc-signature: 1zeajUWbFs2LreETTfmUtAzDSZw=
```

Response

```
HTTP/1.1 200 OK
Date: Fri, 06 Jan 2012 16:49:24 GMT
Server: Apache
x-emc-policy: default
x-emc-utf8: true
x-emc-meta: %cf%87%cf%81%cf%8e%ce%bc%ce%b1=%ce%bc%cf%80%ce%bb%ce%b5,
   %ce%bc%ce%ad%ce%b3%ce%b5%ce%b8%ce%bf%cf%82=%ce%bc%ce%b9%ce%ba%cf%81
   %cf%8c, atime=2012-01-06T16:45:22Z,
mtime=2012-01-06T15:59:28Z, ctime=2012-01-06T16:45:22Z,
   itime=2012-01-06T15:59:27Z, type=regular, uid=user1, gid=apache,
   objectid=4ef49feaa106904c04ef4a066e778104f071a5ff0c85,
objname=%cf%85%cf%80%ce%bf%ce%bb%ce%bf%ce%b3%ce%b9%cf%83%cf%84%ce%ae.j
   pg, size=459, nlink=1, policyname=default
x-emc-listable-meta:
   %cf%80%ce%b5%cf%81%ce%b9%ce%bf%cf%87%ce%ae=%ce%b2%cf%8c%cf%81%ce%b5
   %ce%b9%ce%b1
x-emc-useracl: user1=FULL_CONTROL
x-emc-groupacl: other=NONE
Content-Length: 459
Connection: close
Content-Type: application/octet-stream
```

Namespace interface examples

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: LYcvpkX1jpjdguTf2VpO5Dkt4TM=
```

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/D7jyjyifx2U/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>
```

Read a directory

This example shows how to use the read object operation for a directory. It uses:

- "x-emc-limit" to request that up to two entries be returned. When listing a directory using ReadObject or when using ListObjects, the "x-emc-token" header may be returned in the response headers at any time.
- If the "x-emc-token"header exists, it means that a partial list of results was returned, and that you must use pagination to retrieve the full list of results as shown in Request 4.

Request 3

```
GET /rest/namespace/testdirectory/ HTTP/1.1
   accept: */*
   x-emc-limit: 2
   date: Mon, 15 Mar 2010 19:27:48 GMT
   content-type: application/octet-stream
   x-emc-date: Mon, 15 Mar 2010 19:27:48 GMT
   host: 168.159.116.116:8080
   x-emc-uid: 1fd94b5d1a30483b818e4926c6edbb81/test1
   x-emc-signature: ydK9cONyE4JSfBxl/HMaXIrrBkk=
```

Response 3

```
HTTP/1.1 200 OK
   Date: Mon, 15 Mar 2010 19:27:48 GMT
   Server: Apache
   x-emc-groupacl: other=NONE
   x-emc-useracl: test1=FULL_CONTROL
   x-emc-policy: _int
   x-emc-meta: atime=2010-03-15T17:23:56Z, mtime=2010-03-15T17:24:36Z,
   ctime=2010-03-15T17:24:36Z, itime=2010-03-15T17:23:56Z,
   type=directory, uid=test1, gid=apache,
   objectid=4b97cdfca2068f2c04b97ce826fb9504b9e6d2c4c859,
   objname=testdirectory, size=4096, nlink=1, policyname=default
   x-emc-token: file3
   Content-Length: 489
   Connection: close
   Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
   <ListDirectoryResponse xmlns='http://www.emc.com/cos/'>
     <DirectoryList>
      <DirectoryEntry>
        <ObjectID>4b97cdfca2068f2c04b97ce826fb9504b9e6d40a
          1270</ObjectID>
        <FileType>regular</FileType>
        <Filename>file1</Filename>
      </DirectoryEntry>
      <DirectoryEntry>
        <ObjectID>4b97cdfca2068f2c04b97ce826fb9504b9e6d41d
          0308</ObjectID>
        <FileType>regular</FileType>
        <Filename>file2</Filename>
      </DirectoryEntry>
     </DirectoryList>
   </ListDirectoryResponse>
```

Request 4

To get the next set of results (next page) invoke the operation again, providing the value of "x-emc-token" of the response in the subsequent request. This example uses the token that was returned from the previous call:

```
GET /rest/namespace/testdirectory/ HTTP/1.1
   x-emc-token: file3
   accept: */*
   x-emc-limit: 2
   date: Mon, 15 Mar 2010 19:35:45 GMT
   content-type: application/octet-stream
   x-emc-date: Mon, 15 Mar 2010 19:35:45 GMT
   host: 168.159.116.116:8080
   x-emc-uid: 1fd94b5d1a30483b818e4926c6edbb81/test1
   x-emc-signature: Ng5fqKtkzl5Ho0o4t2PUeq+CCYM=
```

Response 4

```
HTTP/1.1 200 OK
   Date: Mon, 15 Mar 2010 19:35:49 GMT
   Server: Apache
   x-emc-groupacl: other=NONE
   x-emc-useracl: test1=FULL_CONTROL
   x-emc-policy: _int
   x-emc-meta: atime=2010-03-15T17:23:56Z, mtime=2010-03-15T17:24:36Z,
   ctime=2010-03-15T17:24:36Z, itime=2010-03-15T17:23:56Z,
   type=directory, uid=test1, gid=apache,
   objectid=4b97cdfca2068f2c04b97ce826fb9504b9e6d2c4c859,
   objname=testdirectory, size=4096, nlink=1, policyname=default
   Content-Length: 489
   Connection: close
   Content-Type: text/xml
<?xml version='1.0' encoding='UTF-8'?>
   <ListDirectoryResponse xmlns='http://www.emc.com/cos/'>
   <DirectoryList>
   <DirectoryEntry>

   <ObjectID>4b97cdfca2068f2c04b97ce826fb9504b9e6d436bd68</ObjectID>
      <FileType>regular</FileType>
      <Filename>file3</Filename>
   </DirectoryEntry>
   <DirectoryEntry>

   <ObjectID>4b97cdfca2068f2c04b97ce826fb9504b9e6d48c1d26</ObjectID>
      <FileType>regular</FileType>
      <Filename>file4</Filename>
   </DirectoryEntry>
   </DirectoryList>
   </ListDirectoryResponse>
```

Read object with checksum

Request

```
GET /rest/namespace/file1.txt HTTP/1.1
accept: */*
date: Fri, 11 Jun 2010 11:14:44 GMT
x-emc-date: Fri, 11 Jun 2010 11:14:44 GMT
host: 168.159.116.112:2345
x-emc-uid: ebd858f829114dfabbcf069637a07cfe/user1
x-emc-signature: QxCk89s7TvWsoPptteVEAXPO8KM=
```

Response

```
HTTP/1.1 200 OK
Date: Thu, 17 Jun 2010 12:40:53 GMT
Server: Apache
x-emc-policy: default
x-emc-meta: atime=2010-06-11T11:16:44Z, mtime=2010-06-11T11:16:44Z,
   ctime=2010-06-11T11:16:44Z, itime=2010-06-11T11:16:44Z,
   type=regular, uid=user1, gid=apache,
   objectid=4bf520e2a105737304bf52170a4e6204c121b1ca464d, size=1037,
   nlink=0
x-emc-useracl: user1=FULL_CONTROL
x-emc-groupacl: other=NONE
x-emc-wschecksum: sha0/1037/87hn7kkdd9d982f031qwe9ab224abjd6h1276nj9
Content-Length: 1037
Connection: close
Content-Type: application/octet-stream
```

Namespace interface — Directory listing examples

The examples in this section show how to use various headers to request that different metadata be returned for the contents of a directory.

Request1

This example shows the default read directory operation. By default, the operation returns the Object ID, file type, and file name for each entry (or file) in the directory. In this example, the dir3 directory includes a single file called file1.

```
GET /rest/namespace/dir3 HTTP/1.1
accept: */*
date: Tue, 01 Feb 2011 09:44:09 GMT
content-type: application/octet-stream
x-emc-date: Tue, 01 Feb 2011 09:44:09 GMT
host: 10.4.136.25:1234
x-emc-uid: 470302c7294145f2b0ca5cabc4f3e0fe/testUID
x-emc-signature: uQNlndtyTCjroTdO+qy+mhSEuLE=
```

Response1

```
HTTP/1.1 200 OK
Date: Tue, 01 Feb 2011 09:44:13 GMT
Server: Apache
x-emc-policy: _int
x-emc-meta: atime=2011-02-01T09:38:51Z, mtime=2011-02-01T09:38:52Z,
   ctime=2011-02-01T09:38:52Z, itime=2011-02-01T09:38:52Z,
   type=directory, uid=testUID, gid=apache,
   objectid=4d3e8694a10574f604d3e8eea8f08404d47d4ac54bef,
objname=dir3, size=134, nlink=2, policyname=default
x-emc-useracl: testUID=FULL_CONTROL
x-emc-groupacl: other=NONE
Content-Length: 322
Connection: close
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<ListDirectoryResponse xmlns='http://www.emc.com/cos/'>
  <DirectoryList>
    <DirectoryEntry>
         <ObjectID>4d3e8694a10574f604d3e8eea8f08404d47d4ac
           8d808</ObjectID>
      <FileType>regular</FileType>
      <Filename>file1</Filename>
    </DirectoryEntry>
  </DirectoryList>
</ListDirectoryResponse>
```

Request2

This request uses the "x-emc-include-meta" header so that all of the system and user metadata for each directory entry is included in their response.

```
GET /rest/namespace/dir3 HTTP/1.1
accept: */*
date: Tue, 01 Feb 2011 10:50:35 GMT
content-type: application/octet-stream
x-emc-date: Tue, 01 Feb 2011 10:50:35 GMT
host: 10.4.136.25:1234
x-emc-uid: 470302c7294145f2b0ca5cabc4f3e0fe/testUID
x-emc-signature: nHskNUaVzmLpXaLuUvFcHTjce/0=
x-emc-include-meta: true
```

Response2

```
HTTP/1.1 200 OK
Date: Tue, 01 Feb 2011 10:50:39 GMT
Server: Apache
x-emc-policy: _int
x-emc-meta: atime=2011-02-01T09:38:51Z, mtime=2011-02-01T09:38:52Z,
   ctime=2011-02-01T09:38:52Z, itime=2011-02-01T09:38:52Z,
   type=directory, uid=testUID, gid=apache,
   objectid=4d3e8694a10574f604d3e8eea8f08404d47d4ac54bef,
   objname=dir3, size=134, nlink=2, policyname=default
x-emc-useracl: testUID=FULL_CONTROL
x-emc-groupacl: other=NONE
Content-Length: 1758
Connection: close
Content-Type: text/xml
<?xml version='1.0' encoding='UTF-8'?>
<ListDirectoryResponse xmlns='http://www.emc.com/cos/'>
 <DirectoryList>
   <DirectoryEntry>
    <ObjectID>4d3e8694a10574f604d3e8eea8f08404d47d4ac8d
        808</ObjectID>
    <FileType>regular</FileType>
    <Filename>file1</Filename>
    <SystemMetadataList>
     <Metadata>
       <Name>atime</Name>
       <Value>2011-02-01T09:38:53Z</Value>
     </Metadata>
     <Metadata>
      <Name>mtime</Name>
      <Value>2011-02-01T09:38:51Z</Value>
     </Metadata>
     <Metadata>
      <Name>ctime</Name>
      <Value>2011-02-01T09:38:51Z</Value>
     </Metadata>
     <Metadata>
      <Name>itime</Name>
      <Value>2011-02-01T09:38:52Z</Value>
     </Metadata>
     <Metadata>
      <Name>type</Name>
      <Value>regular</Value>
     </Metadata>
     <Metadata>
      <Name>uid</Name>
      <Value>testUID</Value>
     </Metadata>
     <Metadata>
      <Name>gid</Name>
      <Value>apache</Value>
     </Metadata>
     <Metadata>
      <Name>objectid</Name>
      <Value>4d3e8694a10574f604d3e8eea8f08404d47d4ac8d8
           08</Value>
     </Metadata>
     <Metadata>
      <Name>objname</Name>
      <Value>file1</Value>
     </Metadata>
     <Metadata>
      <Name>size</Name>
      <Value>0</Value>
     </Metadata>
```

```
     <Metadata>
      <Name>nlink</Name>
      <Value>1</Value>
     </Metadata>
     <Metadata>
      <Name>policyname</Name>
      <Value>default</Value>
     </Metadata>
    </SystemMetadataList>
    <UserMetadataList>
     <Metadata>
      <Name>city-boston</Name>
      <Value></Value>
      <Listable>false</Listable>
     </Metadata>
     <Metadata>
      <Name>state</Name>
      <Value>ma</Value>
      <Listable>false</Listable>
     </Metadata>
     <Metadata>
      <Name>color</Name>
      <Value>blue</Value>
      <Listable>true</Listable>
     </Metadata>
     </UserMetadataList>
   </DirectoryEntry>
 </DirectoryList>
</ListDirectoryResponse>
```

Get specific metadata tags

Request3

This request uses the "x-emc-user-tags" to request that only the state and color user metadata tags get returned, and it uses the "x-emc-system-tags" header to limit the system metadata to atime and size.

```
content-type: application/octet-stream
x-emc-date: Tue, 01 Feb 2011 09:42:51 GMT
host: 10.4.136.25:1234
x-emc-user-tags: state,color
x-emc-system-tags: atime,size
x-emc-uid: 470302c7294145f2b0ca5cabc4f3e0fe/testUID
x-emc-signature: wyreXy+3U3xW9SLKVl5NW6oRcVA=
```

Response3

```
HTTP/1.1 200 OK
Date: Tue, 01 Feb 2011 09:42:56 GMT
Server: Apache
x-emc-policy: _int
x-emc-meta: atime=2011-02-01T09:38:51Z, mtime=2011-02-01T09:38:52Z,
   ctime=2011-02-01T09:38:52Z, itime=2011-02-

01T09:38:52Z, type=directory, uid=testUID, gid=apache,
   objectid=4d3e8694a10574f604d3e8eea8f08404d47d4ac54bef,

objname=dir3, size=134, nlink=2, policyname=default
x-emc-useracl: testUID=FULL_CONTROL
x-emc-groupacl: other=NONE
Content-Length: 787
Connection: close
Content-Type: text/xml
```

```
<?xml version='1.0' encoding='UTF-8'?>
<ListDirectoryResponse xmlns='http://www.emc.com/cos/'>
   <DirectoryList>
   <DirectoryEntry>

   <ObjectID>4d3e8694a10574f604d3e8eea8f08404d47d4ac8d808</ObjectID>
      <FileType>regular</FileType>
      <Filename>file1</Filename>
      <SystemMetadataList>
         <Metadata>
            <Name>atime</Name>
            <Value>2011-02-01T09:38:53Z</Value>
         </Metadata>
         <Metadata>
            <Name>size</Name>
            <Value>0</Value>
         </Metadata>
      </SystemMetadataList>
      <UserMetadataList>
         <Metadata>
            <Name>state</Name>
            <Value>ma</Value>
            <Listable>false</Listable>
         </Metadata>
         <Metadata>
            <Name>color</Name>
            <Value>blue</Value>
            <Listable>true</Listable>
         </Metadata>
      </UserMetadataList>
   </DirectoryEntry>
   </DirectoryList>
</ListDirectoryResponse>
```

Renaming a file or directory in the namespace

Renames a file or a directory within its current namespace. Requires the "x-emc-path" custom header to provide the full path to the new file or directory name. If the new name passed in on the "x-emc-path" header is in Unicode format, the client application must percent-encode the data, and include the "x-emc-utf8" header on the request.

Use the optional "x-emc-force" header to specify whether the operation should overwrite the target file or directory if it already exists. To overwrite the target file or directory (if it already exists), set "x-emc-force" to true. If "x-emc-force" is not specified or set to false, the target file will not be overwritten and the rename operation will fail. A directory must be empty to be overwritten.

This operation is not supported in the object interface. It returns an error code 1042 if attempted.

Permissions

Write (execute) permissions on both the parent and target directories.

HTTP method

POST

Object interface URI

Not supported

Namespace interface URI

`/namespace/pathname?rename`

Request parameters

Required:

- "x-emc-date" or "Date"
- "x-emc-path"
- "x-emc-signature"
- "x-emc-uid"
- "x-emc-utf8" (only required when the x-emc-path data is in Unicode format)

Optional:

- "x-emc-force"

Namespace interface examples

Rename a file

The following example shows how to rename a file called custnames (located in the /dir directory of the namespace) to custinfo.

Request

```
POST /rest/namespace/dir/custnames?rename HTTP/1.1
   date: Wed, 06 Jan 2010 16:12:09 GMT
   x-emc-date: Wed, 06 Jan 2010 16:12:09 GMT
   x-emc-path: dir/custinfo
   x-emc-force: true
   x-emc-uid: 47cadb22de2e46328e49bafc02f64637/user1
   x-emc-signature: snxbvMmc4vyCm/b+XsDje30coSs=
```

Response

```
HTTP/1.1 200 OK
   Date: Wed, 06 Jan 2010 16:12:09 GMT
   Server: Apache
   x-emc-policy: _int
   Content-Length: 0
   Connection: close
   Content-Type: text/plain; charset=UTF-8
```

Rename a file when x-emc-force is false

This operation requests a rename from myDir/myfile.txt to myNewDir/newName.txt, but the x-emc-force header is set to false and the operation fails.

Request

```
POST /rest/namespace/myDir/myfile.txt?rename HTTP/1.1
accept: */*
```

```
date: Thu, 29 Jul 2010 19:17:01 GMT
content-type: application/octet-stream
x-emc-date: Thu, 29 Jul 2010 19:17:00 GMT
x-emc-path: myNewDir/newName.txt
host: 168.159.116.112:1234
x-emc-uid: 624a29f7a544467dabaf791f6daf6939/user1
x-emc-signature: go08bLAGmT3dP8t1U/tG9fEFWO0=
x-emc-force: false
```

Response

```
HTTP/1.1 400 Bad Request
Date: Thu, 29 Jul 2010 19:17:01 GMT
Server: Apache
Content-Length: 149
Connection: close
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<Error>
 <Code>1016</Code>
 <Message>The resource you are trying to create already
      exists.</Message>
</Error>
```

Rename a directory

The following example shows how to rename a directory called samples to examples.

Request

```
POST /rest/namespace/samples?rename HTTP/1.1
   date: Wed, 06 Jan 2010 16:17:51 GMT
   x-emc-date: Wed, 06 Jan 2010 16:17:51 GMT
   x-emc-path: examples
   x-emc-force: true
   x-emc-uid: 47cadb22de2e46328e49bafc02f64637/user1
   x-emc-signature: zZ0HcFSpiW1bKbWS9QF9eofViGU=
```

Response

```
HTTP/1.1 200 OK
   Date: Wed, 06 Jan 2010 16:17:51 GMT
   Server: Apache
   x-emc-policy: _int
   Content-Length: 0
   Connection: close
   Content-Type: text/plain; charset=UTF-8
```

Move a file to a different directory

This example shows how to move the file custinfo from the directory dir/ to the directory archive/.

Request

```
POST /rest/namespace/dir/custinfo?rename HTTP/1.1
date: Wed, 06 Jan 2010 16:20:52 GMT
x-emc-date: Wed, 06 Jan 2010 16:20:52 GMT
x-emc-path: archive/custinfo
x-emc-force: true
x-emc-uid: 47cadb22de2e46328e49bafc02f64637/user1
x-emc-signature: 4YAhxg9fIiIajX1J4eDiFrWdNnE=
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 06 Jan 2010 16:20:52 GMT
Server: Apache
x-emc-policy: _int
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Rename an object to a Unicode value

This examples shows how to rename an object from images/computer.jpg to images/υπολογιστή.jpg.

Note: Do not encode the path separator '/'.

```
POST /rest/namespace/images/computer.jpg?rename HTTP/1.1
date: Fri, 06 Jan 2012 16:16:00 GMT
content-type: application/octet-stream
x-emc-date: Fri, 06 Jan 2012 16:16:00 GMT
x-emc-path:
   images/%CF%85%CF%80%CE%BF%CE%BB%CE%BF%CE%B3%CE%B9%CF%83%CF%84%CE%AE
   .jpg
x-emc-utf8: true
x-emc-uid: 071464e3e8fb4cce8609a623fd9df025/user1
x-emc-signature: NLGgfTvQSrckJOx64ACJy9Nxhz8=
StringToSign

POST
application/octet-stream

Fri, 06 Jan 2012 16:16:00 GMT
/rest/namespace/images/computer.jpg?rename
x-emc-date:Fri, 06 Jan 2012 16:16:00 GMT
x-emc-path:images/%CF%85%CF%80%CE%BF%CE%BB%CE%BF%CE%B3%CE%B9%CF%83%CF%
   84%CE%AE.jpg
x-emc-uid:071464e3e8fb4cce8609a623fd9df025/user1
x-emc-utf8:true
```

Restoring a version

Restores a version of an object to the top-level object. This operation is synchronous so any applications that request a restore operations might experience some delay depending on the size of the object being restored. The request must meet these requirements:

- You cannot use a versioned object to restore another versioned object.
- You cannot restore from a deleted versioned object.

Permissions

Write permission on the top-level object, and read permission on the version object. This is a UID-based permission, not an administrative role.

HTTP method

PUT

Object interface URI

/rest/objects/<*objectID*>?versions

Object interface examples

Request

```
PUT
   /rest/objects/491abe33a105736f0491c2088492430491c5d0d67efc?versions
   HTTP/1.1
x-emc-version-oid: 491abe33a105736f0491c2088492430491c5d0f0daa8
accept: */*
date: Thu, 13 Nov 2008 16:59:58 GMT
content-type: application/octet-stream
x-emc-date: Thu, 13 Nov 2008 16:59:58 GMT
host: 168.159.116.51
x-emc-uid: 6039ac182f194e15b9261d73ce044939/user1
x-emc-signature: 4oZorU2nBQlADhq7fTMklstL1eU=
```

Response

```
HTTP/1.1 200 OK
Date: Thu, 13 Nov 2008 16:59:58 GMT
Server: Apache/2.0.63 (rPath)
x-emc-delta: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Setting an ACL

Sets the access control for this object. The operation can be used for setting or resetting permissions. The request must include "x-emc-groupacl" or "x-emc-useracl", or the operation returns an error.

Permissions

Must be the owner of the object or file.

HTTP method

POST

Object interface URI

`/rest/objects/`*`objectID`*`?acl`

Namespace interface URI

`/namespace/`*`pathname`*`?acl`

Request parameters

Required:

- "x-emc-date" or "Date"

- "x-emc-groupacl" or "x-emc-useracl"
- "x-emc-signature"
- "x-emc-uid"

Object interface examples

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 examples

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

Writes user metadata into the object. To set:

- One or more listable user metadata tags, pass the tag name and tag value (name=value) on the "x-emc-listable-meta" header on the request.
- One or more non-listable user metadata tags, pass the tag name and tag value (name=value) on the "x-emc-meta" header on the request.

When specifying more than one tag, supply the name=value pairs in a comma-separated list. The request fails if it does not include either "x-emc-listable-meta" or "x-emc-meta".

When writing data in Unicode format, the data must be percent-encoded, and the request must include the "x-emc-utf8" header or the request fails.

Permissions

Write permission on the object.

HTTP method

POST

Object interface URI

/rest/objects/‹objectID›?metadata/user

Namespace interface URI

```
POST /rest/namespace/<pathname>?metadata/user
```

Request parameters

Required:

- "x-emc-date"or "Date"
- "x-emc-meta" or "x-emc-listable-meta"
- "x-emc-signature"
- "x-emc-uid"
- "x-emc-utf8" (required if the data passed on the x-emc-meta or x-emc-listable-meta includes Unicode data)

Object interface examples

Set listable and non-listable metadata tags

This example sets the listable user metadata tag named color to the value of blue, and it sets the non-listable user metadata tags city and state to the values of boston and MA respectively.

Request

```
POST /rest/objects HTTP/1.1
x-emc-listable-meta: color=blue
x-emc-meta: city=boston,state=MA
accept: */*
x-emc-useracl: anne=FULL_CONTROL
date: Wed, 11 May 2011 20:34:26 GMT
content-type: application/octet-stream
x-emc-date: Wed, 11 May 2011 20:34:26 GMT
host: 10.238.112.140:1234
content-length: 110076
x-emc-uid: 66371ac3bd8148348c0f3f1545e2da69/test-uid
x-emc-signature: z8VDtRLDNZ6cjq7VYUsZXtOuiQs=
```

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
```

Set listable and non-listable metadata tags for Unicode values

Sets non-listable metadata to χρώμα=μπλε,μέγεθος=μικρό (color=blue, size=small), and sets listable metadata to περιοχή=βόρεια (region=north). Because the metadata values are in Unicode, the values must be percent-encoded and the request must include the x-emc-utf8:true header.

Request

```
POST
   /rest/objects/4ef49feaa106904c04ef4a066e778104f071a5ff0c85?metadata
   /user HTTP/1.1
x-emc-listable-meta:
   %CF%80%CE%B5%CF%81%CE%B9%CE%BF%CF%87%CE%AE=%CE%B2%CF%8C%CF%81%CE%B5
   %CE%B9%CE%B1
x-emc-meta:
   %CF%87%CF%81%CF%8E%CE%BC%CE%B1=%CE%BC%CF%80%CE%BB%CE%B5,%CE%BC%CE%A
   D%CE%B3%CE%B5%CE%B8%CE%BF%CF%82=%CE%BC%CE%B9%CE%BA%CF%81%CF%8C
accept: */*
date: Fri, 06 Jan 2012 16:41:23 GMT
content-type: application/octet-stream
x-emc-date: Fri, 06 Jan 2012 16:41:23 GMT
x-emc-utf8: true
host: 127.0.0.1
x-emc-uid: 071464e3e8fb4cce8609a623fd9df025/user1
x-emc-signature: x2IhX/4lVPXf4cDTazu2ZDS+In8=
StringToSign
```

Request

```
POST
application/octet-stream

Fri, 06 Jan 2012 16:41:23 GMT
/rest/objects/4ef49feaa106904c04ef4a066e778104f071a5ff0c85?metadata/us
   er
x-emc-date:Fri, 06 Jan 2012 16:41:23 GMT
x-emc-listable-meta:%CF%80%CE%B5%CF%81%CE%B9%CE%BF%CF%87%CE%AE=%CE%B2%
   CF%8C%CF%81%CE%B5%CE%B9%CE%B1
```

```
x-emc-meta:%CF%87%CF%81%CF%8E%CE%BC%CE%B1=%CE%BC%CF%80%CE%BB%CE%B5,%CE
   %BC%CE%AD%CE%B3%CE%B5%CE%B8%CE%BF%CF%82=%CE%BC%CE%B9%CE%BA%CF%81%CF
   %8C
x-emc-uid:071464e3e8fb4cce8609a623fd9df025/user1
x-emc-utf8:true
```

Namespace interface examples

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

Updates the contents of an object, including its metadata and ACLs. You can update part of the object or the complete object.

You can also use this operation to add or modify existing metadata or ACLs; for example, you can change metadata from listable to non-listable and vice versa.

To update part of the object, use the "Range" header to specify the beginning and ending offsets.

You can change the size of an object in these ways:

- 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
  ```

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

If the metadata tags that you pass in this request are Unicode, you must percent-encode the data before submitting the request, and include the "x-emc-utf8" header on the request. Atmos will percent-encode the values that it returns.

For applications that must conform to SEC 17a-4f standards, you must specify the "x-emc-wschecksum" header. When you use this header, you must send the checksum of the entire object that is part of the request.

Updating checksummed objects

To update an object that was created with a checksum, the update request must:

- Be an append operation.
- Include the "x-emc-wschecksum" header. The algorithm name included in the header must match the value stored in the object's metadata.

When you make the append request, you must pass in the checksum of the complete object (the current object size + the amount appended). This ensures that any data inconsistency is detected as soon as it happens. Suppose you have a 10k object, and you append 10k to it four times. The data flow would be:

- Create 10k object (POST request that includes checksum of the 10k object.)
- Append 10k to the existing 10k object (PUT request that includes the checksum of the now 20k object).
- Append 10k to the existing 20k object. (PUT request with the checksum of the now 30k object).
- Append 10k to the existing 30k object. (PUT request with the checksum of the now 40k object).
- Append 10k to the existing 40k object. (PUT request with the checksum of the now 50k object.)

You cannot:

- Pass in a checksum if the object was not created with a checksum.
- Convert an object that has a checksum to one that does not (or vice versa). To remove or add a checksum to an object, you must delete the object and recreate it.

Permissions

Write permission on the object.

HTTP method

PUT

Object interface URI

`/rest/objects/<objectID>`

Namespace interface URI

`/rest/namespace/<pathname>`

Request parameters

Required:

- "x-emc-date"or "Date"
- "x-emc-signature"
- "x-emc-uid"
- "x-emc-utf8" (only required if metadata values are in Unicode)
- "x-emc-wschecksum" (only required if the application must conform to SEC 17a-4f standards)

Optional:

- "Range"
- "x-emc-wschecksum"
- "x-emc-groupacl"
- "x-emc-listable-meta"
- "x-emc-meta"
- "x-emc-useracl"

Object interface examples

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
```

Checksum append example

Request

```
PUT /rest/objects/4bf520e2a105737304bf52170a4e6204c337e3f24ba0
   HTTP/1.1
accept: */*
date: Tue, 06 Jul 2010 19:41:30 GMT
content-type: application/octet-stream
x-emc-date: Tue, 06 Jul 2010 19:41:30 GMT
range: Bytes=1037-1086
content-length: 50
x-emc-uid: ebd858f829114dfabbcf069637a07cfe/user1
x-emc-signature: /hNuFdtlDO9Z0Ix6T2+ZxJVk/3E=
x-emc-wschecksum: sha0/1087/4a5411a2c94ef84d32e9ff955a04d8f9f10c6ae9
```

Response

```
HTTP/1.1 200 OK
Date: Thu, 17 Jun 2010 13:22:13 GMT
Server: Apache
x-emc-policy: default
x-emc-wschecksum: sha0/1087/4a5411a2c94ef84d32e9ff955a04d8f9f10c6ae9
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

Namespace interface examples

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: Z5Sl6Pyeu0ehqcyXx7TZgffle8o=
```

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
```

CHAPTER 6
Security

This chapter describes the Atmos web services security model.

Overview

Security for web services consists of:

- *Authentication* using an encrypted signature model. See "Managing authentication" on page 140.
- *Authorization* through access-control lists (ACLs) at the user (UID) level. See "Access Control Lists" on page 143.

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 "Providing anonymous access" on page 33.

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 Atmos 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: securing REST messages with signatures" on page 141.

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.

Timestamps

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: 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 Atmos administrator. Make sure the shared secret is base64-decoded before using it as an input to the HMACSHA1 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: `xxxxxxxxxx\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' +
   CanonicalizedEMCHeaders
```

where + is the concatenation operator.

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

Table 21 HashString Components

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`

CanonicalizedEMCHeaders	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=
```

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: ACLs cannot be used to grant access to UIDs across different subtenants.

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

REST ACLs

You set user-level authorization with the x-emc-groupacl or x-emc-useracl headers, which define access control for objects (see “Atmos custom headers” on page 52). 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/5ca1ab1e0a05737604847ff1f7a26d04848167b63d9f?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
```

CHAPTER 7
Reserved Namespace for Extended Attributes

This chapter describes the Atmos reserved namespace for extended attributes.

Overview

For each file/object, there is a protected namespace — user.maui.* — 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.maui extended-attribute namespace is reserved; for example, EMC controls the contents of the namespace and the format of its fields. Some of the xattrs are exposed to applications (see the table in "Capability" on page 147.") 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, you can query this with:

```
man 5 attr
```

Atmos extended attributes

The protected namespace contains the following attributes:

Table 22 Attributes in the Protected Namespace (page 1 of 2)

Attribute	Can Query?	Can Set?	See...
`capability`	X		"Capability" on page 147
`expirationEnable`	X	X	"Expiration of objects" on page 147
`expirationEnd`	X	X	"Expiration of objects" on page 147
`lso`	X		"Layout storage object" on page 148
`mdsmaster`	X		"MDS (Metadata Service)" on page 149

Table 22 Attributes in the Protected Namespace (page 2 of 2)

Attribute	Can Query?	Can Set?	See...
mdsreplicas	X		"MDS (Metadata Service)" on page 149
nlink	X		"Number of links" on page 149
objectid	X		"Object ID" on page 150
objState	X		objState is an internal field and not relevant for users.
queues	X		"Queues" on page 150
refCount	X		"Reference count" on page 150
retentionEnable	X	X	"Retention of objects" on page 150
retentionEnd	X	X	"Retention of objects" on page 150
stats	X	X	"Statistics" on page 151
tracer	X	X	"Log tracing" on page 151
updateNum	X		"updateNum" on page 152

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.maui.capability /mnt/mauifs/bar
getfattr: Removing leading '/' from absolute path names
# file: mnt/mauifs/bar
user.maui.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.maui.expirationEnable: Of type string ("true" or "false")
- user.maui.expirationEnd:Of type xsd:dateTime (for example, 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: Expiration applies to files, not directories.

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

Get and set examples

```
# 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="2009-04-04T23:22:14Z"

# 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="true"

# setfattr -n user.maui.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 (for example, 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 examples

```
# 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="49a660fb00000000000000000000049a7d 8ea59701"
```

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.maui.refCount /mnt/mauifs/bar
getfattr: Removing leading '/' from absolute path names

# file: mnt/mauifs/bar user.maui.refCount="0"
```

Retention of objects

A retention period is a period during which the data cannot be modified. Object retention is controlled by policies, and the default retention period, unless specified differently in the policy specification, is 0 seconds.

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.maui.retentionEnable— Of type string ("true" or "false")
- user.maui.retentionEnd— Of type xsd:dateTime (for example, 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).

You can use user.maui.retentionEnd to lengthen the retention period, but you cannot use it shorten it or set it to a time in the past.

Objects created by a compliant subtentant cannot turn off retention by setting user.maui.retentionEnable to false.

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

Get and set examples

```
# 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="true"

# 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="2009-03-05T23:22:14Z"

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

# 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="2009-03-06T23:22:14Z"

# setfattr -n user.maui.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.

CHAPTER 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>
```

Error codes

lists the HTTP status codes and descriptions for REST operations:.

Table 23 HTTP status codes for REST (page 1 of 4)

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. **Note:** This error code may indicate that the system is temporarily too busy to process the request. This is a non-fatal error; you can re-try the request later.	409	Conflict
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

Table 23 HTTP status codes for REST (page 2 of 4)

Error Code	Error Message	HTTP Status Code	HTTP Status Description
1011	The specified object length does not match the actual length of the attached object.	400	Bad Request
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 I/O 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
1034	Unable to read the contents of the HTTP body.	400	Bad Request
1037	The specified token is invalid.	400	Bad Request

Table 23 HTTP status codes for REST (page 3 of 4)

Error Code	Error Message	HTTP Status Code	HTTP Status Description
1040	The server is busy. Please try again	500	Internal Server Error
1041	The requested filename length exceeds the maximum length allowed.	400	Bad Request
1042	The requested operation is not supported.	400	Bad Request
1043	The object has the maximum number of links	400	Bad Request
1044	The specified parent does not exist.	400	Bad Request
1045	The specified parent is not a directory.	400	Bad Request
1046	The specified object is not in the namespace.	400	Bad Request
1047	Source and target are the same file.	400	Bad Request
1048	The target directory is not empty and may not be overwritten	400	Bad Request
1049	The checksum sent with the request did not match the checksum as computed by the server	400	Bad Request
1050	The requested checksum algorithm is different than the one previously used for this object.	400	Bad Request
1051	Checksum verification may only be used with append update requests	400	Bad Request
1052	The specified checksum algorithm is not implemented.	400	Bad Request
1053	Checksum cannot be computed for an object on update for which one wasn't computed at create time.	400	Bad Request
1054	The checksum input parameter was missing from the request.	400	Bad Request
1056	The requested operation is not supported for symlinks.	400	Bad Request
1057	If-Match precondition failed.	412	Precondition failed
1058	If-None-Match precondition failed.	412	Precondition failed
1059	The key you are trying to create already exists.	400	Bad Request
1060	The requested key was not found.	404	Not found
1061	The requested pool already exists.	400	Bad Request
1062	The requested pool was not found.	404	Not found
1063	The maximum number of pools has been reached.	400	Bad request
1064	The request could not be completed because the subtenant is over quota	403	Forbidden

Table 23 HTTP status codes for REST (page 4 of 4)

Error Code	Error Message	HTTP Status Code	HTTP Status Description
1065	The request could not be completed because the UID is over quota	403	Forbidden
1070	Did not receive the expected amount of data.	400	Bad request
1071	Client closed connection before reading all data.	499	Client Closed Request
1072	Could not write all bytes to the client.	499	Client Closed Request
1073	Timeout writing data to the client.	499	Client Closed Request

HTTP Success Codes

lists the HTTP status codes and descriptions for REST operations:

Table 24 HTTP success codes

HTTP Status Code	HTTP Status Description	Description
200	OK	The request has succeeded.
201	Created	The request has been fulfilled. For createobject and version object operations, it means that new object was successfully created.
204	No Content	The request has been fulfilled, and no content is being sent with the response. This applies to DeleteObject and DeleteUserMetadata requests.
206	Partial Content	The server has fulfilled the partial GET request for the object. This applies to ReadObject requests that include the Range header).

INDEX