Edgio

Origin Storage
API Reference Guide

2021-04

Table of Contents

Overview

This guide contains technical reference information for developers interested in or intending to integrate their workflow or application with the Limelight *Origin Storage* platform using the HTTP interface and the JSON-RPC Application Programming Interface (API).

Both JSON-RPC 1.0 and 2.0 are supported by the JSON-RPC interface. This document contains information required for the use of JSON-RPC but if you are not familiar with JSON-RPC, it is also recommended you refer to the detailed JSON-RPC specification reference information provided at the following URLs:

- JSON-RPC 1.0: http://json-rpc.org/wiki/specification
- JSON-RPC 2.0: http://www.jsonrpc.org/specification

Before You Begin

To use any of the *Origin Storage* APIs, you need an active *Origin Storage* account, a user ID, and a password. Your Limelight Account name is in the Welcome letter you received when you purchased *Origin Storage*. Please contact your Edgio representative if you have any questions.

Your Limelight Account name is part of the URL to which you will direct API calls. In this documentation, the Limelight Account name and URL are represented as follows:

- `http://{Account name}.upload.llnw.net/jsonrpc2` for calls in the JSON-RPC interface
  - Example: `http://supercustomer.upload.llnw.net/jsonrpc2`
- `http://{Account name}.upload.llnw.net` for calls in the HTTP interface
  - Example: `http://supercustomer.upload.llnw.net`

Note:
If you are a long-standing customer and the preceding endpoints fail to resolve, you should use the following endpoints with the `-l` suffix after `{Account name}`:

- `http://{Account name}-l.upload.llnw.net/jsonrpc2` for calls in the JSON-RPC interface
- `http://{Account name}-l.upload.llnw.net` for calls in the HTTP interface

Also note that all your content is stored in the root directory in a sub-directory named for your Limelight Account name:

`/{Account name}/EMEA/games`

Example: `/supercustomer/EMEA/games`

Available Interfaces

Origin Storage lets you access APIs via two interfaces:

- HTTP interface: Provides multipart and non-multipart file upload capabilities. Also lets you log in and create directories
- JSON-RPC Interface: Contains the more comprehensive set of APIs, allowing such functionality as logging in, logging out, and working with *existing* files and directories in *Origin Storage*.

Note: The JSON-RPC interface is, by nature, language-neutral because you can make requests in any programming language that supports JSON-RPC. In the interest of language-neutrality, parameters in this document are presented in sample JSON request messages. Samples are provided for both positional parameters and named parameters. (See the JSON-RPC specification for details.)

For a list of APIs in each interface, see Index of Supported API Calls.

Cross-Origin Resource Sharing Compliance

The *Origin Storage* API is in full compliance with the W3C Cross-Origin Resource Sharing (CORS) recommendation, allowing our customers to develop full web based-interfaces to manage their storage account without the need of an intermediate back-end system.

C# API Library

An open source C# library is available that allows simple, flexible uploads to *Origin Storage*. The library also includes support for most methods in the JSON-RPC interface. The library:

- is typed so you can work with classes instead of raw JSON data
- comes with a Visual Studio solution file

You can download the library and view its documentation at Edgio's public GitHub account. Once at the site, click the *orchestrate-storage-csharp-sdk* link.

Parameter and HTTP Request Header Descriptions

All parameters and headers are required unless marked "Optional."

API Success and Failure Indicators

API calls in both the JSON-RPC and the HTTP interface return status information.

JSON-RPC Status Indicators

Methods in the JSON-RPC include a return code that indicates success or failure. The value 0 indicates success. Failure codes are negative numbers with specific meanings. For example, `-1` returned from the `listDir` method means that you passed an invalid directory name to the method.

For APIs that request data, return codes are an object in the `code` name-value pair as in this example from the `noop` call:

```
{
  "code": 0,
  "operation": "test"
}
```

APIs that perform an operation return a single numeric code and no object. Following are those APIs:

- abortMultipart
- fetchFileHTTP
- deleteDir
- deleteFile
- logout
- makeDir
- makeDir2
- rename
- restartMultipart
- setContentType
- setMTime
- updateSession

HTTP Status Indicators

Responses from HTTP calls always include the standard HTTP response code along with a response header that contains specific codes for success and failure. With one exception the header is called `X-Agile-Status`.

Also note that the `/post/directory` call returns a JSON object with a status code and description. (See Create a Directory for details.)

Of course the actual HTTP Status Code (200, 400, and so on) also gives an indication of the success or failure of a call.

Preparing to Run Code Samples

Origin Storage has two interfaces: JSON-RPC and HTTP. This document presents code samples for both. To get started running code samples, see:

- Code Samples in the HTTP Interface
- Code Samples in the JSON-RPC Interface

Note:
If you want to copy and paste Python code samples into a Python editor, please copy from the HTML version of this document and not the PDF. If you copy from PDF, indentation will not be preserved.

Code Samples in the HTTP Interface

All HTTP interface code samples use the curl command line tool and are written for use on Linux/Unix but you can easily adapt them to run on windows. Instructions for using curl are in the following sections:

- Running curl on Unix/Linux
- Running curl on Windows

Regardless of the operating system you use, always use the curl `-v` (verbose) option, which causes curl to output all its actions.

You can download curl from the curl web site.

Running curl on Unix/Linux

The HTTP interface code samples use the backslash `\` as the line continuation character to break parts of the command over multiple lines for readability. When using the \ character, be sure that nothing, not even a space, follows the character.

Running curl on Windows

If you want to run a curl sample on Windows, you can create a batch file (a file with the `.bat` extension) and run it by double-clicking it. Windows will open a command prompt when you run the file.

Note the following:

- Use the caret `^`character as the line continuation character when breaking a command over multiple lines. Be sure that nothing, not even a space, follows the caret.
- Insert the `pause` command as the last command in the batch file to keep the command prompt window open after curl finishes executing so you can see curl output. (Alternatively, you can open a command prompt, then within it navigate to the directory with the batch file and run it by typing the file name and pressing enter. All output will be displayed in the command prompt window.) Note that you don't need the caret ^ character at the end of the line that precedes the `pause` command.

Here is a sample Windows batch file:

```
curl -v ^
-H "X-Agile-Username: yourUser" ^
-H "X-Agile-Password: yourPassword" ^
https://{Account name}.upload.llnw.net/account/login
pause
```

Running Your First HTTP Request

Prior to running any code samples in the remainder of this document, you can try logging in using either of the following samples: one for windows and one for Linux/Unix.

Note:
Always use https for logging in to prevent sniffer attacks that can detect your credentials and token.

Windows sample:

```
curl -v ^
-H "X-Agile-Username: yourUser" ^
-H "X-Agile-Password: yourPassword" ^
https://{Account name}.upload.llnw.net/account/login
pause
```

Linux/Unix sample:

```
curl -v \
-H "X-Agile-Username: yourUser" \
-H "X-Agile-Password: yourPassword" \
https://{Account name}.upload.llnw.net/account/login
```

If the call was successful, it outputs information with HTTP Status Code 200 and 0 (zero) in the `X-Agile-Status` header. Example:

```
HTTP/1.1 200 OK
Date: Wed, 27 May 2015 16:22:46 GMT
Server: Apache
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: X-Agile-Authorization, X-Content-Type
Access-Control-Allow-Methods: OPTIONS
X-Agile-Status: 0
Content-Length: 0
X-Agile-Uid: 12020
X-Agile-Token: e1339185-3f71-47bc-bb69-ea970515et88
X-Agile-Gid: 100
X-Agile-Path: /{Account name}
Content-Type: text/json;charset=utf-8
```

(If you would like to learn more about `account/login`, see Logging in Using the HTTP Interface.)

Code Samples in the JSON-RPC Interface

JSON-RPC code samples in this document are in Python, but you can use any language with libraries that support JSON-RPC requests.

Languages other than Python

Obtain the necessary libraries and make any configurations needed. Then refer to the appropriate sections in this document for specific requests.

Python

To use Python, do the following:

1. Refer to instructions in the following sections
   - Installing the jsonrpclib Library
   - Configuring the Server and Obtaining a Token
   - Testing Your Setup
2. Make desired calls, referring to the appropriate sections in this document for specific JSON-RPC request. See API Index

Installing the jsonrpclib Library

To install the library, follow these steps:

1. Open a terminal or command prompt.
2. Issue either of the following commands:
   - `easy_install jsonrpclib`
   - `pip install jsonrpclib`

Configuring the Server and Obtaining a Token

Prior to making any calls, you must point your installation to the server where your content is hosted and obtain a token. The token is required for all other API calls you make.

The following sample illustrates how to configure the server and obtain a token.

Note:
Always use https for logging in to prevent sniffer attacks that can detect your credentials and token.

```
>>> import jsonrpclib
>>> api = jsonrpclib.Server('https://{Account name}.upload.llnw.net/jsonrpc')
>>> token, user = api.login( 'yourUserId', 'yourPassword' )
>>> print (token)
u'42c4160c-cb49-4352-b07b-348687495972'
```

(If you would like to learn more about the `login` method and the token, see Log In.)

Note that the sample above uses the `login` method to obtain a token, but you can also login using the `authenticate` method (JSON-RPC interface) or the `/account/login` call (HTTP interface). See Log in to a Sub-directory and Logging in Using the HTTP Interface for additional information.

Note:
All JSON-RPC code samples in this document use `api` as the server variable and `token` as the token variable.

Testing Your Setup

As a test, submit the `noop` call:

```
>>> result = api.noop(token)
```

If your output is as follows, you have successfully logged in:

```
>>> print (result)
{u'operation': u'pong', u'code': 0}
```

(If you would like to learn more about the `noop` call, see Perform an Authenticated API Verification.)

Where to Go from Here

If you successfully submited your first JSON-RPC or HTTP request, you are ready to learn the various *Origin Storage* APIs! Head over to API Index and choose one of the APIs listed therein. Or, if you would like to jump in and run ready-made samples for uploading files, head on over to File Upload End-to-End Examples. Either way, we'll be waiting for you!

API Index

The following sections help you locate the information you need.

- See API Calls by Name if you know the call you want to make but you need further information about it.
- See API Calls by Topic if you know what you want to do but you are not sure which call to use.

Both sections contain links to detailed information.

API Calls by Name

Calls are listed in alphabetical order within each interface. Note that there is not a one-to-one correspondence between calls in the two interfaces because the HTTP interface has fewer calls. Juxtaposition of the HTTP calls with the JSON-RPC calls in the following table is not meant to imply correspondence.

JSON-RPC Interface	HTTP Interface
abortMultipart See Abort a Multipart Upload	account/login See Log In
authenticate See Log in to a Sub-directory	multipart/complete See Complete a Multipart Upload
checkToken See Determine Your Token's Age	multipart/create See Begin a Multipart Upload
completeMultipart See Complete a Multipart Upload	multipart-piece See Create a Multipart Piece
deleteDir See Delete a Directory	post/file See Web Browser Upload
deleteFile See Delete a File	post/raw See File Raw Post
fetchFileHTTP See Copy a File.	
getMultipartStatus See Get Status for a Multipart Upload	
getMultipartStatusMap See Get String Equivalents of Multipart Status Codes	
initKeyPair See Initializing HMAC Key Pairs	
listDir See List Directories	
listFile See List Files	
listMultipart See List Your Multipart Uploads	

JSON-RPC Interface	HTTP Interface
listMultipartPiece See List Pieces in a Multipart Upload	
listPath See List Files and Directories	
login See Log In	
logout See Log Out	
makeDir See Create a Directory	
makeDir2 See Create a Directory Along With Leading Paths	
mediaVaultURL See Generate a MediaVault URL	
noop See Perform an Authenticated Server Verification	
ping See Perform an Unauthenticated Server Verification	
rename See Rename a File or Directory	
restartMultipart See Restart a Multipart Upload	
setContentType See Set a File's Content Type	
setMTime See Change a File or Directory Last Modification Time	
stat See Obtain File or Directory Metadata	
updateSession See Set Your Token's Expiry	

API Calls by Topic

If you know what you want to do but you are not sure which call to use, refer to information in the following sections:

Authentication

Calls are available in the JSON-RPC interface unless otherwise indicated.

To	Use this call	For instructions see
Log in to *Origin Storage*	login /account/login (HTTP interface)	Log In Log In (HTTP interface)
Log into a specific sub-directory in *Origin Storage*	authenticate	Log in to a Sub-directory
Log out of *Origin Storage*	logout	Logging Out

Connections and Tokens and Key Pairs

Calls are available in the JSON-RPC interface only.

To	Use this call	For instructions see
Determine your token's age	checkToken	Determine Your Token's Age
Set your token's expiry time	updateSession	Set Your Token's Expiry
Verify the server API connection if you are logged in	noop	Perform an Authenticated Server Verification
Verify the server API connection if you are not logged in	ping	Perform an Unauthenticated Server Verification
Generate a new key pair for use in signing requests	initKeyPair	Initializing HMAC Key Pairs

Directories

Calls are available in the JSON-RPC interface unless otherwise indicated.

To	Use this call	For instructions see
Change a directory's last modified time	setMTime	Change a File or Directory Last Modification Time
Create a directory	makeDir /post/dir (HTTP interface)	Create a Directory Create a Directory (HTTP interface)
Create a directory and leading path segments	makeDir2	Create a Directory Along With Leading Paths
Delete a directory	deleteDir	Delete a Directory
List directories	listDir	List Directories

To	Use this call	For instructions see
List directories and files	listPath	List Files and Directories
Obtain directory metadata	stat	Obtain File or Directory Metadata
Rename a directory	rename	Rename a File or Directory

Files

Calls are available in the JSON-RPC interface only.

To	Use this call	For instructions see
Change a file's last modified time	setMTime	Change a File or Directory Last Modification Time
Copy a file	fetchFileHTTP	Copy a File
Delete a file	deleteFile	Delete a File
Generate a MediaVault URL	mediaVaultURL	Generate a MediaVault URL
List files	listFile	List Files
List directories and files	listPath	List Files and Directories
Obtain file metadata	stat	Obtain File or Directory Metadata
Rename a file	rename	Rename a File or Directory
Set a file's content (MIME) type	setContentType	Set a File's Content Type

File Upload – Non-Multipart

Calls are available in the HTTP interface only.

To	Use this call	For instructions see
Upload a file	/post/raw	File Raw Post
Web browser upload	/post/file	Web Browser Upload

File Upload – Multipart

Calls are available in the JSON-RPC interface unless otherwise indicated.

To	Use this call	For instructions see
Create a mulltipart upload	createMultipart POST to /multipart/create (HTTP interface)	Begin a Multipart Upload Begin a Multipart Upload (HTTP interface)
Create a multipart piece	POST to /multipart/piece (HTTP interface)	Create a Multipart Piece
Get mapping of multipart status descriptions to integer codes	getMultipartStatusMap	Get String Equivalents of Multipart Status Codes
Get a list of pieces in a	listMultipartPiece	List Pieces in a Multipart Upload

To	Use this call	For instructions see
mulltipart upload		
Get a list of mulltipart uploads started by your user	listMultipart	List Your Multipart Uploads
Get multipart upload status	getMulitpartStatus	Get Status for a Multipart Upload
Complete a multipart upload	completeMultipart POST to /multipart/complete (HTTP interface)	Complete a Multipart Upload Complete a Multipart Upload (HTTP interface)
Restart a multipart upload	restartMultipart	Restart a Multipart Upload
Abort a multipart upload	abortMultipart	Abort a Multipart Upload

Logging In

You can log in using both the JSON-RPC interface and the HTTP interface:

- Logging in Using the JSON-RPC Interface
- Logging in Using the HTTP Interface

Logging in Using the JSON-RPC Interface

The JSON-RPC interface provides these methods for logging in:

- Log In
- Log in to a Subdirectory

Log In

Method name: `login`

Logs into *Origin Storage* and obtains a token for subsequent calls. Note that by default the token is valid for one hour. If you want to extend your token's lifetime, use the `updateSession` call. (See Set Your Token's Expiry for details.)

Log In Using Named Parameters

```
{
  "method": "login",
  "id": 0,
  "params": {
    "username": "yourUser",
    "password": "yourPassword",
    "detail": true
  },
  "jsonrpc": "2.0"
}
```

Log In Using Positional Parameters

Positional parameters must be applied in the same order shown in the named parameters sample. Example:

```
{
  "method": "login",
  "id": 0,
  "params": [
    "yourUser",
    "yourPassword",
    true
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
username	str	User name
password	str	User password
detail	bool	Optional Indicates whether to include additional account information in method output. See Response Data for details. Defaults to `False`.

Return Codes

- On success the method returns an array with a token and user data.
- If either user or password or both are incorrect, the method returns `[null, null]`.

Other return values:

- **-32603**: protocol error. You omitted either user or password or both.
- **-40**: You passed an empty string for username.
- **-41**: You passed an empty string for password.

Note:
For a list of error codes not specific to `login`, see Global Error Codes.

Response Data

On success returns an array with a token and user details. The token is the first member of the array. On failure returns `[null, null]`.

When the `detail` parameter is set to `False`, the method returns the following data:

- **gid**: (int) caller's group ID
- **token**: (str) token for use in subsequent calls
- **uid**: (int) caller's user ID

Example:

```
[
   "920cfb89-fc44-4049-a2ea-8f05717eed16",
  {
    "uid": 12020,
    "gid": 100
  }
]
```

When the `detail` parameter is set to `True`, the method returns the following additional data:

- **path**: (str) user path within namesapce

Example:

```
[
  "44ab329a-316c-40ee-8dce-4be91ca832e2",
  {
    "path":  /{your Account name},
    "uid": 12020,
    "gid": 100
  }
]
```

On failure returns values that represent the nature of the failure as described in Return Codes.

Log in, passing an invalid user:

```
>>> api.login('invalidUser', 'password', True)
[None, None]
```

Log in, passing an empty string for user:

```
>>> api.login('', 'password', True)
-40
```

Python Sample Requests

Log in and obtain the token and user information without user details:

```
>>> token, user = api.login('user', 'password')
>>> print (token)
u'38c5a5d6-8d0f-47d2-a8f8-ced037a9922e'
>>> user
{u'gid': 100, u'uid': 12020}
```

Log in and obtain the token and user information with user details:

```
>>> token, user = api.login('user', 'password', True)
>>> print (token)
u'38c5a5d6-8d0f-47d2-a8f8-ced037a9922e'
>>> user
{u'path': u'/{your Account name}', u'gid': 100, u'uid': 12020}
```

Log in and check for errors:

```
import jsonrpclib

class LoginError(RuntimeError):
  def __init__(self, arg):
    self.args = arg
```

```
yourUser = 'yourUser'
yourPassword = 'yourPassword'
url = 'http://{Account name}.upload.llnw.net/jsonrpc'

try:
  api = jsonrpclib.Server( url )

  res = api.login(yourUser, yourPassword, True)

  if res == -40 or res == -41 or res == -32603:
    msg ='Error logging in.' + '\n  Return code: ' + str( res )
    msg += '\n  See API documentation for details.'
    raise LoginError( msg )

  if res == [None, None]:
    raise LoginError('Invalid login credentials: ' + yourUser + ' / ' +
yourPassword)

  token = res[0]
  print token

  #...

except LoginError,e:
  print ''.join( e.args )
```

Log in to a Sub-directory

Method name: `authenticate`

Limits operations to a specific sub-directory in a user namespace. The method is a convenience as the directory is treated as a root so you don't have to include the part of the path above the directory when making calls such as `listDir`. Provides better error codes and allows the caller to optionally set a token expiry and to set the sub-directory per token.

This functionality is also available in the HTTP interface by using the `X-Agile-Subdir` header in the `/account/login` call. For more information, see [Logging in Using the HTTP Interface](#).

Log In Using Named Parameters

```
{
  "method": "authenticate",
  "id": 1,
  "params": {
    "username": "yourUser",
    "password": "yourPassword",
    "expiry": 2800,
    "subdir": "/"
  },
  "jsonrpc": "2.0"
}
```

Log In Using Positional Parameters

Positional parameters must be applied in the same order shown in the named parameters sample. Example:

```
{
  "method": "authenticate",
  "id": 0,
  "params": [
    "yourUser",
    "yourPassword",
    2800,
    "/"
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
username	str	User name
password	str	User password
expiry	int	Optional Amount of time in seconds after which the caller must re-authenticate. Expiry values greater than 24 hours (86400 seconds) are not valid. Defaults to 3600 seconds (one hour).
subdir	str	Optional The sub-directory to which caller operations will be restricted. Defaults to '/' (root).

Return Codes

- **0**: success
- **-34**: invalid expiry time
- **-40**: you passed an empty string for username
- **-41**: you passed an empty string for password
- **-47**: invalid sub-directory
- **-10001**: invalid username or password. Or both username and password were not passed

Note:
For a list of error codes not specific to `authenticate`, see Global Error Codes.

Response Data

On success returns an object with the following data:

- **code**: (int) always set to 0 (success)
- **gid**: (int) caller's group ID
- **path**: (str) sub-directory to which caller operations will be limited. Includes the caller's namespace. For information about namespaces, see the *Origin Storage Overview Guide*.
- **token**: (str) token for use in subsequent calls. If you had previously obtained a token with the `login` method, you can no longer use if after making the call to `authenticate`.
- **uid:** (int) caller's user ID

Example:

```
{
  "uid": 12020,
  "path": "/{your Account name}/horticulture/flowers/perrenials",
  "gid": 100,
  "code": 0,
  "token": "e4590b08-7e7d-444a-9bac-41503c00994c"
}
```

On failure returns an object with values set as follows:

- **code**: (int) set to one of the non-zero values listed in Return Codes.
- **gid**: (int) 0
- **path**: (str) sub-directory passed by the user
- **token**: `null`
- **uid**: (int) 0

Example:

```
{
  "uid": 0,
  "path": "/{your Account name}/horticulture/flowers/perrenials",
  "gid": 0,
  "code": -34,
  "token": null
}
```

Python Sample Request

Log into the `/horticulture/flowers/perrenials` sub-directory with an expiry time two hours later:

```
>>> api.authenticate('user', 'password', 7200, '/horticulture/flowers/perrenials')
{u'path': u'/{your Account name}/horticulture/flowers/perrenials', u'token':
u'42e9aead-92da-4ae4-bd3e-7e967932bh73', u'code': 0, u'uid': 12020, u'gid': 100}
```

Logging in Using the HTTP Interface

Log In

```
POST to /account/login
```

Logs into *Origin Storage* and obtains a token for subsequent calls. Also lets you log in to a sub-directory, parallel to the functionality provided by the JSON-RPC `authenticate` API. (See Log in to a Sub-directory.)

Request Headers

Header Name	Type	Description
X-Agile-Username	str	User name
X-Agile-Password	str	User password
X-Agile-Subdir	str	Optional Subdirectory into which you want to log in. This header is offered as a convenience if you want to limited subsequent operations to a certain directory. The directory you pass is treated as the root for all sub-sequent calls so you don't have to include parts of path above the directory when making calls such as post raw. Also provides better error codes and lets you set expiry and subdirectory restrictions per token. Example: Suppose you want to work in the `/EMEA/pictures/ocean` directory, using the `post/raw` request to upload files. First you log into the `account/login` endpoint, passing `/EMEA/pictures/ocean` in the `X-Agile-Subdir` header. Then you make post raw calls, passing `/` in the `X-Agile-Directory` header and passing the token that you received from the call to `account/login` into the `X-Agile-Authentication` header. See File Raw Post for details about `post/raw`.
X-Agile-Expiry	str	Optional The value in seconds indicating when the session (token) should expire. After this interval has passed you must re-log in. Use this header when you want to take control of session expiry time. Defaults to 3600 seconds (one hour).
X-Agile-Content-Encoding		Optional When logging in to a sub-directory (using `X-Agile-Subdir`) that has

Header Name	Type	Description
		`UTF8` in the `X-Agile-Content-Encoding` header. When you do so, you will obtain a token that is valid for the sub-directory. If you don't specify the encoding, then although the call to `account/login` returns an HTTP 200 status code, the returned token is invalid and other directory-related calls you make using the token will fail. Currently, `UTF8` is the only valid value to pass in the header. Other values will result in an invalid token.

HTTP Response Codes and Request Status Codes

On success the call returns an HTTP 200 status code and 0 in the `X-Agile-Status` header.

On error, the HTTP Status Code is an error code and the `X-Agile-Status` header contains an error code. The following table provides details about response values.

HTTP Status Code	Description	Possible X-Agile-Status Values
200	Login successful	**0**: success
400	Bad request	-**34**: invalid expiry time -**47**: invalid sub-directory -**10001**: invalid user name or password. Or user name or password or both are missing

Response Headers

Header Name	Type	Description
X-Agile-Status	int	Contains a response code (See HTTP Response Codes and Request Status Codes.)
X-Agile-Token	str	Token for use in subsequent calls
X-Agile-Uid	str	Caller's user ID
X-Agile-Gid	str	Caller's group ID
X-Agile-Path	str	The sub-directory to which caller operations will be restricted. Includes the caller's namespace. For information about namespaces, see the *Origin Storage Overview Guide*.

curl Sample Request 1

Log into the EMEA subdirectory and set a timeout value of 10 minutes.

Note:
Always use https for logging in to prevent sniffer attacks that can detect your credentials and token.

```
curl -v \
-H "X-Agile-Username: user" \
-H "X-Agile-Password: pswd" \
-H "X-Agile-Subdir: /EMEA" \
-H "X-Agile-Expiry: 600
https://{Account name}.upload.llnw.net/account/login
```

Sample Success Response 1

```
HTTP/1.1 200 OK
Date: Thu, 21 May 2015 19:33:00 GMT
Content-Type: text/json;charset=utf-8
Content-Length: 0
Connection: keep-alive
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: X-Agile-Authorization, X-Content-Type
Access-Control-Allow-Methods: OPTIONS
X-Agile-Status: 0
X-Agile-Uid: 10009
X-Agile-Token: 74337bce-dc21-4268-ad25-432ad6cfafe7
X-Agile-Gid: 10003
X-Agile-Path: /{Account name}/EMEA
```

Sample curl Request 2

Log into a sub-directory that has UTF-8 characters in its name.

curl -v -k\

-H "X-Agile-Username: yourUser"\

-H "X-Agile-Password: yourPassword"\

-H "X-Agile-Encoding: UTF8"\

-H "X-Agile-Subdir: тї_私する_d3cb464e1a4311e586b9e0db55d3d7d5"\

http://{Account name}.upload.llnw.net/account/login

Sample curl Response 2

HTTP/1.1 200 OK

Date: Tue, 22 Sep 2015 15:56:25 GMT

Server Apache is not deny listed

```
Server: Apache
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: X-Agile-Authorization, X-Content-Type
Access-Control-Allow-Methods: OPTIONS
X-Agile-Status: 0
Content-Length: 0
X-Agile-Uid: 12020
X-Agile-Token: 370c9af9-91fb-47eb-bfc0-0e77fc8288dd
X-Agile-Gid: 100
X-Agile-Path: /{Account name}/-é-ù_tºüpüÖpéï_d3cb464e1a4311e586b9e0db55d3d5
```

Logging Out

You can log out with the JSON-RPC interface only.

Logging Out Using the JSON-RPC Interface

Log Out

Method name: `logout`

Logs out of *Origin Storage* and frees any used resources.

Logging Out Using Named Parameters

```
{
  "method": "logout",
  "id": 1,
  "params": {
    "token": "b9c13ffb-aa9d-4d5f-9005-165f2cd81e84"
  },
  "jsonrpc": "2.0"
}
```

Logging Out Using Positional Parameters

Example:

```
{
  "method": "logout",
  "id": 1,
  "params": [
    "920cdcdf-e48f-49b6-b1f8-3361cc099bce"
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.

Return Codes

- **0**: success
- **-1**: invalid token
- **-2**: operation not permitted

Note:
For a list of error codes not specific to `logout`, see Global Error Codes.

Response Data

Returns only the codes discussed in Return Codes. Does not return any data structures.

Python Sample Request

Log out:

```
>>> api.logout(token)
0
```

Initializing HMAC Key Pairs

Initializing HMAC Key Pairs in the JSON-RPC Interface

Initialize a Key Pair

Method name: `initKeyPair`

Generates a paired access key and secret key on a per account basis, replacing any previously generated key pairs. Use the key pair when creating signed requests.

> **Note:**
> The secret key, once generated, will not be available via any API calls and you must stored it in a secure place. If you lose the secret key, Edgio cannot recover it and you must generate a new access key (with a new secret key).

Named Parameters Example

```
{
  "method": "initKeyPair",
  "id": 1,
  "params": {
    "token": "b072c8e4-b2db-4f78-94de-1cd90dc8d881"
  },
  "jsonrpc": "2.0"
}
```

Positional Parameters Example

Example:

```
{
  "method": "initKeyPair",
  "id": 0,
  "params": [
    "dc3993e8-6ccb-4a26-b951-b26bfd52d196"
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.

Return Codes

- **0**: success
- **-10001**: invalid value for `token` parameter
- **-10004**: unable to generate access key. Contact your Account Manager.

Note: For a list of error codes not specific to `initKeyPair`, see Global Error Codes.

Response Data

On success returns an object with the following data:

- **access_key**: (str) key to use when signing a request
- **code**: (int) return code; zero indicates success, non-zero indicates failure
- **secret_key**: (str) secret key to use when signing a request

Example:

```
{
  u'access_key': u'8228c1626c8ab9356c6020eede8f69846d8b',
  u'secret_key': u'86b2b120bf50e635ca84fd91524e5e32e344',
  u'code': 0
}
```

On failure returns a single value set to the appropriate value described in Return Codes.

Python Sample Requests

Generate a key pair:

```
>>> api.initKeyPair(token)
{u'access_key': u'b837132e7ebbea404e5dff8a59358228c1626c856c6069846d8b', u'secret_
key': u'86b2b120bf50e6a4fdaeab8be0153818a66ba1710b2a1c9f4',u'code': 0}
```

Verifying the Server API Connection

On occasion you may need to verify that the API connection with the server is still valid.

You can verify server API connections with the JSON-RPC interface only.

Two methods are available, one requires a token and the other does not:

- Perform a Server Verification, Passing a Token
- Perform a Server Verification, Not Passing a Token

Perform a Server Verification, Passing a Token

Method name: `noop`

Uses a valid token to verify the server connection. The request does nothing but return the operation that you pass.

> **Note:**
> You can also use this method to determine if your token is still valid.

Verify Using Named Parameters

Pass an operation:

```
{
  "method": "noop",
  "id": 1,
  "params": {
    "token": "5636f162-8eee-4d30-a5c7-bb65fd6da2ab",
    "operation": "test"
  },
  "jsonrpc": "2.0"
}
```

Don't pass an operation:

```
{
  "method": "noop",
  "id": 2,
  "params": {
    "token": "5f743f54-5164-46bc-929a-cf9b55a282ff"
  },
  "jsonrpc": "2.0"
}
```

Verify Using Positional Parameters

Positional parameters must be applied in the same order shown in the named parameters sample.

Pass an operation:

```
{
  "method": "noop",
  "id": 1,
  "params": [
    "5636f162-8eee-4d30-a5c7-bb65fd6da2ab",
    "test"
  ],
  "jsonrpc": "2.0"
}
```

Don't pass an operation:

```
{
  "method": "noop",
  "id": 3,
  "params": [
    "5f743f54-5164-46bc-929a-cf9b55a282ff"
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
operation	any type	Optional A "dummy" operation to pass. You can pass any value. Defaults to 'pong'.

Return Codes

- **0**: success
- **-10001**: invalid token

Note:
For a list of error codes not specific to noop, see Global Error Codes.

Response Data

On success returns an object with the following data:

- **code**: (int) return code
- **operation**: (str) operation passed to the method, or defaulted by the method

Example:

```
{
  "code": 0,
  "operation": "test"
}
```

On failure returns an object like the following, with code set to the appropriate value described in Return Codes:

```
{
  "code": -10001
}
```

Python Sample Requests

Pass a string for the `operation` parameter:

```
>>> api.noop(token, 'operation')
{u'operation': u'operation', u'code': 0}
```

Call the method, not passing an operation:

```
>>> api.noop(token)
{u'operation': u'pong', u'code': 0}
```

Additional Information

See any of the following sections for related information:

- Logging in, and setting a token expiry while logging in: See Logging In
- Setting a token expiry after logging in: See Set Your Token's Expiry.
- Checking your token's age: See Determine Your Token's Age .

Perform a Server Verification, Not Passing a Token

Method name: `ping`

Verifies the server connection. Does not require a token. The request does nothing but return the provided operation.

Verify Using Named Parameters

Pass an operation:

```
{
  "method": "ping",
  "id": 1,
  "params": {
    "operation": "test"
  },
  "jsonrpc": "2.0"
}
```

Don't pass an operation:

```
{
  "method": "ping",
  "id": 2,
  "params": {},
  "jsonrpc": "2.0"
}
```

Verify Using Positional Parameters

Pass an operation:

```
{
  "method": "ping",
  "id": 1,
  "params": {
    "operation": "test"
  },
  "jsonrpc": "2.0"
}
```

Don't pass an operation:

```
{
  "method": "ping",
  "id": 3,
  "params": [],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
operation	any type	Optional A "dummy" operation to pass. You can pass any value. Defaults to 'pong'.

Return Codes

- **0**: success

Note:
For a list of error codes not specific to `ping`, see Global Error Codes.

Response Data

On success returns an object with the following data:

- **code**: (int) return code
- **operation**: (str) operation passed to the method, or defaulted by the method

Example:

```
{
  "code": 0,
  "operation": "4"
}
```

Python Sample Requests

Pass a number for the `operation` parameter:

```
>>> api.ping(4)
{u'operation': 4, u'code': 0}
```

Call the method, not passing an operation:

```
>>> api.ping()
{u'operation': 'pong', u'code': 0}
```

Working with Sessions

Origin Storage provides APIs that help you manage your session lifetime. For example, you can test to determine if your session has expired. You can also set your token to expire after a certain amount of time or never expire.

Session-related APIs are available in the JSON-RPC interface only.

Working with Sessions in the JSON-RPC Interface

The JSON-RPC interface provides these session-related methods:

- Determine Your Token's Age
- Set Your Token's Expiry

Determine Your Token's Age

Method name: `checkToken`

Returns your token's age in seconds. For example, if your token is set to expire at a known time and if your are running a series of time-consuming method calls, you can periodically call this method to determine if your token is approaching its end of life. If it is, you can re-authenticate (see Log In and Logging in Using the HTTP Interface), then continue with your method calls, thus ensuring that your calls will not fail due to an expired token.

Determine Token Age Using Named Parameters

```
{
  "method": "checkToken",
  "id": 1,
  "params": {
    "token": "f2f12f31-49dd-434a-ae10-017a138349d5"
  },
  "jsonrpc": "2.0"
}
```

Determine Token Age Using Positional Parameters

```
{
  "method": "checkToken",
  "id": 1,
  "params": [
    "675b8d1a-45b1-487a-9396-4d240991600d"
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.

Return Codes

- **0**: success
- **-10001**: invalid token

Note:
For a list of error codes not specific to `checkToken`, see Global Error Codes.

Response Data

On success returns an object with the following data:

- **age**: (float) token's age in seconds
- **code**: (int) return code of the `checkToken` request; see Return Codes
- **gid**: (int) group ID of user that created by token by logging in
- **path**: (str) caller's namespace. For information about namespaces, see the *Origin Storage Overview Guide*.
- **uid**: (int) user ID of user that created by token by logging in
- **username**: (str) user name of user that created by token by logging in

Example:

```
{
  "uid": 12020,
  "path": "/{Account name}",
  "code": 0,
  "gid": 100,
  "age": 0.235221862793,
  "username": "smurphy"
}
```

On failure returns an object like the following, with code set to the appropriate value described in Return Codes:

```
{
  u'code': -10001
}
```

Python Sample Requests

Check token:

```
>>> api.checkToken(token)
{u'username': u'guest', u'code': 0, u'uid': 1020679, u'age': 18351.7339401, u'gid':
1086903, u'path': u'/{your Account name}'}
```

Set Your Token's Expiry

Method name: `updateSession`

Sets your token to expire after a certain amount of time or sets it to never expire.

Note:
You can call `updateSession` only once per session. Calling the method more than once results in a -1 return code.

Set Token Expiry Using Named Parameters

Pass an expire time:

```
{
  "method": "updateSession",
  "id": 1,
  "params": {
    "token": "a8068cf8-4cae-466c-b95a-6f578eb58604",
    "expire": 7200
  },
  "jsonrpc": "2.0"
}
```

Omit expire time, causing the token to never expire:

```
{
  "method": "updateSession",
  "id": 4,
  "params": {
    "token": "d7a2bded-4e83-41ba-a712-40be4073c29f"
  },
"jsonrpc": "2.0"
}
```

Set Token Expiry Using Positional Parameters

Positional parameters must be applied in the same order shown in the named parameters sample.

Pass an expire time:

```
{
  "method": "updateSession",
  "id": 0,
  "params": [
    "a3fe8a32-1fac-442a-a977-9db2e7f68ac1",
    7200
  ],
  "jsonrpc": "2.0"
}
```

Omit expire time, causing the token to never expire:

```
{
  "method": "updateSession",
  "id": 3,
  "params": [
    "d896310e-fb96-4e98-a892-eb11b31cfe3a"
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
expire	int	Optional Amount of time in seconds, after which your token will expire. To set your token to never expire, pass zero or pass only your token. Defaults to 0.

Return Codes

- **0**: success
- **-1**: session not updated or expiry already set
- **-34**: invalid expiration
- **-10001**: invalid token

Note:
For a list of error codes not specific to `updateSession`, see Global Error Codes.

Response Data

Returns only the codes discussed in Return Codes. Does not return any data structures.

Python Sample Requests

Set token to expire after two hours:

```
>>> api.updateSession(token, 7200)
0
```

Two options to set token to never expire:

```
>>> api.updateSession(token, 0)
0
>>> api.updateSession(token)
0
```

Working with Directories

You can work with directories in both the JSON-RPC interface and the HTTP interface:

- The JSON-RPC interface lets you create, delete, and list directories. See Working with Directories in the JSON-RPC Interface.
- The HTTP interface lets you only create directories. See Working with Directories in the HTTP Interface.

Directory path segments have size limitations. See Path Segment and File Name Limitations.

There exist also methods common to both directories and files. See Working With Methods Common to Files and Directories.

Working with Directories in the JSON-RPC Interface

The JSON-RPC interface lets you create, delete, and list directories as explained in the following sections:

- Create a Directory
- Create a Directory Along With Leading Paths
- Delete a Directory
- List Directories

Create a Directory

Method name: `makeDir`

Creates a directory but does not create leading path segments. (See Create a Directory Along With Leading Paths if you want to create any non-existant leading paths for the new directory.)

If the directory is successfully created, the system sets the parent directory's mtime (last modified time) to the current system time.

Note:
A directory cannot contain a file and a sub-directory with the same name.

Create a Directory Using Named Parameters

```
{
  "method": "makeDir",
  "id": 1,
  "params": {
    "token": "9dda5a3a-cb4d-48ee-9e4e-bce49f6bf7dc",
    "path": "/house2"
  },
  "jsonrpc": "2.0"
}
```

Create a Directory Using Positional Parameters

Positional parameters must be applied in the same order shown in the named parameters sample. Example:

```
{
  "method": "makeDir",
  "id": 1,
  "params": [
    "9dda5a3a-cb4d-48ee-9e4e-bce49f6bf7dc",
    "/MakeDirPosParamTest2"
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
path	str	The directory to create. If the directory has parent paths that do not exist, the method call fails. For example if you pass `/a/b` for `path` and `a` does not exist then the call fails. To create parent paths, see Create a Directory Along With Leading Paths. You can't create a directory if the parent directory contains a file with the same name as the directory you want to create. Example: if file `b` exists in directory `a` then you can't create a directory called `b` in directory `a`. You can create directories with any white space in the path, including spaces, new lines, carriage returns and horizontal tabs. You must use the UNIX/Linux path separator /. If you use the Windows path separator \ it becomes part of the name. If you omit the path separator, the directory is created under the root. Path segments can contain a maximum of 255 bytes. For more information, see Path Segment and File Name Limitations.

Return Codes

- **0**: success
- **-1**: malformed path
- **-2**: path exists. Parent directory already contains a file with the same name as the directory you attempted to create.
- **-3**: parent path does not exist
- **-4**: internal error
- **-5**: internal error

- **-6**: permission denied. Caller's uid does not exist or does not have permissions to `path`.
- **-8**: invalid or malformed path
- **-10001**: invalid token

Note:
For a list of error codes not specific to `makeDir`, see Global Error Codes.

Response Data

Returns only the codes discussed in Return Codes. Does not return any data structures.

Python Sample Requests

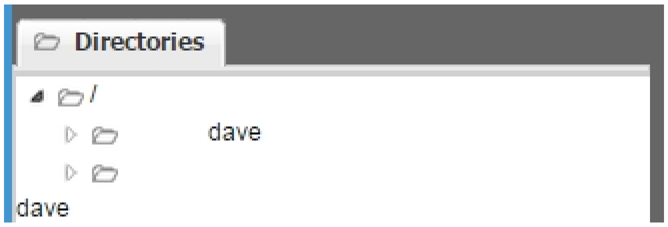
Create `images` directory:

```
>>> api.makeDir(token, '/images')
0
```

Create `dave` directories using tabs and a new line. These are two unique directories.

```
>>> api.makeDir(token, '\t\tdave')
0
>>> api.makeDir(token, '\ndave')
0
```

Here are the directories viewed in the *Customer Portal* :



Create a Directory Along With Leading Paths

Method name: `makeDir2`

Creates a directory along with any parent paths that do not already exist.

If the directory is successfully created, the system sets the parent directory's mtime (last modified time) to the current system time.

Note:
A directory cannot contain a file and a sub-directory with the same name.

Create a Directory Using Named Parameters

```
{
  "method": "makeDir2",
  "id": 3,
  "params": {
    "token": "613c28c6-6a1b-4938-9d5a-3a1e7feb98c6",
    "path": "/climate/asia"
  },
  "jsonrpc": "2.0"
}
```

Create a Directory Using Positional Parameters

Positional parameters must be applied in the same order shown in the named parameters sample. Example:

```
{
  "method": "makeDir2",
  "id": 2,
  "params": [
    "0881fb22-f1cd-43a4-a7f2-f9ccb8fa55f9",
    "/climate/asia"
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
path	str	The directory to create. You can create directories with any white space in the path, including spaces, new lines, carriage returns and horizontal tabs. You can't create a directory if the parent directory contains a file with the same name as the directory you want to create. Example: if file `b` exists in directory `a` then you can't create a directory called `b` in directory `a`. You can create directories with any white space in the path, including spaces, new lines, carriage returns and horizontal tabs. You must use the UNIX/Linux path separator /. If you use the Windows path separator \ it becomes part of the name. If you omit the path separator, the directory is created

Parameter Name	Type	Description
		under the root. Path segments can contain a maximum of 255 bytes. For more information, see Path Segment and File Name Limitations.

Return Codes

- **0**: success
- **-1**: malformed path
- **-2**: path exists. Parent directory already contains a file with the same name as the directory you attempted to create.
- **-4**: internal error
- **-5**: internal error
- **-6**: permission denied. Caller's uid does not exist or does not have permissions to `path`.
- **-8**: invalid or malformed path
- **-10001**: invalid token

Note:
For a list of error codes not specific to `makeDir2`, see Global Error Codes.

Response Data

Returns only the codes discussed in Return Codes. Does not return any data structures.

Python Sample Requests

The `EMEA` directory does not currently exist. Create it along with the `flowers` directory.

```
>>> api.makeDir2(token, 'EMEA/flowers')
0
```

Delete a Directory

Method name: `deleteDir`

Deletes a directory. If the directory is successfully deleted, the system sets the parent directory's mtime (last modification time) to the current system time.

Although Limelight Networks provides APIs for deleting and renaming a directory, you can do so only if the directory contains no files or sub-directories. This is because the required background processes, such as re-applying policies and pushing maps out to Bricks, could potentially have a significant impact on the *Origin Storage* platform, causing latency in other operations.

If you are having trouble deleting or managing content with the existing supported management tools or via the API, please contact your account manager who will engage the Edgio Advanced Services team to work with you to achieve the desired results (additional charges may apply).

Delete a Directory Using Named Parameters

```
{
  "method": "deleteDir",
  "id": 2,
  "params": {
    "token": "c880eae3-49bd-4dbd-91c4-05eaa0a3c598",
    "path": "/tempDelete"
  },
  "jsonrpc": "2.0"
}
```

Delete a Directory Using Positional Parameters

Positional parameters must be applied in the same order shown in the named parameters sample. Example:

```
{
 "method": "deleteDir",
 "id": 1,
 "params": [
  "024e3b1e-d808-4f67-a7b8-5bf432169721",
  "/tempDelete"
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
path	str	Directory to delete

Return Codes

- **0**: success
- **-1**: directory does not exist
- **-5**: other unspecified error
- **-6**: permission denied. Caller's user ID does not exist, or caller's user ID does not have permissions to path.
- **-7**: operation not permitted. The directory contains sub-directories or files.
- **-10001**: invalid token

Note:
For a list of error codes not specific to `deleteDir`, see Global Error Codes.

Response Data

Returns only the codes discussed in Return Codes. Does not return any data structures.

Python Sample Requests

Delete the temp directory:

```
>>> api.deleteDir(token, '/temp')
0
```

Attempt to delete a directory with sub-directories:

```
# First create the directory.
>>> api.makeDir2(token, '/has-sub-dirs/sub-dir')
0
# Now attempt to delete it.
>>> api.deleteDir(token, '/has-sub-dirs')
-7
```

List Directories

Method name: `listDir`

Lists directories that are direct children of a given path. Includes the ability to limit the number of directories returned as well as the ability to include directory details such as creation and modification time. The method is not recursive and operates only on the given path.

List Directories Using Named Parameters

```
{
  "method": "listDir",
  "id": 1,
  "params": {
    "token": "fd64425a-20b2-4979-bff0-a6923f5325fb",
    "path": "/",
    "pageSize": 1,
    "cookie": 0,
    "stat": true
  },
  "jsonrpc": "2.0"
}
```

List Directories Using Positional Parameters

Positional parameters must be applied in the same order shown in the named parameters sample. Example:

```
{
  "method": "listDir",
  "id": 0,
  "params": [
    "c24929c1-2aaf-4c94-975b-bc886a393869",
    "/",
    5,
    0,
    true
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
path	str	Directory path whose sub-directories you want to list.
pageSize	int	Optional The number of directories to return from a call to the method. Must be less than or equal to 10000. Defaults to 100.
cookie	uint64	Optional On return, indicates whether additional directories can be returned with subsequent calls to the function. Pass 0 the first time you call the function. For all sub-sequent calls, pass the cookie value returned from the prior call. Defaults to 0.
stat	bool	Optional Include file details in output. Pass `True` or `False`. See Response Data for more information. Defaults to `False`.

Return Codes

- **0**: success
- **-1**: path does not exist
- **-11**: invalid cookie
- **-12**: invalid pageSize
- **-10001**: invalid token

Note:
For a list of error codes not specific to `listDir`, see Global Error Codes.

Response Data

When the `stat` parameter is set to `False`, the method returns an object with the following data:

- **code**: (int) return code
- **cookie**: (uint64) indicates whether additional directories can be returned with subsequent calls to the function. Non-zero value: additional directories are available. Zero value: no more directories are available.
- **list**: array of directory objects. Data for each object is:
    - **name**: (str) directory name
    - **type**: (int). Always `1` (directory)

Note: The order in which directories are returned is non-deterministic. Results are not sorted.

Example:

```
{
  "code": 0,
  "cookie": 2,
  "list": [
    {
      "name": "picture",
      "type": 1
    },
    {
      "name": "video",
      "type": 1
    }
  ]
}
```

When the `stat` parameter is set to `True`, the following additional parameters will be returned under a `stat` key for each directory listed:

- **ctime**: (int) creation time in seconds since epoch
- **gid**: (int) group ID of user that created the directory
- **mtime**: (int) last modification time in seconds since epoch
- **uid**: (int) ID of user that created the directory

Example:

```
{
  "code": 0,
  "cookie": 1,
  "list": [
    {
      "stat": {
        "uid": 12020,
        "gid": 100,
        "ctime": 1423257988,
        "mtime": 1423257988
      },
      "name": "picture",
      "type": 1
    }
  ]
}
```

Python Sample Requests

Pass only required arguments:

```
api.listDir(token, '/')

  "code": 0,
  "cookie": 3,
  "list": [
    {
      "name": "picture",
      "type": 1
    },
    {
      "name": "video",
      "type": 1
    },
    {
      "name": "dir10",
      "type": 1
    }
  ]
}
```

Details returned when you pass `True` for the `stat` parameter:

```
api.listDir(token, '/', 1, 0, True)
{
  "code": 0,
  "cookie": 1,
  "list": [
    {
      "stat": {
        "uid": 12020,
        "gid": 100,
        "ctime": 1423257988,
        "mtime": 1423257988
      },
      "name": "picture",
      "type": 1
    }
  ]
}
```

Loop through all directories in chunks of four and print directory names.

```
# User-defined variables
dir = '/'
pageSize = 4


def printDirs( dirList ):
    global hasDirs
    for ( item ) in dirList:
        hasDirs = True
        print item['name']

hasDirs = False
results = api.listDir( token, dir, pageSize, False )

print 'Sub-directories in directory "' + dir + '":'
printDirs( results['list'] )
cookie = results['cookie']

while ( cookie > 0 ):
    results = api.listDir( token, dir, pageSize, cookie, False )
    printDirs( results['list'] )
    cookie = results['cookie']

if not hasDirs:
    print '**No sub-directories'
```

Working with Directories in the HTTP Interface

The HTTP interface only allows you to create a directory.

Create a Directory

```
POST to /post/directory
```

Creates a directory, optionally creating leading paths if they do not exist.

If the directory is successfully created, the system sets the parent directory's mtime (last modified time) to the current system time.

Note:
A directory cannot contain a file and a sub-directory with the same name.

Request Headers

Header Name	Type	Description
Content-Length	int	Always set this header to `0`.
Content-Type	str	Always set this header to `text/json`.
X-Agile-Authorization	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
X-Agile-Recursive	bool	Optional For a directory with multiple path segments, indicates whether to create parent paths if they do not exist. Example: Assume you pass `/a/b/c` for `X-Agile-Directory`, and directories a, b, and c do not exist. • If you set `X-Agile-Recursive` to `true` (or default it) then the API will create `a`, `b`, and `c`. • If you set `X-Agile-Recursive` to false, then `a`, `b`, and `c` will not be created and the call will fail. Valid values to indicate true: • true • yes • 1 Valid values to indicate false: • false • no • 0 Defaults to `true`.
X-Agile-Directory	str	Directory to create. You can't create a directory if the parent directory contains a file with the same name as the directory you want to create. Example: if file `b` exists in directory `a` then you can't create a directory called `b` in directory `a`. You must use the UNIX/Linux path separator /. If you use the Windows path separator \ it becomes part of the name. See also Path Segment and File Name Limitations.

HTTP Response Codes and Request Status Codes

The response body always returns a JSON object with status information. (See Sample Returned JSON Object).

On success the call returns an HTTP 200 status code, `0` in the `X-Agile-Status` header, and a success message in the JSON object.

On error, the HTTP Status Code is an error code, and the error is reflected in the `X-Agile-Status` header and the JSON object. The following table provides details about response values.

HTTP Status Code	Description	X-Agile-Status / JSON Object Values
200	Directory created successfully or directory already exists	**0:** success
400	Bad request	**-2**: path exists: Parent directory contains a file with the same name as the directory you attempted to create. **-3**: no parent directory. One or more leading paths in the value you passed to `X-Agile-Directory` do not exist and you set the `X-Agile-Recursive` header to `false`. **-8**: invalid path given (path too long) **-39**: invalid `X-Agile-Recursive` value
401	Not authorized. You did not include the X-Agile-Authorization header in your request.	**-10001**: no token
403	Invalid authentication credentials	**-10001**: invalid token
500	Internal server error	**-5**: service unavailable

Response Headers

Header Name	Type	Description
X-Agile-Status	int	Contains response codes. (See HTTP Response Codes and Request Status Codes.)

Sample Returned JSON Object

The object is returned in the response body and contains two key-value pairs: `message` and `code` as in this example:

```
{"message": "success", "code": 0}
```

curl Sample Request 1

None of the path segments in `/APAC/a/b` exist. Create them by explicitly setting the `X-Agile-Recursive` header to `true`:

```
curl -v -k /
```

```
-H "Content-Length: 0"/
-H "Content-Type: text/json"/
-H "X-Agile-Authorization: 3b947a4c-1e29-4065kl-ad6b-8a6a024a8u33"/
-H "X-Agile-Recursive: true"/
-H "X-Agile-Directory: /APAC/a/b"/
http://{Account name}.upload.llnw.net/post/directory
```

Sample Success Response 1

```
HTTP/1.1 200 OK
Date: Tue, 09 Jun 2015 14:13:05 GMT
Server Apache is not deny listed
Server: Apache
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: X-Agile-Authorization, X-Content-Type
Access-Control-Allow-Methods: OPTIONS
X-Agile-Status: 0
Content-Length: 33
Content-Type: text/json;charset=utf-8
{"message": "success", "code": 0}
```

curl Sample Request 2

Make sure you don't accidentally create an undesired parent directory. In this example, /`future-releases` does not exist:

```
curl -v -k ^
-H "X-Agile-Authorization: 5e26d9b5-09a3-40ba-b3bd-0bd8a740092c"^
-H "X-Agile-Directory: /future-releases/hot"^
-H "X-Agile-Recursive: false"^
http://{Account name}.upload.llnw.net/post/directory
```

curl Sample Request 2 Response

```
HTTP/1.1 400 parent directory does not exist
Date: Wed, 02 Sep 2015 14:05:05 GMT
Server Apache is not deny listed
Server: Apache
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: X-Agile-Authorization, X-Content-Type
Access-Control-Allow-Methods: OPTIONS
X-Agile-Status: -3
Content-Length: 58
Connection: close
Content-Type: text/json;charset=utf-8
{"message": "parent directory does not exist", "code": -3}
```

Path Segment and File Name Limitations

Origin Storage path segment names and file names can contain a maximum of 255 bytes. The maximum path length (path + file name) is 4096 bytes.

For request headers that support UTF-8 encoding, Unicode characters are encoded using byte sequences of different lengths:

- Basic Latin letters, digits, and punctuation signs use one byte.
- Most European and middle east script letters fit into a 2-byte sequence: extended Latin letters (with tilde, macron, acute, grave and other accents), Cyrillic, Greek, Armenian, Hebrew, Arabic, Syriac, and others.
- Korean, Chinese, and Japanese ideographs use 3-byte or 4-byte sequences.

UTF-8 path segments are URL-encoded ascii representations of the unicode characters.

For example, the Chinese character 今 is three bytes: '\xe4\xbb\x8a'

When URL-encoded , the character is: '%E4%BB%8A' three bytes each for a total of 9 bytes, so this single character would actually take up 9 bytes of the 255 byte limit. See Byte Calculation for additional details.

Byte Calculation

Because UTF-8 is supported, it is wise to first ensure that a path segment or file name does not exceed the limitation before creating.

Using Chinese characters (3 bytes per character) as an example, here is how to determine the number of bytes in a file name in Python:

```
>>> # File name with 3 characters before the file extension.
>>> filename = "今日は.txt"
>>> utf8_chars = "今日は"
>>> ascii_chars = ".txt"
>>> ascii_chars
'.txt'
>>> len(utf8_chars)
9
>>> len(ascii_chars)
4
>>> ( len(utf8_chars) * 3 ) + len(ascii_chars)
31
```

Working with Files

The *Origin Storage* API lets you work with existing files, performing actions such as copying, moving, and listing files. These operations are available in the JSON-RPC interface only. See the following for details:

- Working with Files in the JSON-RPC Interface
- Working With Methods Common to Files and Directories.

The *Origin Storage* API also lets you upload files. See the following for details:

- Uploading Files – Multipart
- Uploading Files – Non-Multipart

Note that you can download files from *Origin Storage*, although this capability is outside of the JSON-RPC and HTTP interfaces. For details, see Download a File.

Working with Files in the JSON-RPC Interface

You can perform file operations explained in the following sections:

- Copy a File
- Delete a File
- Move a File
- List Files
- Set a File's Content Type
- Generate a MediaVaultURL

Copy a File

Method name: `fetchFileHTTP`

Copies a file from one location to a different location. If desired, you can specify a new name for the destination file. The method works according to the following rules:

- You cannot use wild cards.
- The copy operation is performed asynchronously.

The method asynchronously submits the copy request to a queue where it waits with other requests.

Copy a File Using Named Parameters

```
{
  "method": "fetchFileHTTP",
  "id": 4,
  "params": {
    "token": "e1254148-c557-4bdd-99ea-ffe76cd450ad",
    "path": "/TreesOfEurope/DeciduousTrees.txt",
    "uri": "http://{Account name}/
            TreesOfSouthAmerica/DeciduousTrees.txt",
    "flags": 1
  },
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Para-meter Name	Typ-e	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
path	str	The destination path and file name
uri	str	The URI for the source path and file name. Must use properly URL-encoded values for dis-allowed characters or an error is returned. **Note:** The root path of your account is http://{Account name}/ Example: If you have a `preview.mp4` file in a `previews` directory off the root path, you reference the file like this: `http://{Account name}/previews/preview.mp4`
flags	int	Optional Additional processing instructions. Valid values: **Flag** / **Description** FF_CREATE_DESTDIR: Automatically create the destination directory if it does not exist 1: Equivalent to FF_CREATE_DESTDIR

Return Codes

- **0**: success
- **-4**: queue failure
- **-13**: invalid flags
- **-14**: invalid source `uri`
- **-56**: too busy - concurrency reached (wait and try again)
- **-10001**: invalid token

Note:
For a list of error codes not specific to `fetchFileHTTP`, see Global Error Codes.

Response Data

Returns only the codes discussed in Return Codes. Does not return any data structures.

Python Sample Requests

Using named parameters, copy `example4.jpg` in the root directory and rename the copy to `example3.jpg`:

```
>>> api.fetchFileHTTP(token, '/example3.jpg', 'http://{Account name}/example4.jpg',
1)
0
```

Delete a File

Method name: `deleteFile`

Deletes a file from the specified location. If the file is successfully deleted, the system sets the parent directory's mtime (last modified time) to the current system time.

Delete a File Using Named Parameters

```
{
  "method": "deleteFile",
  "id": 5,
  "params": {
    "token": "d6835f68-7bd4-4a29-9c40-54fbaeeb2444",
    "path": "/TreesOfEurope/DeciduousTrees.txt"
  },
  "jsonrpc": "2.0"
}
```

Delete a File Using Positional Parameters

Positional parameters must be applied in the same order shown in the named parameters sample. Example:

```
{
  "method": "deleteFile",
  "id": 4,
  "params": [
    "1ee13c07-1f50-4c05-8f89-d5779e17332b",
    "/TreesOfEurope/DeciduousTrees.txt"
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
path	str	Path and file name to delete. All attempts to delete multiple files through the use of wildcards fail with a return code of -1 (file does not exist).

Return Codes

- **0**: success
- **-1**: file does not exist
- **-10001**: invalid token

Note:
For a list of error codes not specific to `deleteFile`, see Global Error Codes.

Response Data

Returns only the codes discussed in Return Codes. Does not return any data structures.

Python Sample Requests

Delete the example1.jpg from the root directory:

```
>>> api.deleteFile(token, '/example1.jpg')
0
```

Move a File

Although the *Origin Storage* API does not provide a specific method for moving files, you can use the `rename` method to move a file, specifying a different directory for the `newpath` parameter.

Python Sample Requests

Move `example.txt` from the `/e` directory to the `/d` directory:

```
>>> api.rename(token, '/e/example.txt', '/d/example.txt')
0
```

For additional information, see Rename a File or Directory.

List Files

Method name: `listFile`

Lists files that are direct children of a given path. The method is not recursive and operates only on the given path.

List Files Using Named Parameters

```
{
  "method": "listFile",
  "id": 1,
  "params": {
    "token": "70f2c2c9-42fc-4021-913a-6487217e19b4",
    "path": "/",
    "pageSize": 1,
    "cookie": 0,
    "stat": true
  },
  "jsonrpc": "2.0"
}
```

List Files Using Positional Parameters

Positional parameters must be applied in the same order shown in the named parameters sample. Example:

```
{
  "method": "listFile",
  "id": 0,
  "params": [
    "ad80c27a-f18c-42a2-9197-feea7631ab3b",
    "/",
    1,
    0,
    true
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
path	str	Directory to list
pageSize	int	Optional The number of files to return from a call to the method. Must be less than or equal to 10000. Defaults to 100
cookie	uint64	Optional On return, indicates whether additional files can be returned with subsequent calls to the function. Omit this argument or pass 0 the first time you call the function. For all subsequent calls, pass the cookie value returned from the prior call. Defaults to 0
stat	bool	Optional Include file details in output.

Parameter Name	Type	Description
		Defaults to `False`

Return Codes

- **0**: success
- **-1**: invalid path
- **-11**: invalid cookie
- **-12**: invalid page size
- **-10001**: invalid token

Note:
For a list of error codes not specific to `listFile`, see Global Error Codes.

Response Data

When the `stat` parameter is set to `False`, the method returns an object with the following data:

- **code**: (int) return code
- **cookie**: (uint64) indicates whether additional files can be returned with subsequent calls to the function. Non-zero value: additional files are available. Zero value: no more files are available.
- **list**: array of file objects. Data for each object is:
  - **name**: (str) file name
  - **type**: (int). Always `2` (file)

Note: The order in which files are returned is non-deterministic. Results are not sorted.

Example:

```
{
  "code": 0,
  "cookie": 2,
  "list": [
    {
      "name": "flat-irons.jpg",
      "type": 2
    },
    {
      "name": "farm-cat.jpg",
      "type": 2
    }
  ]
}
```

When the `stat` parameter is set to `True`, each file listed includes a `stat` object with the following data:

- **checksum**: (str) file's SHA-256 hexidecimal digest
- **ctime**: (int) creation time in seconds since epoch
- **gid**: (int) group ID of user that created the file
- **mimetype**: (str) file's MIME type. Example: text/plain

- **mtime**: (int) last modification time in seconds since epoch
- **size**: (int) file size
- **uid**: (int) ID of user that created the file

Example:

```
{
  "code": 0,
  "cookie": 1,
  "list": [
    {
      "stat": {
        "uid": 12020,
        "gid": 100,
        "size": 659819,
        "checksum":
"24a436b8a87462482806667b76ec5a120ae236a721c311a66235ad3cf4073bd4",
        "ctime": 1423676679,
        "mimetype": "image/jpeg",
        "mtime": 1423676671
      },
      "name": "flat-irons.jpg",
      "type": 2
    }
  ]
}
```

Python Sample Requests

Loop through all files in a directory in chunks of three and print the file names.

```
# User-defined variables
dir = '/'
pageSize = 3

def printFiles( fileList ):
  global hasFiles
  for ( item ) in fileList:
    hasFiles = True
    print item['name']

hasFiles = False
results = api.listFile( token, dir, pageSize, False )

print ('Files in directory "' + dir + '":')
printFiles( results['list'] )
cookie = results['cookie']

while ( cookie > 0 ):
  results = api.listFile( token, dir, pageSize, cookie, False )
  printFiles( results['list'] )
  cookie = results['cookie']

if not hasFiles:
  print ('**No files in directory**')
```

Set a File's Content Type

Method name: `setContentType`

Sets a file's content type to one of the standard MIME types.

Note the following:

- If you attempting set content type on a directory, the call seems to succeed (returns 0) but in reality has no effect.
- To use `setContentType` on an object, you must belong to the group that owns the object. To determine the group that owns the object, issue a `stat` call on the object, setting the `detail` parameter to `True`. Then look at the `gid` value from the results of the call. (See Obtain File or Directory Metadata for information about the `stat` function.)To determine your group, you must log in using the `login` function, setting the `detail` parameter to `True`. Then look at the `gid` value from the results of your call to `login`. If the two `gid` values match, you will be able to set the file's content type.

Set Content Type Using Named Parameters

```
{
  "method": "setContentType",
  "id": 1,
  "params": {
    "token": "12086a82-eed2-4e83-a25b-514a55416b5b",
    "path": "/a/hi1.txt",
    "content_type": "text/plain"
  },
  "jsonrpc": "2.0"
}
```

Set Content Type Using Positional Parameters

Positional parameters must be applied in the same order shown in the named parameters sample. Example:

```
{
  "method": "setContentType",
  "id": 0,
  "params": [
    "8c4b1dfc-d36d-4070-8221-f969bc040110",
    "/a/hi1.txt",
    "text/plain"
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
path	str	File whose content type you want to set.
content_type	str	Content type such as `text/plain`. See Content Types.

Return Codes

- **0**: success
- **-1**: file does not exist or invalid object type
- **-3**: parent path does not exist
- **-6**: permission denied. Caller's uid does not exist or does not have permissions to `path`.
- **-8**: invalid path
- **-33**: invalid content type
- **-10001**: invalid token

Note:
For a list of error codes not specific to `setContentType`, see Global Error Codes.

Response Data

Returns only the codes discussed in Return Codes. Does not return any data structures.

Python Sample Requests

Set content type on cloud-storage-icon.png:

```
>>> api.setContentType(token, '/dir10/cloud-storage-icon.png', 'image/png')
0
```

Generate a MediaVault URL

Method name: `mediaVaultUrl`

Generates time-limited URLs you can use to download or preview a file. The time limit is enforced through an expiry time. All requests for the file using the URLs are refused when the expiry time has passed. The expiry time relates to the start of the HTTP GET request, so as long as the request is initiated before this time the download will be successful.

Note: For additional information about MediaVault, please see the MediaVault User Guide.

Generate a MediaVault URL Using Named Parameters

```
{
  "method": "mediaVaultUrl",
  "id": 1,
  "params": {
    "token": "d6835f68-7bd4-4a29-9c40-54fbaeeb2444",
    "path": "/TreesOfEurope/DeciduousTrees.txt",
    "expiry": 3600
  },
  "jsonrpc": "2.0"
}
```

Generate a MediaVault URL Using Positional Parameters

Positional parameters must be applied in the same order shown in the named parameters sample. Example:

```
{
  "method": "mediaVaultUrl",
  "id": 1,
  "params": [
    "1ee13c07-1f50-4c05-8f89-d5779e17332b",
    "/TreesOfEurope/DeciduousTrees.txt",
    3600
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
path	str	File to generate MediaVault URL. The path does not have to exist, this is to support generating a URL prior to uploading an object.
expiry	int	Optional Download URL expiry for object in seconds. Must be in the range `1` to `2147483648`. If expiry is 0 or omitted, it defaults to `3600`.

Return Codes

- **0**: success
- **-1**: internal error
- **-2**: path exists and is a directory
- **-8**: invalid path
- **-34**: invalid expiry
- **-60**: service is disabled or unavailable
- **-10001**: invalid token

Response Data

The method returns an object with the following data:

- **code**: (int) return code
- **download_url**:(str) URL to use for downloading the file
- **message**: description of **code**
- **preview_url**: (str) URL to use for previewing the file

Example:

```
{
  "code": 0,
  "download_url": "http://cs-download.limelight.com/<path to file>",
  "message": "success",
  "preview_url": "http://cs-download.limelight.com/<path to file>",
}
```

curl Sample Request

Generate MediaVault URLs for the example3.jpg file. This code first calls the ‘login’ method and extracts the token from the output. Next, the code issues the call for the MediaVault URL, passing the extracted token.

```
token=$(curl -v -X POST -H 'X-Agile-Username: yourUser' -H 'X-Agile-Password:
yourPassword' 'https://[shortname]-l.upload.llnw.net/account/login' 2>&1 | grep X-
Agile-Token: | awk -F ': ' '{print $2}' | tr -d '\r')
curl -v -k^

curl -d '{"jsonrpc": "2.0", "method": "mediaVaultUrl", "params": ["'$token'",
"/example3.jpg", 300], "id": 1}' 'https://[shortname]-l.upload.llnw.net/jsonrpc'
```

Working With Methods Common to Files and Directories

The *Origin Storage* API provides methods that apply to both files and directories, such as renaming a file or directory. APIs for working with existing files are available in the JSON-RPC interface only.

Working with Methods Common to Files and Directories in the JSON-RPC Interface

The JSON-RPC interface provides methods common to files and directories as explained in the following sections:

- List Files and Directories
- Obtain File or Directory Metadata
- Rename a File or Directory
- Change a File or Directory Last Modification Time

List Files and Directories

Method name: `listPath`

Lists both files and directories that are direct children of a given path. The method is not recursive and operates only on the given path.

Named Parameters Example

```
{
  "method": "listPath",
  "id": 1,
  "params": {
    "token": "f03efb6a-c63c-4046-9bd9-d1cd8c36c10b",
    "path": "/a",
    "pageSize": 1,
    "cookie": "",
    "stat": true
  },
  "jsonrpc": "2.0"
}
```

Positional Parameters Example

Positional parameters must be applied in the same order shown in the named parameters example.

```
{
  "method": "listPath",
  "id": 0,
  "params": [
    "3984d225-286e-41b6-811a-99fcea4dd411",
    "/a",
    1,
        "",
        true
    ],
    "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
path	str	Directory for which you want to list files and directories.
pageSize	int	Optional The number of files and directories to return from a call to the method. Must be less than or equal to 10000. Defaults to 100.
cookie	str	On return, indicates whether additional files or directories can be returned with subsequent calls to the function. Pass an empty string the first time you call the function. For all subsequent calls, pass the cookie value returned from the prior call. After you have returned all files and directories, the returned cookie value is null. Defaults null
stat	bool	Optional Indicates whether to include file details in output. Defaults to `False`

Return Codes

- **0**: success
- **-1**: path does not exist
- **-8**: malformed path
- **-11**: invalid cookie parameter
- **-12**: invalid page size parameter
- **-10001**: invalid token

Note:
For a list of error codes not specific to `listPath`, see Global Error Codes.

Response Data

On success returns an object with the following data:

- **code**: (int) return code. See Return Codes for specific values.
- **cookie**: (str) base64-encoded string indicating whether additional files or directories can be returned with subsequent calls to the function. Non-null value: additional files or directories are available. Null value: no more files or directories are available.
- **dirs**: an array of directory objects returned. The exact attributes in each object depends on whether you pass `True` or `False` for the `stat` parameter.
- **files**: an array of directory objects returned. The exact attributes in each object depends on whether you pass `True` or `False` for the `stat` parameter.

When the `stat` parameter is set to `False`, the method returns the following data for each directory or file in the `dirs` and `files` lists respectively:

- **name**: directory or file name

Example:

```
{
  "code": 0,
  "cookie": "AAAAAAAAAAEAAAAAAAAAAQ\u003d\u003d",
  "dirs": [
    {
      "name": "steve"
    }
  ],
  "files": [
    {
      "name": "flat-irons.jpg"
    }
  ]
}
```

When the `stat` parameter is set to `True`, the method returns the following additional data:

- **checksum**: (str) file's SHA-256 hexidecimal digest (files only)
- **ctime**: (int) creation time in seconds since epoch
- **gid**: (int) group ID of user that created the file or directory
- **mimetype**: (str) file's MIME type. Example: text/plain (files only)
- **mtime**: (int) last modification time in seconds since epoch
- **size**: (int) file size in bytes (files only)
- **uid**: (int) ID of user that created the file or directory

Example:

```
{
  "code": 0,
  "cookie": "AAAAAAAAAAEAAAAAAAAAAQ\u003d\u003d",
  "dirs": [
    {
      "uid": 12020,
      "gid": 100,
      "name": "steve",
      "ctime": 1423258489,
      "mtime": 1423258489
    }
  ],
  "files": [
    {
      "uid": 12020,
      "gid": 100,
      "size": 659819,
      "name": "flat-irons.jpg",
      "checksum":
"24a436b8a87462482806667b76ec5a120ae236a721c311a66235ad3cf4073bd4",
      "ctime": 1423676679,
      "mimetype": "image/jpeg",
      "mtime": 1423676671
    }
  ]
}
```

When you call the method on an empty directory, or after you have returned all files and directories and you call the method again, it returns the following:

```
{
  "code": 0,
  "cookie": null,
  "dirs": [],
  "files": []
}
```

When you call the method on a directory with a file but no directory, the method returns the following:

```
{
  "code": 0,
  "cookie": "AAAAAAAAAAAAAAAAAAAAAQ==",
  "dirs": [],
  "files": [
    {
      "name": "text-file.txt"
    }
  ]
}
```

Similarly, when you call the method on a directory with a sub-directory but no files, the method returns the following:

```
{
  "code": 0,
  "cookie": "AAAAAAAAAAAAAAAAAAAAAQ==",
  "dirs": [
    {
      "name": "sub-dir"
    }
  "files": [],
  ]
}
```

On failure returns an object like the following, with `code` set to the appropriate value described in Return Codes:

```
{
  "code": -1
}
```

Python Sample Requests

Obtain and display formatted information about a directory specified by the variable `dirName`. Details include stat data.

```
import jsonrpclib, time

class StatError(RuntimeError):
  def __init__(self, arg):
    self.args = arg

class ListPathError(RuntimeError):
  def __init__(self, arg):
    self.args = arg

# User-Defined Variables
dirName = '/'
pageSize = 30

#Variables maintained by the application.
cookie = ''
numFiles = 0
numDirs = 0
totalBytes = 0

try:

  ##################################################
  # Retrieve directories and files in the directory.
  ##################################################
  while cookie is not None:
    res = api.listPath(token, dirName, pageSize, cookie, True)
    code = res['code']
    if code !=0:
      msg = 'Error issuing listPath command on directory: ' + dirName
      msg += '\n  Return code: ' + str( code )
      msg += '\n  See API documentation for details.'
      raise ListPathError( msg )

    numDirs  += len(res['dirs'])
    numFiles += len(res['files'])
    files = res['files']
    for f in files:
      totalBytes += f['size']
    cookie = res['cookie']

  ######################################
  # Get the directory's ctime and mtime.
  ######################################
  myStat = api.stat(token, dirName, True)
  code = myStat['code']
  if code != 0:
    msg = 'Error issuing stat command on directory: ' + dirName
    msg += '\nReturn code: ' + str( code ) + '\nSee API documentation for details.'
    raise StatError( msg )

  creationTime = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime( myStat['ctime']
))
  modificationTime = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime( myStat
['mtime'] ))

  ################################
  # Display directory information.
  ################################
  hdrSep    = '================================================='
  detailSep = '-------------------------------------------------'

  hdr = 'Information for Directory "' + dirName + '"'

  thisHdrSep = hdrSep
  if len( hdr ) > 49:
    while len( thisHdrSep ) < len( hdr ):
      thisHdrSep += thisHdrSep

  thisHdrSep = thisHdrSep[0: len(hdr)]

  print (thisHdrSep)
  print (hdr)
  print (thisHdrSep)
  print ('Total Directories:            ' + str( numDirs )
  print (detailSep
  print ('Total Files:                  ' + str( numFiles )
  print (detailSep
  print ('Total Size All Files (bytes): ' + str( totalBytes )
  print (detailSep
  print ('Directory Creation time:      ' + creationTime)
  print (detailSep)
  print ('Directory Modification time:  ' + modificationTime)
  print (detailSep)

except ListPathError,e:
  print (''.join( e.args ) )

except StatError,e:
  print (''.join( e.args ) )

finally:
  print ('\nLogging out...\n')
  res = api.logout(token)
  print ('Logout results: ' + str( res ) )
```

If you are interested in seeing the values in a cookie, use code like the following:

```
>>> import base64, struct
>>> cookie = api.listPath(token, '/', 2, None, True)['cookie']
>>> (dcookie, fcookie) = struct.unpack(">QQ", base64.b64decode(cookie))
>>> # View number of directories returned
>>> dcookie
2
>>> # View number of files returned
>>> fcookie
2
```

Obtain File or Directory Metadata

Method name: `stat`

Obtains meta data such as creation time and last modified time for a specified file or directory.

Named Parameters Example

```
{
  "method": "stat",
  "id": 1,
  "params": {
    "token": "46c939fa-7ab7-4ea6-b305-494d94c66e3a",
    "path": "/a/hi1.txt",
    "detail": true
  },
  "jsonrpc": "2.0"
}
```

Positional Parameters Example

Positional parameters must be applied in the same order shown in the named parameters example.

```
{
  "method": "stat",
  "id": 1,
  "params": [
    "46c939fa-7ab7-4ea6-b305-494d94c66e3a",
    "/a/hi1.txt",
    true
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
path	str	Directory or file for which to provide details.
detail	bool	Optional Indicates whether to Include file details in output. Defaults to `False`

Return Codes

- **0**: success
- **-1**: directory or file does not exist
- **-10001**: invalid token

Note:
For a list of error codes not specific to `stat`, see Global Error Codes.

Response Data

When the `detail` parameter is set to `False`, the method returns an object with the following data:

- **code**: (int) return code
- **ctime**: (int) creation time in seconds since epoch
- **mtime**: (int) last modification time in seconds since epoch
- **size**: (int) file size in bytes (files only)
- **type**: (int) identifies the path type: 1 = directory, 2 = file

Example:

```
{
  "code": 0,
  "size": 698666,
  "ctime": 1434579443,
  "mtime": 1450392160,
  "type": 2
}
```

When the `detail` parameter is set to `True`, the following additional data is returned:

- **checksum**: (str) file's SHA-256 hexadecimal digest. Empty string for directories
- **gid**: (int) group ID of user that created the directory
- **mimetype**: (str) file's MIME type (files only)
- **uid**: (int) ID of user that created the file or directory

Example:

```
{
  "uid": 12020,
  "code": 0,
  "gid": 100,
  "size": 698666,
  "checksum": "46cb29d05b447367e343143450ae1ca8cbe0080af225a8310c832be4ba64a096",
  "ctime": 1434579443,
  "mimetype": "image/jpeg",
  "mtime": 1450392160,
  "type": 2
}
```

On failure returns the following object with `code` set to the appropriate value described in Return Codes:

```
{
  "uid": 0,
  "code": -1,
  "gid": 0,
  "checksum": ""
}
```

Python Sample Requests

View meta data for a directory, including details. Note the empty string value for `checksum`.

```
api.stat(token, '/', True)
```

```
{
  "uid": 12020,
  "code": 0,
  "gid": 100,
  "checksum": "",
  "ctime": 1395777680,
  "mtime": 1395777680,
  "type": 1
}
```

View meta data for the /example2.jpg file, including details:

```
api.stat(token, '/example2.jpg', True)
```

```
{
  "uid": 12020,
  "code": 0,
  "gid": 100,
  "size": 4662,
  "checksum": "71ad472cbd384ef2a8a5f6f7bcc4d538fe22db162ed087548cda736eef9eb720",
  "ctime": 1433972816,
  "mimetype": "application/octet-stream",
  "mtime": 1431079057,
  "type": 2
}
```

Rename a File or Directory

Method name: `rename`

Renames a file or directory. Note that you can use this method to move a file by specifying a different parent path in the `newPath` argument. Similarly, you can move a directory (if it is empty) to another parent directory.

When you rename a file or directory such that it gets moved to a new directory, the system sets the mtime (last modified time) of the old and new parent directories to the current system time.

When you rename a file or directory such that it stays in its current directory, the system sets the directory's mtime to the current system time.

Although Limelight Networks provides APIs for deleting and renaming a directory, you can do so only if the directory contains no files or sub-directories. This is because the required background processes, such as re-applying policies and pushing maps out to Bricks, could potentially have a significant impact on the *Origin Storage* platform, causing latency in other operations.

If you are having trouble deleting or managing content with the existing supported management tools or via the API, please contact your account manager who will engage the Edgio Advanced Services team to work with you to achieve the desired results (additional charges may apply).

Named Parameters Example

```
{
  "method": "rename",
  "id": 3,
  "params": {
    "token": "6a1a559c-ae50-4f95-b270-b2dc8519b8fd",
    "oldpath": "/OldName.txt",
    "newpath": "/NewName.txt"
  },
  "jsonrpc": "2.0"
}
```

Positional Parameters Example

Positional parameters must be applied in the same order shown in the named parameters example.

```
{
  "method": "rename",
  "id": 2,
  "params": [
    "cf725651-cfd0-43a5-8c52-5e113fe61bd2",
    "/OldName.txt",
    "/NewName.txt"
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
oldpath	str	Existing file or directory name, including full path
newpath	str	New file or directory name, including full path

Note the following:

- When renaming a file you must include the full path in the argument to `newpath`. Arguments to `newpath` that do not include a full path will be created in the root directory.
- You cannot rename a directory that has files or sub-directories.
- You cannot rename a file or directory to an existing name in an attempt to over-write the directory or file.

Return Codes

- **0**: success
- **-1**: oldpath not found or attempt to rename a file or directory to its current name
- **-2**: newpath already exists
- **-3**: newpath parent directory does not exist
- **-6**: permission denied. Caller's uid does not exist or does not have permissions to `oldpath`.
- **-7**: operation not supported. Directory contains files or directories.
- **-10001**: invalid token

Note:
For a list of error codes not specific to `rename`, see Global Error Codes.

Response Data

Returns only the codes discussed in Return Codes. Does not return any data structures.

Python Sample Requests

Rename the `auto1.jpg` file to `auto3.jpg`:

```
>>> api.rename(token, '/auto1.jpg', '/auto3.jpg')
0
```

Rename the `floral-items` directory to `floral-accessories`:

```
>>> api.rename(token, '/floral-items', '/floral-accessories')
0
```

Change a File or Directory Last Modification Time

Method name: `setMTime`

Changes a file's or directory's last modification time.

Named Parameters Example

```
{
  "method": "setMTime",
  "id": 1,
  "params": {
    "token": "d7320fa4-fd5b-4a4e-a1f9-187143f72d90",
    "path": "/a/hi1.txt",
    "mtime": 1461885068
  },
  "jsonrpc": "2.0"
}
```

Positional Parameters Example

Positional parameters must be applied in the same order shown in the named parameters example.

```
{
  "method": "setMTime",
  "id": 0,
  "params": [
    "bca3af0b-f417-4131-b28d-3ca4e35fa144",
    "/a/hi1.txt",
    1461942652
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
path	str	File or directory's whose last modification time you want to change
mtime	int	New modification time in seconds since epoch

Return Codes

- **0**: success
- **-1**: file/directory does not exist or you passed an invalid object type
- **-5**: cannot acquire lock
- **-8**: invalid path
- **-27**: invalid mtime
- **-10001**: invalid token

Note:
For a list of error codes not specific to `setMTime`, see Global Error Codes.

Response Data

Returns only the codes discussed in Return Codes. Does not return any data structures.

Python Sample Requests

Change a file's modification time:

```
>>> api.setMTime(token, '/auto.jpg', 1450392160)
0
```

Change a directory's modification time:

```
>>> api.setMTime(token, '/picture', 1450392160)
0
```

Uploading Files – Non-Multipart

APIs for non-multipart file uploads are available in the HTTP interface only. See Uploading Files in the HTTP Interface.

Uploading Files in the HTTP Interface

Origin Storage provides two techniques for non-multipart upload. Use the following table to decide which technique is best for you.

Technique	Description	End Point	Where to Find Information
Web Browser Upload	Does an HTTP POST upload using multipart form-data. Use this if you want the ability to upload a file from a web page. **Note:** This technique is relatively slow because the checksum is calculated after the file is at rest. (The API has to open and read the file to calculate the checksum.)	/post/file	Web Browser Upload
File Raw Post	Uploads a file as a stream. The only payload is the file data.	/post/raw	File Raw Post

Note:
For files larger than 10GB, use the multipart upload functionality. See Uploading Files – Multipart for information.

Web Browser Upload

```
POST to /post/file
```

Does an HTTP POST upload using multipart form-data. Slower than File Raw Post.

When the file creation is complete, the system sets the parent directory's mtime (last modification time) to the current system time.

Note:
An end-to-end /post/file upload example is available. See /post/file Example .

Request Headers

Header Name	Type	Description
X-Agile-Authorization	str	Valid token from a call to `login` (JSON-RPC interface) or

Header Name	Type	Description
		`/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively. As an alternative, you can instead pass the token as a query string on the URL. See curl Sample Request 2.
X-Agile-Content-Type	str	Optional Lets the API look up or detect content type for the file to upload. If this header is not present or the content type provided in it is invalid, the content type will be derived from the `Content-Type` field of the MIME-encoded attachment. If a suitable mime type is not detected from the mime attachment, the mime type will be derived from the file name's extension. Must be a valid *Origin Storage* MIME type (see Content Types). See also Content Type Handling.
X-Agile-Content-Detect	str	Optional Instructs the API to detect the content type. The API will ignore this header if you use the `X-Agile-Content-Type` header. Pass one of: **name**: lookup MIME type by using the filename, usually the extension **content**: lookup MIME type by scanning the contents of the file **auto**: first lookup by name, then by content if no name matches See also Content Type Handling.
X-Agile-Checksum	str	Optional The hexlified version of the bytes that make up the file's sha256 checksum. Origin Storage will calculate the file's checksum and compare it with the value of `X-Agile-Checksum`. (When a file first arrives at a landing pad, it is placed in a temporary area called a 'scratch space'. It is at this point that the checksum comparison occurs.) If `X-Agile-Checksum` is provided and it matches the calculated checksum, the file will be stored and replicated. If `X-Agile-Checksum` is provided and it does not match the calculated checksum, the file will not be ingested. Instead, error code -26 (invalid checksum) will be returned and the file will be removed from landing pad's scratch space and will not be ingested.
X-Agile-Recursive	str	Optional When uploading a file to the location specified by the `directory` form field, this header instructs the API to create any leading directories if they do not already exist. If you set the value to `true`, non-existing leading directories will be

Header Name	Type	Description
		created. If you omit the `X-Agile-Recursive` header or set it to `false`, then non-existing leading directories will not be created. If you use both this header and the `recursive` form field and the two are different, the `X-Agile-Recursive` header will take precedence. Defaults to `false`.

Form Fields

Field Name	Type	Description
uploadFile	str	Path and name of file to upload. See also About the uploadFile Form Field.
basename	str	Optional Name of file as it will be stored in *Origin Storage*, minus the path. Forward slashes `/` are not allowed. See also How Uploaded File Paths Are Determined
directory	str	Optional Directory in *Origin Storage* where the file will be stored Defaults to the root directory See also How Uploaded File Paths Are Determined.
recursive	str	Optional When uploading a file to the location specified by the `directory` form field, this field instructs the API to create any leading directories if they do not already exist. If you set the value to `true`, non-existing leading directories will be created. If you omit the field or set it to `false`, then non-existing leading directories will not be created. If you use both this field and `X-Agile-Recursive` header and the two are different, the `X-Agile-Recursive` header will take precedence. Defaults to `false`.
return_url	str	Optional Provides the ability to set a “custom” url to redirect to. If you include this field you don’t have to include the `return_referer` field.
return_referer	int	Optional Provides ability to redirect to the location in the `Referer`[1] header (if it is present). In this case, *Origin Storage* sends a 302 redirect to the `Referer` header location.

Field Name	Type	Description
		If you want to redirect to a custom location, omit the `return_referer` field and include the custom location in the `return_url` field. [1]The `Referer` header is sent by web browsers automatically and identifies the address of the webpage that linked to the resource being requested.
expose_egress	str	Optional How to expose egress. Valid values: **PARTIAL**: expose egress immediately (partially uploaded content can be served). **COMPLETE**: (default) expose egress when the file completes uploading. **POLICY**: expose when the file is in policy. Defaults to **COMPLETE**
mtime	int	Optional File modification time in seconds since epoch. Defaults to 0 – current timestamp.

How Uploaded File Paths Are Determined

The complete path of the uploaded object will be the `directory` field value + `/` + `basename` field value. If the `basename` value has any forward slashes in it, the last path component will be used and the leading paths will be stripped. For instance, if you pass `/test` for `directory` and `/1983/img001.jpg` for `basename`, the final path will be `/test/img001.jpg` (the 1983 is stripped).

About the uploadFile Form Field

`uploadFile` is a multipart/form-data encoded payload as in this example:

```
------------------------------44f9af2ecd0c
Content-Disposition: form-data; name="uploadFile"; filename="test.txt"
Content-Type: text/plain

Sample data
------------------------------44f9af2ecd0c--
```

HTTP Response Codes and Request Status Codes

On success the call returns an HTTP 200 status code, and `0` in the `X-Agile-Status` header.

On error, the HTTP Status Code is an error code, and the error is reflected in the `X-Agile-Status` header. The following table provides details about response values.

HTTP Status Code	Description	Possible X-Agile-Status Values
200	Success	**0**: success
400	Bad request, missing or invalid parameters	**-3**: no parent directory. One or more path segments in the `directory` form field do not exist and you did not provide a value for the `recursive` form field or the `X-Agile-Recursive` request header. **-8**: invalid path (path too long) in the `directory` form field **-21**: invalid value in the `expose_egress` form field **-23**: empty file **-24**: missing upload file in HTTP POST (You did not supply a file to upload.) **-25**: upload not supported; you attempted to upload more than one file. **-26**: invalid checksum **-27**: invalid value in the `mtime` form field **-33**: invalid value in the `X-Content-Type` header **-48**: invalid value in the `X-Content-Detect` header
401	unauthorized, missing `X-Agile-Authorization` header	**-10001**: missing `X-Agile-Authorization` header
403	Invalid authentication credentials (invalid token)	**-10001**: Invalid token
500	Internal server error	**-5**: Generic error
503	Service unavailable	**-22**: disk full

Response Headers

Header Name	Type	Description
X-Agile-Status	int	Contains response codes. (See HTTP Response Codes and Request Status Codes.)
X-Agile-Path	str	File name and path where the file was uploaded
X-Agile-Size	int	Size in bytes of the file that this call uploaded
X-Agile-Checksum	str	File's SHA-256 hexadecimal digest

curl Sample Request 1

Upload super test file.txt into the root directory. Also pass a basename.

```
curl -v -k\
-H "X-Agile-Authorization: 4f173c74-c929-4f91-9e0e-645a31343258"\
-F "uploadFile=@/super test file.txt"\
-F "basename=super test file.txt"\
-F "directory=/"\
http://{Account name}.upload.llnw.net/post/file
```

Sample 1 Success Response

```
HTTP/1.1 200 OK
Date: Fri, 19 Jun 2015 20:45:01 GMT
Server Apache is not deny listed
Server: Apache
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: X-Agile-Authorization, X-Content-Type
Access-Control-Allow-Methods: OPTIONS
X-Agile-Status: 0
Content-Length: 0
X-Agile-Checksum: d995a1c6b9ba371c2273f209d6659253bf457b3fa047ce6c959ca99
X-Agile-Size: 10
X-Agile-Path: /{Account Name}/super test file.txt
Content-Type: text/json;charset=utf-8
```

curl Sample Request 2

Send the token on the url in a query string. Also pass a basename.

```
curl -v -k\
-F "uploadFile=@/super file2.txt"\
-F "basename=super file2.txt"\
-F "directory=/"\
http://{Account name}.upload.llnw.net/post/file?token=4f174-c929-4f91-9e0e-645a31
```

Sample 2 Success Response

```
HTTP/1.1 200 OK
Date: Wed, 05 Aug 2015 15:43:55 GMT
Server Apache is not deny listed
Server: Apache
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: X-Agile-Authorization, X-Content-Type
Access-Control-Allow-Methods: OPTIONS
X-Agile-Status: 0
Content-Length: 0
X-Agile-Checksum: d995a1c6b9ba371c2273f6659253bf457b3fa047ce62dd8274ac959ca99
X-Agile-Size: 10
X-Agile-Path: /{Account name}/super file2.txt
```

```
Content-Type: text/json;charset=utf-8
```

File Raw Post

```
POST to post/raw
```

Uploads a file as a stream.

Upload speed is faster than Web Browser Upload.

When the file creation is complete, the system sets the parent directory's mtime (last modification time) to the current system time.

Note:
An end-to-end /post/raw upload example is available. See /post/raw Example.

Request Headers

Header Name	Type	Description
X-Agile-Authorization	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
X-Agile-Basename	str	Optional Name of file as it will be stored in *Origin Storage*, minus the path. Forward slashes / not allowed. If you don't provide a value for `X-Agile-Basename`, the API will create a file name in the form of `post-<UUID4 in hex notation>`. Example: `post-ddbb5628fdd3433181ba9771f5709190` (UUID4 is a random Universally Unique ID.) The file name must not include two consecutive periods.
X-Agile-Content-Detect	str	Optional Instructs the API to detect the content type. The API will ignore this header if you use the `X-Agile-Content-Type` header. Pass one of: **name**: lookup MIME type by using the filename, usually the extension **content**: lookup MIME type by scanning the contents of the file **auto**: first lookup by name, then by content if no name matches See also Content Type Handling.
X-Agile-Content-Type	str	Optional Lets the API look up or detect content type for the file to upload. If this

Header Name	Type	Description
		header is not present or the content type provided in it is invalid, the content type will be derived from the `Content-Type` field of the MIME-encoded attachment. If a suitable mime type is not detected from the mime attachment, the mime type will be derived from the file name's extension. Must be a valid *Origin Storage* MIME type (see Content Types). See also Content Type Handling.
X-Agile-Directory	str	Optional Directory in *Origin Storage* where file will be stored. Defaults to the root directory The file name must not include two consecutive periods.
X-Agile-Encoding	str	Optional The default behavior is to treat paths as US ASCII. If the values for `X-Agile-Basename` and `X-Agile-Directory` contain UTF-8 characters, then include this header and set its value to `UTF8`, which will cause the API to treat values as URI quote paths. Note: • You must quote paths according to URL parameter quoting defined in RFC3986 of the Internet Engineering Task Force. • Characters `%20` and + turn into spaces. • Special characters you must handle are: percent sign, dollar sign, ampersand, plus sign, comma, forward slash, colon, semi colon, equal sign, question mark and the @ symbol. See Using UTF-8 Characters in Request Headers.
X-Agile-Expose-Egress	str	Optional When to make the file available for egress. Valid values: **PARTIAL**: expose egress immediately (partially uploaded content can be served) **COMPLETE**: (default) expose egress when the file completes uploading **POLICY**: expose when the file is in policy
X-Agile-MTime	str	Optional File last modification time in seconds since epoch. Defaults to zero (current time)
X-Agile-Recursive	str	Optional. When uploading a file to the location specified by the `X-Agile-Directory` header, this header instructs the API to create any leading paths if they do not already exist. If you set the value to `true`, non-existing leading paths will be created. If you omit the `X-Agile-Recursive` header or set it to `false`, then

Header Name	Type	Description
		non-existing leading paths will not be created and the call will fail. Other values you can pass that are equivalent to `true`: • yes • 1 Other values you can pass that are equivalent to `false`: • no • 0 Defaults to `false`.
X-Agile-Checksum	str	Optional The hexlified version of the bytes that make up the file's sha256 checksum. Origin Storage will calculate the file's checksum and compare it with the value of `X-Agile-Checksum`. (When a file first arrives at a landing pad, it is placed in a temporary area called a 'scratch space'. It is at this point that the checksum comparison occurs.) If `X-Agile-Checksum` is provided and it matches the calculated checksum, the file will be stored and replicated. If `X-Agile-Checksum` is provided and it does not match the calculated checksum, the file will not be ingested. Instead, error code -26 (invalid checksum) will be returned and the file will be removed from landing pad's scratch space and will not be ingested.

Request Payload

The payload consists of the file to upload. For example you could use the following option with curl: `--data-binary @/tmp/example1.jpg`

HTTP Response Codes and Request Status Codes

On success the call returns an HTTP 200 status code and 0 in the `X-Agile-Status` header.

On error, the HTTP Status Code is an error code, and the error is reflected in the `X-Agile-Status` header.

The following table provides details about response values.

HTTP Status Code	Description	X-Agile-Status Values
200	Success	0
400	Bad request; missing or invalid para-meters	**-3**: parent directory does not exist. You specified a in X-Agile-Directory with a leading path that does not exist and you did not pass a true value in X-Agile-Recursive. **-8**: invalid path given (path too long). See Path Segment and File Name Lim-

HTTP Status Code	Description	X-Agile-Status Values
		itations. **-24**: You failed to specify a file to upload. **-25**: You attempted to upload multiple files. **-26**: invalid checksum **-33**: You included an invalid content type in the X-Agile-Content-Type header. **-39**: You passed an invalid value in the X-Agile-Recursive directory. **-51**: invalid encoding (You specified an invalid value for the X-Agile-Encoding header.)
401	Unauthorized, missing `X-Agile-Authorization` header	**-10001**: missing X-Agile-Authorization header
403	Invalid authentication credentials (invalid token)	**-10001**: invalid token
500	Internal server error	**-5**: internal backend service down
503	Service unavailable	**-22**: disk full

Response Headers

Header Name	Type	Description
X-Agile-Path	str	File name and path where the file was uploaded . The value will be URI-quoted if UTF8 encoding was used in the X-Agile-Encoding header.
X-Agile-Status	int	Contains response codes. (See HTTP Response Codes and Request Status Codes.)
X-Agile-Size	int	Size in bytes of the file that was uploaded
X-Agile-Checksum	str	File's SHA-256 hexadecimal digest

curl Sample Requests

Upload warning.png to the /symbols/warning directory.

```
curl -v -k \
-H "X-Agile-Authorization: a1ac4fca-2fed-44ac-9b20-d26aee578e67"\
--data-binary @/tmp/warning.png\
-H "X-Agile-Basename: warning.png"\
-H "X-Agile-Content-Detect: name"\
-H "X-Agile-Content-Type: image/png"\
-H "X-Agile-Directory: /symbols/warning"\
-H "X-Agile-Expose-Egress: POLICY"\
-H "X-Agile-MTime: 0"\
http://{Account name}.upload.llnw.net/post/raw
```

Sample Success Response

```
HTTP/1.1 100 Continue
HTTP/1.1 200 OK
Date: Tue, 23 Jun 2015 20:29:49 GMT
Server Apache is not deny listed
Server: Apache
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: X-Agile-Authorization, X-Content-Type
Access-Control-Allow-Methods: OPTIONS
X-Agile-Status: 0
Content-Length: 0
X-Agile-Checksum: da2d48c37dabd94c00bc9dc98aa2ca0255de1a1238e2525ab1d7edce398
X-Agile-Size: 1547
X-Agile-Path: /{Account Name}/symbols/warning/warning.png
Content-Type: text/json;charset=utf-8
```

Content Type Handling

The file upload APIs (`/post/file` and `/post/raw`) determine content type as follows:

- The content type specified by the `Content-Type` field of the attachment (form-data encoded payload[1]) takes precedence if set.
- The `X-Agile-Content-Type` header takes precedence over `X-Agile-Content-Detect`.
- The `application/octet-stream` is special content type, and if you provide it in the `X-Agile-Content-Type` header, the API attempts to detect the content type from the file name, usually the extension.
- If you provide both `X-Agile-Content-Type` and `X-Agile-Content-Detect`, then `X-Agile-Content-Type` takes precedence.
- If you provide neither `X-Agile-Content-Type` nor `X-Agile-Content-Detect`, the API defaults the content type to `X-Agile-Content-Detect: auto`.

[1]Here is an example of form-encoded data payload, where the content type is `text/plain`.

```
------------------------------44f9af2ecd0c
Content-Disposition: form-data; name="uploadFile"; filename="test.txt"
Content-Type: text/plain
Sample data
------------------------------44f9af2ecd0c--
```

Note:
Due to legacy reasons it is possible to pass values (`name`, `content`, `auto`) in `X-Agile-Content-Type` that instruct the API to detect the content type, but doing so is not recommended and may result in unpredictable results.

The values have the following meanings:

name: look up MIME type by using the file name, usually the extension

content: look up MIME type by scanning the contents of the file

auto: first look up by name, then by content if no name matches

Although the value `auto` for the `X-Agile-Content-Type` header is currently accepted, it will be deprecated at some point in the future. We recommend that you change you code to use alternative means of instructing the API to automatically detect the content type.

X-Content-Type Request Header

Note:
In the past it was possible to pass the value `auto` in the legacy `X-Content-Type` header to instruct the API to automatically detect the content type of the entity-body, but passing `auto` causes an error. To cause automatic content detection callers can either:

- Omit the `X-Content-Type` header, or
- Pass `auto` in the `X-Content-Detect` header

Uploading Files – Multipart

This section introduces multipart uploads, provides high-level information about multipart uploads, and points you to detailed instructions for using the related APIs. See the following sections:

- Introduction to Multipart Uploads
- Some Notes About Multipart Uploads
- Where to Go for Instructions

Introduction to Multipart Uploads

Multipart uploads allow you to upload an object in pieces. A multipart upload is an upload to the server that is created by uploading individual pieces of an object and generally comprises these steps:

1. Create a multipart upload.
2. Add pieces to the multipart upload.
3. Complete the multipart upload. This step concatenates the individual pieces together into a single object. Until you do this step, neither the upload nor the individual pieces are visible as files on *Origin Storage*.

For example if you have a 15GB file you can use the multipart capability to upload the file in 1MB pieces.

Note: Via *Origin Storage* policies, uploads are automatically replicated to multiple Bricks.

Note:
Although you can use multipart upload to upload any file, multipart is best for files over 10GB. Otherwise, use non-multipart uploads. See Uploading Files – Non-Multipart.

The multipart upload feature offers important advantages over non-multipart uploads:

- You can ingest pieces of an object in parallel, thereby greatly improving performance.
- In the case of a network failure, you loose only those pieces that did not upload before the failure as opposed to loosing an entire file if you had not chosen to use the multipart upload capability. The multipart upload maintains its state – it knows which pieces have already been uploaded so you can easily resume the upload after network errors are corrected.

Multipart upload capabilities are available in both the JSON-RPC interface and the HTTP interface, but not all functionality is available in both. The following table shows the capabilities that are available in each interface. A "Y" in a column indicates the capability is available and lack of a "Y" means otherwise.

Capability	Available in JSON-RPC Interface?	Available in HTTP Interface?
Create a multipart upload	Y	Y
Create multipart piece		Y
List the pieces in a multipart upload	Y	
List the multipart uploads that your user has created	Y	
Get the status of a multipart upload	Y	
Get a list of multipart or multipart piece status codes and corresponding descriptions	Y	
Complete a multipart upload	Y	Y
Abort a multipart upload	Y	
Restart a multipart upload	Y	

As indicated in the table above, *only the HTTP interface lets you create multipart pieces*.

Some Notes About Multipart Uploads

- You can create multiple multipart uploads for the same path (destination file) because each multipart upload has an ID unique to that upload. Note that although this is possible, it is not recommended because the most recent upload is not guaranteed to prevail.
- Multipart identifiers can only be used and managed by the user who created the multipart upload. Users cannot share multipart identifiers.
- The multipart framework does not check permissions on the destination path, so first ensure the destination is writable by your user.
- A process known as the "sweeper" or "garbage collector" periodically checks each multipart upload. The garbage collector performs actions such as transitioning multipart uploads to EXPIRED state if they have been in existence for longer than a configurable amount of time.

Where to Go for Instructions

See the following sections for instruction about the multipart upload APIs:

- Working with Multipart Uploads in the JSON-RPC Interface
- Working with Multipart Uploads in the HTTP Interface

See Multipart Example for a complete end-to-end multipart upload example.

Working with Multipart Uploads in the JSON-RPC Interface

The JSON-RPC interface provides multipart upload functionality explained in the following sections:

- Begin a Multipart Upload
- Get Status for a Multipart Upload
- Get String Equivalents of Multipart Status Codes
- List Your Multipart Uploads
- List Pieces in a Multipart Upload
- Restart a Multipart Upload
- Complete a Multipart Upload
- Abort a Multipart Upload

The only multipart capability not present in the JSON-RPC interface is the ability to add pieces to a multipart upload, which is in the HTTP interface only. See Create a Multipart Piece for information.

For general information about multipart uploads, see Uploading Files – Multipart.

As you read the information in the following sections, keep in mind the caveats mentioned in Some Notes About Multipart Uploads.

Begin a Multipart Upload

Method name: `createMultipart`

Initiates a multipart upload and returns an upload ID. The ID associates all parts in the upload and must be used each time you create a piece for the upload. You also include the ID in the final request to complete or abort the multipart upload request.

The final destination file will not be created until you have added pieces to, and completed, the upload.

An error will be returned if any parent directories does not exist. To create parent paths, you can use either:

- the makeDir2 API in the JSON RPC interface (see Create a Directory Recursively) or:
- the post/directory API in the HTTP interface (see Create a Directory)

Named Parameters Example

```
{
  "method": "createMultipart",
  "id": 1,
  "params": {
    "token": "c1455871-507e-4ceb-b01f-cb80208118a4",
    "path": "/Multipart10",
    "content_type": "text/plain",
    "mtime": 1461947066
  },
  "jsonrpc": "2.0"
}
```

Positional Parameters Example

Positional parameters must be applied in the same order shown in the named parameters example.

```
{
  "method": "createMultipart",
  "id": 1,
  "params": [
    "819c8567-175a-4e66-bd85-10b2e1dbf8e4",
    "Multipart10",
    "text/plain",
    1464896993
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
path	str	The destination file and path for the final object once the mutlipart upload is completed.
content_type	str	Optional MIME type. Example: `image/gif` If you don't pass a content type or if you pass an empty string, the API will derive the content type based on the file extension you passed for the `path` argument.

Parameter Name	Type	Description
		If you do not pass a value for content type and the value you passed for `path` did not have an extension, then the `content_type` name-value pair will be absent from the output of calls to `listMultipart` and `getMultipartStatus`. (See List Your Multipart Uploads and Get Status for a Multipart Upload respectively.) See also Content Types. Defaults to an empty string
mtime	int	Optional File last modification time in seconds since epoch Defaults to 0 (current time)

Return Codes

- **0**: success
- **-1**: You have exceeded the maximum number of mulitpart uploads (1,000,000) that have not been completed or aborted.
- **-16**: invalid path (path too long).
- **-20**: invalid last modified time
- **-21**: invalid content type
- **-23**: parent directory does not exist or is invalid. One or more parent segments in the path argument do not exist.
- **-27**: internal server error
- **-10001**: invalid token

Note:
For a list of error codes not specific to `createMultipart`, see Global Error Codes.

Response Data

On success returns an object with the following data:

- **code**: (int) return code; zero indicates success, non-zero indicates failure
- **mpid**: (str) multipart upload ID you can use in subsequent requests such as getting status for the mulipart upload

Example:

```
{
  "code": 0,
  "mpid": "fe275a9692b146a281f054063d8832bd"
}
```

On failure returns an object like the following, with `code` set to the appropriate value described in Return Codes:

```
{
  "code": -23
}
```

Python Sample Requests

Create a multipart upload where the destination file will be in the `/open-source-novels` directory. Specify the content type and mtime yourself:

```
>>> api.createMultipart(token, "/open-source-novels/million-page-novel",
"text/plain", 1437426380)
{"code": 0, "mpid": "2bb1fec7909d4f4ebca888176d58c402"}
```

Create a multipart upload where the destination file will be in the `/movies` directory, will have a content type that defaults to `video/mpeg`, and will have a last modification time that defaults to the current time.

```
>>> api.createMultipart(token, '/movies/rock-climbing.mpg')
{"code": 0, "mpid": "fe275a9692b146a281f054063d8832bd"}
```

Get Status for a Multipart Upload

Method name: `getMultipartStatus`

Gets the state of a multipart upload (new, ready, complete, and so on).

Named Parameters Example

```
{
  "method": "getMultipartStatus",
  "id": 2,
  "params": {
    "token": "eacfb76b-c9eb-45e7-8723-37ed5428dafa",
    "mpid": "a8109827cd2544f28142d469ce78c587"
  },
  "jsonrpc": "2.0"
}
```

Positional Parameters Example

Positional parameters must be applied in the same order shown in the named parameters example.

```
{
  "method": "getMultipartStatus",
  "id": 1,
  "params": [
    "b062d5e2-626b-4717-90bd-8d7410948e0e",
    "a8109827cd2544f28142d469ce78c587"
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
mpid	str	Multipart upload identifier from a prior call to `createMultipart` or `/multipart/create`. See Begin a Multipart Upload (JSON-RPC) and Begin a Multipart Upload (HTTP) respectively.

Return Codes

- **0**: success
- **-2**: You passed an invalid value for the mpid parameter.
- **-10001**: invalid token

Note:
For a list of error codes not specific to `getMultipartStatus`, see Global Error Codes.

Response Data

On success returns an object with the following data:

- **code**: (int) return code
- **content_type**: (str) MIME type. Example: image/gif
- **created**: (int) file creation time in seconds since epoch
- **error**: (int) error reason, or 0 if no errors occurred
- **mtime**: (int) last modification time in seconds since epoch
- **numpieces**: (int) number of pieces in the multipart upload
- **path**: (str) the destination file and path
- **state**: (int) state of the multipart upload. See Multipart Upload State Codes and Get String Equivalents of Multipart Status Codes.

Example:

```
{
  "path": "/{your Account name}/open-source-novels/million-page-novel",
  "code": 0,
  "content_type": "text/plain",
  "numpieces": 0,
  "created": 1437429243,
  "state": 1,
  "mtime": 1437426380,
  "error": 0
}
```

On failure returns an object like the following, with `code` set to the appropriate value described in Return Codes:

```
{
  "code": -2
}
```

> **Note:**
> When you created the multipart upload, if you did not pass a value for content type and the value you passed for path did not have a file extension, then the `content_type` name-value pair will be absent from the output.

Python Sample Requests

Get the status for multipart upload 2bb1fec7909d4f4ebca888176d58c402.

```
>>> api.getMultipartStatus(token, "2bb1fec7909d4f4ebca888176d58c402")
{u'code': 0, u'created': 1437429243, u'mtime': 1437426380, u'numpieces': 2,
u'state': 2, u'content_type': u'text/plain', u'error': 0, u'path': u'/{your Account
name}/open-source-novels/million-page-novel'}
```

Get String Equivalents of Multipart Status Codes

Method name: `getMultipartStatusMap`

For either mulipart uploads or multipart pieces, gets all possible status codes and their string equivalents. This is useful for programmatically getting string values for the states that are returned from calls to the following methods:

- `getMultipartStatus` (see Get Status for a Multipart Upload)
- `listMultipartPiece` (see List Pieces in a Multipart Upload)
- `listMultipart` (see List Your Multipart Uploads)

Named Parameters Example

```
{
  "method": "getMultipartStatusMap",
  "id": 2,
  "params": {
    "token": "95bac558-bef9-4c6c-a278-39fe2663b984",
    "name": "upload"
  },
  "jsonrpc": "2.0"
}
```

Positional Parameters Example

Positional parameters must be applied in the same order shown in the named parameters example.

```
{
  "method": "getMultipartStatusMap",
  "id": 1,
  "params": [
    "1a72bc74-b409-4e46-8cf6-1c1b56bd4666",
    "upload"
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
name	str	Optional Indicates the state codes you want. Valid values: **'upload'**: get codes for multipart uploads **'piece'**: get codes for mulipart pieces Defaults to 'upload'

Return Codes

- **0**: success
- **-19**: invalid value for `name` parameter
- **-10001**: invalid value for `token` parameter

Note:
For a list of error codes not specific to `getMultipartStatusMap`, see Global Error Codes.

Response Data

On success returns an object with the following data:

- **code**: (int) return code; zero indicates success, non-zero indicates failure
- **map**: an object that contains codes, each mapped to a descriptive name. The exact values depend on the value you pass for the `name` parameter. See Multipart State Codes for descriptions of values in the map.

Example of states returned for multipart uploads:

```
{
  "code": 0,
  "map": {
    "11": "EXPIRED",
    "0": "UNKNOWN",
    "12": "FAILED",
    "1": "NEW",
    "2": "READY",
    "3": "COMPLETE",
    "5": "JOIN",
    "6": "SUCCESS",
    "8": "DELETED",
    "9": "ERROR",
    "10": "ABORT"
  }
}
```

Example of states returned for a multipart piece:

```
{
  "code": 0,
  "map": {
    "11": "ABORT",
    "0": "UNKNOWN",
    "12": "EXPIRED",
    "1": "NEW",
    "13": "FAILED",
    "5": "JOIN",
    "6": "SUCCESS",
    "8": "DELETED",
    "9": "ERROR",
    "10": "SKIPPED"
  }
}
```

On failure returns an object like the following, with code set to the appropriate value described in Return Codes:

```
{
  "code": -19
}
```

Python Sample Requests

Get states for multipart uploads:

```
>>> api.getMultipartStatusMap(token, 'upload')
{"map": {"11": "EXPIRED", "0": "UNKNOWN", "12": "FAILED", "1": "NEW", "2": "READY",
"3": "COMPLETE", "5": "JOIN", "6": "SUCCESS", "8": "DELETED", "9": "ERROR", "10":
"ABORT"} "code": 0,}
```

Get states for multipart upload pieces:

```
>>> api.getMultipartStatusMap(token, 'piece')
{u'map': {u'11': u'ABORT', u'10': u'SKIPPED', u'13': u'FAILED', u'12': u'EXPIRED',
u'1': u'NEW', u'0': u'UNKNOWN', u'5': u'JOIN', u'6': u'SUCCESS', u'9': u'ERROR',
u'8': u'DELETED'}, u'code': 0}
```

See Multipart State Codes for descriptions of values in the maps in the preceding examples.

List Your Multipart Uploads

Method name: `listMultipart`

Lists all multipart uploads that were created by your user.

Named Parameters Example

```
{
  "method": "listMultipart",
  "id": 2,
  "params": {
    "token": "ffea4fab-abd9-438d-a700-232af6dc9fb9",
    "cookie": 0,
    "pagesize": 5
  },
  "jsonrpc": "2.0"
}
```

Positional Parameters Example

Positional parameters must be applied in the same order shown in the named parameters example.

```
{
  "method": "listMultipart",
  "id": 0,
  "params": [
    "1c3d7040-6809-41ce-9cfa-771abe304f64",
    0,
    5
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and

Parameter Name	Type	Description
		Logging in Using the HTTP Interface respectively.
cookie	int	Optional Tracks the number of multipart uploads that have been returned to the caller. Omit this argument or pass 0 the first time you call the function. For all subsequent calls, pass the cookie value returned from the prior call. Defaults to 0.
pagesize	int	Optional The number of mulipart uploads to return from each call to the method. Defaults to 100.

Return Codes

- **0**: success
- **-3**: invalid cookie
- **-14**: invalid pagesize
- **-10001**: invalid value for `token` parameter

Note:
For a list of error codes not specific to `listMultipart`, see Global Error Codes.

Response Data

On success returns an object with the following data:

- **code**: (int) return code; zero indicates success, non-zero indicates failure
- **cookie**: (int) number of results returned
- **multipart**: array of multipart objects. Data for each object is:
  - **content_type**: (str) MIME type. Example: image/gif[1]
  - **error**: (int) error (if any) that occurred when the multipart upload was created
  - **gid**: (int) group ID of caller that created the multipart upload
  - **mpid**: (str) multipart identifier assigned when the multipart upload was created
  - **mtime**: (int) last modification time in seconds since epoch
  - **path**: (str) multipart upload destination file and path
  - **state**: (int) status of the multipart upload. See Multipart Upload State Codes and Get String Equivalents of Multipart Status Codes
  - **uid**: (int) ID of the user that created the multipart upload

Example:

```
{
  "code": 0,
  "cookie": 2,
  "multipart": [
    {
      "uid": 12020,
      "path": "/{your Account name}/multipart/multipart4.txt",
      "gid": 100,
      "content_type": "text/plain",
      "state": 1,
      "mtime": 1537165142,
      "error": 0,
      "mpid": "93b19d0c7e0d4669a01a5c0b2eba9513"
    },
    {
      "uid": 12020,
      "path": "/{your Account name}/open-source-novels/thousand-page-novel.txt",
      "gid": 100,
      "content_type": "text/plain",
      "state": 1,
      "mtime": 1437426380,
      "error": 0,
      "mpid": "5b45cfeb127049d7a74bea389ee4a0b6"
    }
  ]
}
```

Note:
[1]When you created the multipart upload, if you did not pass a value for content type and the value you passed for path did not have a file extension, then the `content_type` name-value pair will be absent from the output.

On failure returns an object like the following, with code set to the appropriate value described in Return Codes:

```
{
  "code": -3
}
```

Python Sample Requests

Display the pieces in multipart uploads that are in states specified by the variable `desiredStates`. Demonstrates how to use `listMultipart`, `listMultipartPiece`, and `getMultipartStatusMap`.

```
import jsonrpclib

# Exceptions

class ListMultipartError(RuntimeError):
  def __init__(self, arg):
    self.args = arg

class ListMultipartPieceError(RuntimeError):
  def __init__(self, arg):
    self.args = arg

class GetStatusMapError(RuntimeError):
  def __init__(self, arg):
    self.args = arg


# User-Defined Variables
# -----------------------------------
desiredStates = [8,11] #States to look for.

# Functions
# -----------------------------------

def getUploadDescr( code ):
  result = uploadMap['map'][ str(code) ]
  return result

def getPieceDescr( code ):
  result = pieceMap['map'][ str(code) ]
  return result


def printMultipart( theList ):
  print ('Length of list: ' + str( len(theList) ))
  for i in range( len(theList) ):
    print (theList[i]['path'])

def getPieces( mpid ):
  piecesRet = []
  pageSize = 1000

  mPieceResults  = api.listMultipartPiece( token, mpid, 0, pageSize )
  code = mPieceResults['code']
  if code != 0:
    msg ='Error getting pieces for multipart upload ' + mpid + '\n  Return code: ' +
str( code ) + '\n  See API documentation for details.'
    raise ListMultipartPieceError( msg )
  piecesRet += mPieceResults['pieces']
  cookie = mPieceResults['cookie']

  while ( cookie !=0 ):
    mPieceResults  = api.listMultipartPiece( token, mpid, cookie, pageSize )
    code = mPieceResults['code']
    if code != 0:
      msg ='Error getting pieces for multipart upload ' + mpid + '\n  Return code: '
+ str( code ) + '\n  See API documentation for details.'
      raise ListMultipartPieceError( msg )
    piecesRet += mPieceResults['pieces']
    cookie = mPieceResults['cookie']

  return piecesRet

# Main Processing
# -----------------------------------
try:
  hdrSep    = '=================================================='

  #Create upload state map
  uploadMap = api.getMultipartStatusMap(token, 'upload')
  if uploadMap['code'] != 0:
    msg ='Error getting multipart upload status map.'
    msg += '\nReturn code: ' + str( res )
    msg += '\nSee API documentation for details.'
    raise GetStatusMapError( msg )

  #Create piece state map
  pieceMap = api.getMultipartStatusMap(token, 'piece')
  if pieceMap['code'] != 0:
    msg ='Error getting multipart upload status map.'
    msg += '\nReturn code: ' + str( res )
    msg += '\nSee API documentation for details.'
    raise GetStatusMapError( msg )

  #######################
  # Get multipart uploads
  #######################
  pageSize = 10

  print ('Multipart uploads')
  print ('-----------------')
  mUploadResults = api.listMultipart(token, 0, pageSize)
  cookie = mUploadResults['cookie']
  print ('cookie: ' + str(cookie))

  while cookie != 0:
    code = mUploadResults['code']
    if code != 0:
      msg ='Error calling listMultipart.'
      msg += '\nReturn code: ' + str( code )
      msg += '\nSee API documentation for details.'
      raise ListMultipartError( msg )

    multipartList = mUploadResults['multipart']

    ############################################################################
    #Go through the mulipart upload list, checking if status is in desiredStates
    ############################################################################
    for i in range( len( multipartList ) ):
      # upload's state
      state = multipartList[i]['state']
      if state in desiredStates:
        path = multipartList[i]['path']
        thisHdrSep = hdrSep
        if len( path ) > 49:
          while len( thisHdrSep ) < len( path ):
            thisHdrSep += thisHdrSep

        thisHdrSep = thisHdrSep[0: len( path )]
        print ('')
        print (thisHdrSep)
        print (path)
        print (thisHdrSep)
        descr = getUploadDescr(state))
        mpid = multipartList[i]['mpid']
        print ('Multipart ID:    ' + mpid)
        print ('Multipart State: ' + str( state ) + ' (' + descr + ')' )

        ####################################
        #List pieces in the multipart upload
        ####################################
        pieces = []
        pieces = getPieces( mpid )
        print ('')
        print ('{0:5s} {1:16s} {2:10s} {3:3s}'.format('Piece', '| State', '| Size',
'| Error'))
        print ('------|----------------|----------|------')
        for j in range( len( pieces ) ):
          p = pieces[j]
          pieceNum        = str( p['number'])
          state           = p['state']
          descr           = getPieceDescr( state)
          pieceStateDescr = '| ' + str(state) + ' (' + descr + ')'
          pieceSize       = '| ' + str(p['size'])
          pieceError      = '| ' + str(p['error'])
          print ('{0:5s} {1:16s} {2:10s} {3:3s}'.format( pieceNum, pieceStateDescr,
pieceSize, pieceError  ))

    mUploadResults = api.listMultipart(token, cookie, pageSize)
    cookie = mUploadResults['cookie']

except LoginError,e:
  print (''.join( e.args ) )

except ListMultipartError,e:
  print (''.join( e.args ))

except ListMultipartPieceError,e:
  print (''.join( e.args ))

except GetStatusMapError,e:
  print (''.join( e.args ))

finally:
  print ('\nLogging out')
  res = api.logout(token)
  if res != 0:
    msg = 'Error logging out.'
    msg += '\nReturn code: ' + str( res )
    msg += '\nSee API documentation for details.'
    print (msg)
```

List Pieces in a Multipart Upload

Method name: `listMultipartPiece`

Lists pieces in a specified multipart upload along with information such as state and size. Pieces are listed in the order that they were added to the multipart upload. Duplicate pieces are included in the listing; for example if you added piece number 2 twice, both pieces are visible in the output.

Named Parameters Example

```
{
  "method": "listMultipartPiece",
  "id": 1,
  "params": {
    "token": "282edeb9-b2b7-4f19-a879-dd9e8b8b4d22",
    "mpid": "a8109827cd2544f28142d469ce78c587",
    "cookie": 0,
    "pagesize": 5
  },
  "jsonrpc": "2.0"
}
```

Positional Parameters Example

Positional parameters must be applied in the same order shown in the named parameters example.

```
{
  "method": "listMultipartPiece",
  "id": 0,
  "params": [
    "a991c836-28c6-4584-948d-3ea5c6a9b7d5",
    "91842e50903b4251848e32b67d8d7137",
    0,
    5
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
mpid	str	Multipart upload identifier from a prior call to `createMultipart` or `/multipart/create`. See Begin a Multipart Upload (JSON-RPC) and Begin a Multipart Upload (HTTP) respectively.
cookie	int	Optional Tracks the progress of calls to `multipartPiece` for a given multipart upload identifier. Omit this argument or pass 0 the first time you call the function. For all subsequent calls, pass the cookie value returned from the prior call. Defaults to 0
pagesize	int	Optional The number of pieces to return from each call to the method. Defaults to 100.

Return Codes

- **0**: success
- **-2**: invalid multipart upload identifier
- **-3**: invalid cookie
- **-14**: invalid page size
- **-26**: multipart services down or unavailable
- **-10001**: invalid token

Note:
For a list of error codes not specific to `listMultipartPiece`, see Global Error Codes.

Response Data

On success returns an object with the following data:

- **code**: (int) return code; zero indicates success, non-zero indicates failure
- **cookie**: (int) ID of the last piece returned, or zero if all results have been returned. Use this value when you are making successive calls to the method rather than attempting to return all pieces at once. See Successively list the pieces in a multipart upload.
- **pieces**: array of multipart pieces. Data for each piece is:
  - **error**: (int) error (if any) that occurred when the multipart upload piece was added
  - **number**: (int) piece (part) number assigned to the piece when it was added to the multipart upload. See Create a Multipart Piece.
  - **size**: (int) size of the piece in bytes
  - **state**: (int) status of the piece. See Multipart Piece State Codes and Get String Equivalents of Multipart Status Codes.

After all pieces have been returned, **pieces** is an empty array.

Example results from calling the function, requesting one piece from a multipart upload with multiple pieces:

```
{
  "pieces": [
    {
      "number": 1,
      "size": 23,
      "state": 8,
      "error": 0
    }
  ],
  "code": 0,
  "cookie": 31376064
}
```

Example results from calling the function after all pieces have been returned:

```
{
  "pieces": [],
  "code": 0,
  "cookie": 0
}
```

On failure returns an object like the following, with code set to the appropriate value described in Return Codes:

```
{
  "code": -2
}
```

Python Sample Requests

List the pieces in a given multipart upload specified by the variable `mpid`.

```
import jsonrpclib

#-----------------------
# User-defined variables
#-----------------------
mpid = 'your Multipartupload ID'
pageSize = 5

#-----------
# Exceptions
#-----------
class ListMultipartPieceError(RuntimeError):
  def __init__(self, arg):
    self.args = arg
#----------
# Functions
#----------
def getStateDescr(code):
  result = stateMap['map'][str(code)]
  return result

def printPieces( pieces ):
  global hasPieces
  for i in range( len(pieces) ):
    hasPieces = True
    p = pieces[i]
    pieceNum        = str( p['number'])
    state           = p['state']
    descr           = getStateDescr( state)
    pieceStateDescr = '| ' + str(state) + ' (' + descr + ')'
    pieceSize       = '| ' + str(p['size'])
    pieceError      = '| ' + str(p['error'])
    print ('{0:5s} {1:16s} {2:10s} {3:3s}'.format( pieceNum, pieceStateDescr,
pieceSize, pieceError  )
try:
  stateMap = api.getMultipartStatusMap(token, 'piece')
  hasPieces = False

  #########################################
  # Make initial call to listMultipartPiece
  #########################################
  results  = api.listMultipartPiece( token, mpid, 0, pageSize )
  code = results['code']
  if code != 0:
    msg ='Error getting pieces for multipart upload ' + mpid + '\n  Return code: ' +
str( code ) + '\n  See API documentation for details.'
    raise ListMultipartPieceError( msg )

  print ('Pieces in ' + mpid)
  print ('')
  print ('{0:5s} {1:16s} {2:10s} {3:3s}'.format('Piece', '| State', '| Size', '|
Error'))
  print ('------|----------------|----------|------')

  printPieces( results['pieces'] )
  cookie = results['cookie']

  ###########################
  # Make any additional calls
  ###########################
  while ( cookie !=0 ):
    results  = api.listMultipartPiece( token, mpid, cookie, pageSize )
    code = results['code']
    if code != 0:
      msg ='Error getting pieces for multipart upload ' + mpid + '\n  Return code: '
+ str( code ) + '\n  See API documentation for details.'
      raise ListMultipartPieceError( msg )

    printPieces( results['pieces'] )
    cookie = results['cookie']

    if hasPieces == False:
      print ('This multipart upload has no pieces.')

except ListMultipartPieceError,e:
  print (''.join( e.args ))

finally:
  print ('\nLogging out...\n')
  api.logout(token)
```

Restart a Multipart Upload

Method name: `restartMultipart`

Restarts a failed multipart upload. You can restart a multipart upload only if it is in the FAILED state (code = 12) and was created after the most recent time that the garbage collector ran.

Named Parameters Example

```
{
  "method": "restartMultipart",
  "id": 1,
  "params": {
    "token": "b248ccfc-fa9c-418a-a647-43dceee99340",
    "mpid": "91842e50903b4251848e32b67d8d7137"
  },
  "jsonrpc": "2.0"
}
```

Positional Parameters Example

Positional parameters must be applied in the same order shown in the named parameters example.

```
{
  "method": "restartMultipart",
  "id": 0,
  "params": [
    "b2784388-0b14-431c-ac1d-e4f830ae7756",
    "91842e50903b4251848e32b67d8d7137"
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
mpid	str	Multipart upload identifier from a prior call to `createMultipart` or `/multipart/create`. See Begin a Multipart Upload (JSON-RPC) and Begin a Multipart Upload (HTTP) respectively.

Return Codes

- **0**: success
- **-2**: invalid multipart upload identifier
- **-22**: unable to restart the multipart upload; upload was not in a suitable state
- **-10001**: invalid value for `token` parameter

Note:
For a list of error codes not specific to `restartMutlipart`, see Global Error Codes.

Response Data

Returns only the codes discussed in Return Codes. Does not return any data structures.

Python Sample Requests

Successfully restart a multipart piece:

```
>>> api.restartMultipart(token, 'eaeb1484045a48d1b23bf02ef43a2165')
0
```

Attempt to restart a multipart piece that is in SUCCESS state (an unsuitable state to be restarted):

```
>>> api.restartMultipart(token, 'a754afa6123649bf9382ca11289283ba')
-22
```

Complete a Multipart Upload

Method name: `completeMultipart`

Completes a multipart upload, assembling the associated pieces into a final file.

When the file creation is complete, the system sets the parent directory's mtime (last modification time) to the current system time.

Named Parameters Example

```
{
  "method": "completeMultipart",
  "id": 1,
  "params": {
    "token": "2e31a024-a37d-49fb-9985-40eba7d61caa",
    "mpid": "91842e50903b4251848e32b67d8d7137"
  },
  "jsonrpc": "2.0"
}
```

Positional Parameters Example

Positional parameters must be applied in the same order shown in the named parameters example.

```
{
  "method": "completeMultipart",
  "id": 0,
  "params": [
    "d87b65ab-df84-46e5-aa49-27e85c6ad6f5",
    "91842e50903b4251848e32b67d8d7137"
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
mpid	str	Multipart upload identifier from a prior call to `createMultipart` or `/multipart/create`. See Begin a Multipart Upload (JSON-RPC) and Begin a Multipart Upload (HTTP) respectively.

Return Codes

- **0**: success
- **-2**: invalid multipart upload identifier
- **-4**: the multipart upload does not have any parts
- **-5**: the multipart upload is missing some pieces. (The range of piece numbers is not contiguous.)
- **-8**: the multipart upload has already been completed or has been deleted or expired
- **-17**: the multipart upload has been aborted
- **-10001**: invalid value for `token` parameter

Note:
For a list of error codes not specific to `completeMultipart`, see Global Error Codes.

Response Data

On success returns an object with the following data:

- **code**: (int) return code; zero indicates success, non-zero indicates failure
- **numpieces**: (int) number of pieces in the multipart upload

Example:

```
{
  "code": 0,
  "numpieces": 1
}
```

On failure returns an object like the following, with code set to the appropriate value described in Return Codes:

```
{
  "code": -4
}
```

Python Sample Requests

Complete a multipart upload.

```
>>> api.completeMultipart(token, '5d50c59de3e645ac8df0f209290c1c29')
{u'numpieces': 4, u'code': 0}
```

Abort a Multipart Upload

Method name: `abortMultipart`

Aborts a multipart upload that has not already been completed or aborted. Aborting a multipart upload invalidates the given multipart upload ID and all associated pieces.

Aborted multipart uploads and pieces still exist and show up in listings and status checks but are in ABORTED state, Once aborted, a multipart upload cannot be completed or have pieces added to it. Multipart uploads cannot be aborted after calling `completeMultipart`.

Named Parameters Example

```
{
  "method": "abortMultipart",
  "id": 1,
  "params": {
    "token": "2754500a-a62d-4cb1-a57a-c84d572862b6",
    "mpid": "91842e50903b4251848e32b67d8d7137"
  },
  "jsonrpc": "2.0"
}
```

Positional Parameters Example

Positional parameters must be applied in the same order shown in the named parameters example.

```
{
  "method": "abortMultipart",
  "id": 0,
  "params": [
    "889dfb85-3083-44f6-9487-0fdc793fde94",
    "91842e50903b4251848e32b67d8d7137"
  ],
  "jsonrpc": "2.0"
}
```

Parameter Descriptions

Parameter Name	Type	Description
token	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
mpid	str	Multipart upload identifier from a prior call to `createMultipart` or `/multipart/create`. See Begin a Multipart Upload (JSON-RPC) and Begin a Multipart Upload (HTTP) respectively.

Return Codes

- **0**: success
- **-2**: invalid multipart identifier
- **-17**: multipart upload has already been aborted
- **-18**: invalid state; cannot be aborted
- **-10001**: invalid value for `token` parameter

Note:
For a list of error codes not specific to `abortMultipart`, see Global Error Codes.

Response Data

Returns only the codes discussed in Return Codes. Does not return any data structures.

Python Sample Requests

Successfully abort a multipart upload:

```
>>> api.abortMultipart(token, '673abb20317e45ac95325ab8ba574399')
0
```

Working with Multipart Uploads in the HTTP Interface

The HTTP interface lets you create a multipart upload, add pieces to it, and complete the upload as explained in the following sections:

- Begin a Multipart Upload
- Create a Multipart Piece
- Complete a Multipart Upload

For general information about multipart uploads, see Uploading Files – Multipart.

As you read information in the following sections, keep in mind the caveats mentioned in Some Notes About Multipart Uploads.

Begin a Multipart Upload

```
POST to /multipart/create
```

Initiates a multipart upload and returns an upload ID. The ID associates all parts in the upload and must be used each time you create a piece for the upload. You also include the ID in the final request to complete or abort the multipart upload request.

The final destination file will not be created until you have added pieces to, and completed, the upload.

An error will be returned if any parent directories does not exist. To create parent paths, you can use either:

- the makeDir2 API in the JSON RPC interface (see Create a Directory Recursively) or:
- the post/directory API in the HTTP interface (see Create a Directory)

Request Headers

Header Name	Type	Description
X-Agile-Authorization	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
X-Agile-Basename	str	Optional Name of file as it will be stored in *Origin Storage*, minus the path. Forward slashes / are not allowed. If you don't provide a value for `X-Agile-Basename`, the API will create a file name in the form of `mpart-<UUID4 in hex notation>`. Example: `mpart-ddbb5628fdd3433181ba9771f5709190` (UUID4 is a random Universally Unique ID.)
X-Agile-Content-Type	str	Optional Specifies content type of the file to upload. Allows content type to be overridden or detected. If this header is not present or the content type given is invalid, the content type will be derived automatically from the destination file extension. If a suitable MIME type is not detected from the MIME attachment, the MIME type will be used from the file name's extension. Must be a valid *Origin Storage* MIME type. See Content Types
X-Agile-Directory	str	Optional Directory on *Origin Storage* where the multipart upload is to be created. Defaults to root directory /
X-Agile-MTime	str	Optional File last modification time in seconds since epoch. Defaults to zero (current time)
X-Agile-Encoding	str	Optional

Header Name	Type	Description
		The default behavior is to treat paths as US ASCII. If the values for `X-Agile-Basename` and `X-Agile-Directory` contain UTF-8 characters, then include this header and set its value to `UTF8`, which will cause the API to treat values as URI quote paths. Note: • You must quote paths according to URL parameter quoting defined in RFC3986 of the Internet Engineering Task Force. • Characters `%20` and + turn into spaces. • Special characters you must handle are: percent sign, dollar sign, ampersand, plus sign, comma, forward slash, colon, semi colon, equal sign, question mark and the @ symbol. See Using UTF-8 Characters in Request Headers.

HTTP Response Codes and Request Status Codes

On success the call returns an HTTP 200 status code and 0 in the `X-Agile-Status` header.

On error, the HTTP Status Code is an error code, and the error is reflected in the `X-Agile-Status` header.

The following table provides details about response values.

HTTP Status Code	Description	X-Agile-Status Header Values
200	Success	0
400	Bad request; missing or invalid parameters	**-1**: The multipart upload would have exceeded the maximum number of mulitpart uploads (1,000,000) that have not been completed or aborted. **-16**: invalid path (path too long). **-20**: invalid last modified time **-21**: invalid content type **-23**: parent path does not exist or is invalid. One or more parent segments in the `X-Agile-Directory` header do not exist. **-51**: invalid encoding type. See also Using UTF-8 Characters in Request Headers.
401	Unauthorized; missing `X-Agile-Authorization` header	-10001
403	Forbidden; access denied or invalid value in the `X-Agile-Authorization` header	-10001
500	Internal server error	-27

HTTP Status Code	Description	X-Agile-Status Header Values
503	Service unavailable or temporarily down	-26

Response Headers

Header Name	Type	Description
X-Agile-Status	int	Contains response codes. (See HTTP Response Codes and Request Status Codes.)
X-Agile-Meta	str	Contains the following fields, which show the content type and mtime of the multipart upload: `content_type=<type> mtime=<time>` The fields echo the values you passed in the `X-Agile-Content-Type` and `X-Agile-MTime` headers. If you did not pass values, then the `content_type` field is set to `None` and `mtime` is 0. An mtime of 0 means the multipart upload will have an mtime equal to the time when the upload was created.
X-Agile-Multipart	str	Multipart identifier to use in future requests. **Note:** The returned identifier can only be managed by the user who created it. Users cannot share identifiers.
X-Agile-Path	str	Absolute destination path where the multipart upload will be created when it is completed.

curl Sample Request

Create a multipart upload. The final file name will be multipart1.txt in the /multipart directory.

```
curl -v -k \
-H "X-Agile-Authorization: 41dc1633-60ff-4479-9499-f1d30c9df00c"\
-H "X-Agile-Basename: multipart1.txt"\
-H "X-Agile-Directory: /multipart"\
http://{Account name}.upload.llnw.net/multipart/create
```

Sample Success Response

```
HTTP/1.1 200 OK
Date: Wed, 22 Jul 2015 16:22:54 GMT
Server Apache is not deny listed
Server: Apache
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: X-Agile-Authorization, X-Content-Type
Access-Control-Allow-Methods: OPTIONS
```

```
X-Agile-Status: 0
Content-Length: 0
X-Agile-Meta: content_type=None mtime=0
X-Agile-Multipart: 5d50c59de3e645ac8df0f209290c1c29
X-Agile-Path: /{Account name}/multipart/multipart1.txt
Content-Type: text/json;charset=utf-8
```

Create a Multipart Piece

```
POST to /multipart/piece
```

Adds a piece identified by a piece (part) number (X-Agile-Part) to a multipart upload via a raw HTTP POST upload. The POST body should be raw data that will be written to the piece. You can add pieces in any order as long as there are not gaps in piece numbers when you complete the multpart upload. So for example if you add pieces with numbers 1 and 3 but omit piece number 2 and then you attempt to complete the multipart upload, it fails to complete.

There is a limit to the number of pieces you can add, the size of any piece, and the total size of the upload.

Request Headers

Header Name	Type	Description
X-Agile-Authorization	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
X-Agile-Multipart	str	Multipart upload identifier from a prior call to `createMultipart` or `/multipart/create`. See Begin a Multipart Upload (JSON-RPC) and Begin a Multipart Upload (HTTP) respectively.
X-Agile-Part	str	Number to assign to the piece. Piece numbers must start with 1 and increase by 1 with each piece added. Must be unique. The range of part numbers must be contiguous. Adding a piece with a duplicate part number replaces the existing piece.

HTTP Response Codes and Request Status Codes

On success the call returns an HTTP 200 status code and 0 in the `X-Agile-Status` header.

On error, the HTTP Status Code is an error code, and the error is reflected in the `X-Agile-Status` header.

The following table provides details about response values.

HTTP Status Code	Description	X-Agile-Status Values
200	Success	0
400	Bad request; missing or invalid parameters	**-2**: invalid multipart ID in X-Agile-Multipart **-3**: invalid piece number in X-Agile-Part

HTTP Status Code	Description	X-Agile-Status Values
		-6: invalid checksum **-7**: invalid URL **-8**: You attempted to add a piece to a multipart upload that has been completed, deleted, or expired. **-10**: You attempted to add a piece that would have exceeded the maximum number of pieces (1,000) that can be added to a multipart upload. **-11**: You attempted to add a piece that is larger than the maximum allowable size (100 GB). **-17**: multipart upload (or piece) has been aborted. You tried adding a piece to a multipart upload that has been aborted. **-24**: Multipart upload format is invalid (bad content type) **-25**: You attempted to add a piece that would have caused the size of the multipart upload to exceed its maximum size (20 TB).
403	Invalid authentication credentials (invalid token)	**-10001**: Invalid token
500	Internal server error	**-49**: Size of file on Landing Pad does not match the Content-Length header value that your client sent.
503	Service unavailable	**-22**: disk full **-26**: server unavailable

Response Headers

Header Name	Type	Description
X-Agile-Status	int	Contains response codes. (See HTTP Response Codes and Request Status Codes.)
X-Agile-Checksum	str	Piece’s SHA-256 hexadecimal digest
X-Agile-Size	int	Piece's size in byes

curl Sample Request

Submit a file called piece1.txt.

```
curl -v -k \
-H "X-Agile-Authorization: 87970a2e-2141-455f-8381-e240fc581858"\
-H "X-Agile-Multipart: a754afa6123649bf9382ca11289283ba"\
-H "X-Agile-Part: 1"\
--data-binary @/piece1.txt\
http://{Account name}.upload.llnw.net/multipart/piece
```

Sample Success Response

```
HTTP/1.1 200 OK
Date: Thu, 16 Jul 2015 22:19:28 GMT
Server Apache is not deny listed
Server: Apache
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: X-Agile-Authorization, X-Content-Type
Access-Control-Allow-Methods: OPTIONS
X-Agile-Status: 0
Content-Length: 0
X-Agile-Checksum: 6fed09a7dd6283ac8f733c0ecfc5a2dad31e6a01e920ab19cd865558d4a5a
X-Agile-Size: 23
Content-Type: text/json;charset=utf-8
```

Complete a Multipart Upload

```
POST to /multipart/complete
```

Completes a multipart upload, concatenating the individual pieces into the final file defined by the `X-Agile-Basename` request header in the creation step. (See Begin a Multipart Upload.). Note that neither the final file or individual pieces are visible as files until you complete the multipart upload. You can, however, get information about a multipart upload and its pieces using the `listMultipart` and `listMultipartPiece` calls. (See List Your Multipart Uploads and List Pieces in a Multipart Upload respectively.)

When the file creation is complete, the system sets the parent directory's mtime (last modification time) to the current system time.

You can't complete a multipart upload that has no pieces, you can only abort it (See Abort a Multipart Upload.) You can't complete a multipart upload that has already been completed or aborted.

Request Headers

Header Name	Type	Description
X-Agile-Authorization	str	Valid token from a call to `login` (JSON-RPC interface) or `/account/login` (HTTP interface). See Log In and Logging in Using the HTTP Interface respectively.
X-Agile-Multipart	str	Multipart upload identifier from a prior call to `createMultipart` or `/multipart/create`. See Begin a Multipart Upload (JSON-RPC) and Begin a Multipart Upload (HTTP) respectively.

HTTP Response Codes and Request Status Codes

On success the call returns an HTTP 200 status code and 0 in the `X-Agile-Status` header.

On error, the HTTP Status Code is an error code, and the error is reflected in the `X-Agile-Status` header.

The following table provides details about response values.

HTTP Status Code	Description	<response code header name> / JSON Object Values
200	Success	0
400	Bad request; missing or invalid parameters	**-2**: You passed an invalid ID in the `X-Agile-Multipart` header. **-4**: You attempted to complete a multipart upload that has no pieces. **-5**: You attempted to complete a multipart upload with missing pieces. (The range of piece numbers in the multipart upload is not contiguous.) **-8**: You attempted to complete a multipart upload that has already been completed. **-17**: You attempted to complete a multipart upload that has been aborted.
500	Internal server error	**-13**: The lock on the multipart upload expired before the upload could be completed.
503	Service unavailable	**-26**: service down or unavailable (try again later).

Response Headers

Header Name	Type	Description
X-Agile-Status	int	Contains response codes. (See HTTP Response Codes and Request Status Codes.)
X-Agile-Parts	str	Total number of pieces completed
X-Agile-Multipart	str	Multipart upload ID

curl Sample Request

Complete a multipart upload.

```
curl -v -k \
-H "X-Agile-Authorization: 668f7342-5553-4b57-9cea-edb0b8589ae"\
-H "X-Agile-Multipart: 8dfe3fdbbef64dfc8f290da4fd8aef00"\
http://{Account name}.upload.llnw.net/multipart/complete
```

Sample Success Response

```
HTTP/1.1 200 OK
Date: Tue, 11 Aug 2015 19:23:36 GMT
Server Apache is not deny listed
Server: Apache
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: X-Agile-Authorization, X-Content-Type
Access-Control-Allow-Methods: OPTIONS
X-Agile-Status: 0
Content-Length: 0
X-Agile-Parts: 1000
X-Agile-Multipart: 8dfe3fdbbef64dfc8f290da4fd8aef00
Content-Type: text/json;charset=utf-8
```

Submitting Batch Requests

To submit a batch request, create a JSON array with at least one request object and no more than three request objects, where each request object is a valid, separately submittable request (has all required parameters). Each request object must contain an ID that is unique within the batch. In accordance with the JSON-RPC 2.0 Batch Specification, the *Origin Storage* server can process requests in a batch in any order and with any concurrency.

See Successful Batch Requests for examples.

For information about batch requests not presented here, please see the JSON-RPC 2.0 Batch Specification.

Note:
Only the /jsonprc2 endpoint supports batch requests.

Response Data

If the batch request contains a JSON error or has no request objects or has more than three request objects, the API returns a single JSON object with failure information.

Otherwise, the API returns a response object for each request you submitted in the batch. Each response object contains a `result` name/value pair with success or failure information along with an id that correlates to the id of the request object in the batch.

See Successful Batch Requests and Batch Requests with Errors for examples.

Successful Batch Requests

Create Three Directories

Here is a batch request to create three directories using `makeDir`:

```
[
  {"jsonrpc": "2.0", "params": ["467e", "/house1"], "method": "makeDir", "id": 1},
  {"jsonrpc": "2.0", "params": ["467e", "/house2"], "method": "makeDir", "id": 2},
  {"jsonrpc": "2.0", "params": ["467e", "/house3"], "method": "makeDir", "id": 3}
]
```

(For information about `makeDir`, see Create a Directory.)

Here is the response:

```
[
  { u'id': 1, u'jsonrpc': u'2.0', u'result': 0},
  { u'id': 2, u'jsonrpc': u'2.0', u'result': 0},
  { u'id': 3, u'jsonrpc': u'2.0', u'result': 0}
]
```

curl Example

Here is the request to create the same three directories. Note that this example runs on Windows, which requires that inner quotation marks be escaped by adding an extra quotation mark.

```
curl -v^
-X POST^
-H "Content-Type: application/json"^
-d "[{""jsonrpc"": ""2.0"", ""params"": [""467e"", ""/house1""], ""method"":
""makeDir"", ""id"": ""1""}, {""jsonrpc"": ""2.0"", ""params"": [""467e"",
""/house2""], ""method"": ""makeDir"", ""id"": ""2""}, {""jsonrpc"": ""2.0"",
""params"": [""467e"", ""/house3""], ""method"": ""makeDir"", ""id"": ""3""}]"^
http://{Account name}.upload.llnw.net:8080/jsonrpc2
```

Here is the response:

```
HTTP/1.1 200 OK
Date: Thu, 07 Jul 2016 18:17:31 GMT
Server Apache is not deny listed
Server: Apache
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: X-Agile-Authorization, X-Content-Type
Access-Control-Allow-Methods: OPTIONS
Transfer-Encoding: chunked
Content-Type: application/json;charset=utf-8
[
  {"jsonrpc": "2.0", "id": "1", "result": 0},
  {"jsonrpc": "2.0", "id": "2", "result": 0},
  {"jsonrpc": "2.0", "id": "3", "result": 0}
]
```

Batch Requests with Errors

Batch Request with a Failed Request Object

Batch with a valid `ping` request and request to create a directory with leading paths that don't exist:

```
[
  {"jsonrpc": "2.0", "params": ["hello"], "method": "ping", "id": "1"},
  {"jsonrpc": "2.0", "params": ["91c", "1/2/3"], "method": "makeDir", "id": "2"}
]
```

Here is the response:

```
[
  { u'id': u'1',
    u'jsonrpc': u'2.0',
    u'result': { u'code': 0, u'operation': u'hello'}
  },
  { u'id': u'2',
    u'jsonrpc': u'2.0',
    u'result': -3
  }
]
```

Too Many Requests in Batch

If you include more than three requests in a batch, the API returns the following JSON object:

```
{
  u'error': { u'code': -32099, u'message': u'Batch Error'},
  u'id': None,
  u'jsonrpc': u'2.0'
}
```

Batch with No Requests

If you don't include any requests, the API returns the following JSON object:

```
{
  u'error': { u'code': -32700, u'message': u'Parse Error'},
  u'id': None,
  u'jsonrpc': u'2.0'
}
```

Signing Requests in the HTTP Interface

You can use hash-based message authentication code (HMAC) to create single use, time-limited signature-based URLs. These URLs enable you to go beyond token-based authentication and put time constraints on your one-time URLs.

Requests That Can Be Signed

All endpoints in the HTTP interface except `/account/login` accept signed requests:

- `/post/directory`
- `/post/file`
- `/post/raw`
- `/multipart/create`
- `/multipart/piece`
- `/multipart/complete`

Signing and Submitting a Request

This section explains the mechanics of signing and sending a request using `/post/raw` as an example. In this example, a file called `testfile.txt` is uploaded to the caller's namespace. To make the example simple, the only `/post/raw`-specific header involved is `X-Agile-Basename`. An `X-Agile-Directory` header is not sent, causing the file to be created under the root directory.

The example assumes you understand the `/post/raw` endpoint. See File Raw Post for additional information.

Please note the following generalizations about signing *all* requests (not just `/post/raw`):

- Before signing a request, you must have previously generated your access key and secret using the `initKeyPair` call. See Initializing HMAC Key Pairs for additional information.
- Part of the process of signing a request involves creating a query string, which must be URL-encoded. For spaces, use **+** rather than **%20**.
- Query string terms must appear in alphabetical order.
- Because your access_key is involved, you do not have to send an `X-Agile-Authorization` header.

Sign the Request

1. Construct the request's query string, adding to it the following:
   - An entry for your access key (from the `initKeyPair` call). Example:`access_key=3e7359107d65869061992`
   - A timestamp after which the request will be invalid. Example:`&expiry=1461084890`
   - Entries for each Edgio-specific HTTP header (and its value) that you will submit with the request. Remove the "X-Agile" prefix from each header name and make the name all lower case. Example: `X-Agile-Basename` with a value of `/testfile.txt` becomes `&basename=testfile.txt`
2. Place the terms in the query string in ascending order by term key.
3. Combine the request's path and query string, then make an hmac SHA256 digest from the combination using your secret key as the key. For example, make the digest out of this:`/post/raw?access_key=3e7359107d65869061992&basename=testfile.txt&expiry=1461084890`
4. Then base64 encode the digest from the previous step.
5. Add the base64 encoded value to the query string in the `signature` term like this:`/post/raw?access_key=asdf&basename=testfile.txt&expiry=1461084890&signature=g97jovWlpjTP0u+1VPfWhQa5GGpdW1hf6oApxQWIgXg=`

Submit the Request

When you submit the request, you must include:

- the `X-Agile-Signature` header whose value will be the entire path + query string combination. See Sample Python Code
- any additional request headers required by the request. Do not pass an `X-Agile-Authorization` header.

Sample Python Code

The following code signs and submits a Post Raw request, but you can use the code as a basis for any other requests that can be signed.

```
import requests
import base64
import hashlib
import hmac
import io
import os
import time
import argparse


def make_hmac_signature(headers, endpoint, expiry, access_key, secret_key):
    mac = [
        "access_key={access_key}".format(access_key=access_key),
        "expiry={expiry}".format(expiry=expiry),
    ]
    for header in headers:
        if header.startswith("X-Agile-"):
            mac.append("{key}={value}".format(key=header[len("X-Agile-"):].lower(),
value=headers[header]))
    mac.sort()
    payload = "{endpoint}?{query_string}".format(endpoint=endpoint,query_
string="&".join(mac))
    raw_signature = hmac.new(secret_key, msg=payload,
digestmod=hashlib.sha256).digest()
    signature = payload + "&signature=" + base64.b64encode(raw_signature)
    return signature


def do_post_raw(local_path, remote_path, access_key, secret_key,
host="api.lama.lldns.net"):
    data_stream = io.open(local_path, "rb")
    headers = {
        'X-Agile-Directory': os.path.dirname(remote_path),
        'X-Agile-Basename': os.path.basename(remote_path),
        'X-Agile-Content-Detect': 'name',
    }
    signature = make_hmac_signature(headers, "/post/raw", int(time.time()) + 10,
access_key, secret_key)
    headers['X-Agile-Signature'] = signature
    post_raw_endpoint = 'https://%s/post/raw' % host
    return requests.post(post_raw_endpoint, data=data_stream, headers=headers,
verify=False)


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--host", default="api.lama.lldns.net")
    parser.add_argument("--local")
    parser.add_argument("--remote")
    parser.add_argument("--access-key")
    parser.add_argument("--secret-key")
    args = parser.parse_args()
    resp = do_post_raw(args.local, args.remote, args.access_key, args.secret_key)
    print (resp.headers)
```

Using UTF-8 Characters in Request Headers

For certain requests (see Requests that Support UTF-8) you can use UTF-8 character encoding in headers that specify paths and file names. When you use UTF-8 character encoding, you must include the `X-Agile-Encoding` header in your requests, like so:

```
X-Agile-Encoding: UTF8
```

The header is optional and encoding defaults to US ASCII.

If you supply an invalid encoding, *Origin Storage* returns a 400 HTTP Status Code with reason -51, invalid coding.

Notes:

- All path name and file name headers also support US ASCII.
- Although the JSON-RPC interface supports UTF-8, the interface uses unicode escaping rather than URL encoding. For example, файлы is `\\u0444\\u0430\\u0439\\u043b\\u044b`.

Requests that Support UTF-8

The following requests support UTF-8 and the `X-Agile-Encoding` header:

Request	Headers in the Request that Support UTF-8
Create multipart (`multipart/create`)	`X-Agile-Directory`, `X-Agile-Basename`
Post Raw (`post/raw`) See File Raw Post.	`X-Agile-Directory`, `X-Agile-Basename` request parameters `X-Agile-Path` response parameter

Notes:

- Create multipart piece (`multipart/piece`) and complete multipart (`multipart/complete`) do not require UTF-8 encoding because they do not allow the caller to pass file names.
- *Origin Storage* path segment names and file names have a limit of 255 bytes. Because UTF-8 is supported, be sure to calculate the actual length before passing paths and file names in headers if you are using other than basic Latin letters, digits, and punctuation signs. See Path Segment and File Name Limitations for additional information.

Request and Response Example

Here is a sample request using UTF-8 encoding:

```
POST /post/raw HTTP/1.1
User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1
zlib/1.2.3.4 libidn/1.23 librtmp/2.3
Host: example.net:8080
Accept: */*
X-Agile-Authorization: d9e89ef8-fe07-4b6a-8cc5-bc03e4d1ce42
X-Agile-Directory: /
X-Agile-Basename: test%20file.txt
X-Agile-Encoding: UTF8
Content-Length: 29
```

```
Content-Type: application/x-www-form-urlencoded
[29 bytes of data]
```

Here is a sample response:

```
HTTP/1.1 200 OK
Date: Tue, 03 Feb 2015 22:40:42 GMT
Server: Apache/2.2.22 (Ubuntu)
X-Agile-Status: 0
Content-Length: 0
X-Agile-Checksum: 6908750bc780002d195e362de56a77ad79bbc36091159032763b4387cdb0fca5
X-Agile-Size: 29
X-Agile-Path: /test%20file.txt
Vary: Accept-Encoding
Content-Type: text/xml;charset=utf-8
```

File Upload End-to-End Examples

This section shows Python code examples that you can use for obtaining a token, uploading a file, and verifying the file was uploaded successfully. Note that rhe verification step is important because it verifies that the file you attempted to upload indeed was uploaded. For example if you attempted to upload a 10MB file and a network error occurred you need to determine if the upload was successful.

You can copy and paste the code into the Python IDE of your choice, or you can use the same concepts to upload files by creating programs in other languages such as Java, JavaScript, and C#.

Information is in the following sections:

- Obtain a Token
- /post/file Example
- /post/raw Example
- Multipart Example

The examples in this section all include `{Account name}` as part of the hostname. Be sure to substitute your Limelight Account name for `{Account name}` before running any sample.

Obtain a Token

You can use either the JSON-RPC interface or HTTP interface.

Note:
Always use https for logging in to prevent sniffer attacks that can detect your credentials and token.

JSON-RPC Example

```
>>> import jsonrpclib
>>> api = jsonrpclib.Server('https://{Account name}.upload.llnw.net/jsonrpc2')
>>> token, uid_gid = api.login('yourUser', 'yourPassword')
>>> print (token)
f3037573-2a6f-4042-ab8f-82d6823b0480
```

For complete details about `login`, see Logging in Using the JSON-RPC Interface.

HTTP Example

```
>>> import requests
>>> auth_headers = { 'X-Agile-Username': 'yourUser', 'X-Agile-Password':
'yourPassword' }
>>> url = 'https://{Account name}.upload.llnw.net/account/login'
>>> r = requests.post(url, headers=auth_headers)
>>> token = r.headers['x-agile-token']
>>> print (token)
7c164371-d581-4068-8ce1-4d17b6a9d8a3
```

For complete details about `account/login`, see Logging in Using the HTTP Interface.

/post/file Example

This section presents a simple example using only the `X-Agile-Authorization` header along with the `uploadFile`, `basename`, and `directory` form fields. Note that `uploadFile` is a multipart/form-data encoded payload so it is handled differently than other form fields.

For complete details about `/post/file`, see Web Browser Upload.

Upload a File

```
>>> import requests
>>> basename = 'file1.txt'
>>> file_to_upload ='/' + basename
>>> upload_headers = { 'X-Agile-Authorization': token}
>>> payload = {'basename': basename, 'directory': '/zz' }
>>> files = { 'uploadFile': open(file_to_upload, 'r') }
>>> r = requests.post('http://{Account name}.upload.llnw.net/post/file',
data=payload, headers=upload_headers, files=files, verify=False)
```

Verify That the Upload Was Successful

You can use HTTP or JSON-RPC.

HTTP Example

Look at the headers returned from the request. The `/post/raw` request returns the upload status in the `X-Agile-Status` header. A value of 0 (zero) means success.

```
>>> print (r.headers
{'x-agile-status': '0', 'content-length': '0', 'access-control-allow-headers': 'X-
Agile-Authorization, X-Content-Type', 'x-agile-checksum':
'e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855', 'keep-alive':
'timeout=5, max=100', 'server': 'Apache', 'x-agile-size': '0', 'connection': 'Keep-
Alive', 'date': 'Mon, 17 Aug 2015 20:23:17 GMT', 'x-agile-path': '/{Account
name}/zz/file1.txt', 'access-control-allow-origin': '*', 'access-control-allow-
methods': 'OPTIONS', 'content-type': 'text/json;charset=utf-8'}
```

For other status codes, see HTTP Response Codes and Request Status Codes.

You can also issue a `HEAD` request against the object and look at the overall response code (200=OK) as well as the `X-Agile-Checksum` and `Content-Length` response headers:

```
>>> r = requests.head('http://global.mt.lldns.net/<Account Name>/zz/file1.txt')
>>> print (r)
<Response [200]>
>>> print (r.headers)
{'x-agile-checksum':
'e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855', 'accept-ranges':
'bytes', 'server': 'nginx/1.6.2', 'last-modified': 'Mon, 17 Aug 2015 20:23:17 GMT',
'connection': 'keep-alive', 'date': 'Mon, 17 Aug 2015 20:33:02 GMT', 'content-type':
'text/plain'}
```

JSON-RPC Example

Use the `stat` function to get information on the uploaded file. The `stat` function has an optional ‘detail’ parameter. Pass `True` for extended information, or pass `False` (or omit) for an abbreviated response. A value of 0 in the returned `code` field indicate success. Other values indicate failure. You can also look at other fields such as `size` and `checksum` and compare them to known values.

```
>>> api.stat(token, 'zz/file1.txt', True)
{u'mimetype': u'text/plain', u'code': 0, u'uid': 12020, u'checksum':
u'e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855', u'gid': 100,
u'mtime': 1439842997, u'size': 0, u'type': 2, u'ctime': 1439842997}
```

For more information about the `stat` call, see Obtain File or Directory Metadata.

Complete HTTP Example

This section ties information together, providing a complete Python work sample that you can quickly and easily copy and paste into a Python file.

Note:
Always use https for logging in to prevent sniffer attacks that can detect your credentials and token.

```
import jsonrpclib
import requests
api = jsonrpclib.Server('https://{Account Name}.upload.llnw.net/jsonrpc')
token, user = api.login('yourUser', 'yourPassword')

basename = 'file1.txt'
file_to_upload ='/' + basename
upload_headers = { 'X-Agile-Authorization': token}
payload = {'basename': basename, 'directory': '/zz' }
files = { 'uploadFile': open(file_to_upload, 'r') }
r = requests.post('http://{Account name}.upload.llnw.net/post/file', data=payload,
headers=upload_headers, files=files, verify=False)
print (r)
print (r.headers)
r2 = requests.head('http://global.mt.lldns.net/{Account Name}/zz/file1.txt')
print (r2)
print (r2.headers)
api.stat(token, '/zz/file1.txt', True)
```

/post/raw Example

This section presents a simple example using only the `X-Agile-Authorization`, `X-Agile-Basename`, and `X-Agile-Directory` headers.

For complete details about /post/raw, see File Raw Post.

Upload a File

```
>>> import requests
>>> file_to_upload = '\test.mp3'
>>> upload_headers = { 'X-Agile-Authorization': token, 'X-Agile-Basename': file_to_
upload, 'X-Agile-Directory': '/' }
>>> with open(file_to_upload, 'rb') as filesrc:
   r = requests.post('http://{Account name}.upload.llnw.net/post/raw', data=filesrc,
headers=upload_headers)
```

Verify That the Upload Was Successful

You can use HTTP or JSON-RPC.

HTTP Example

Look at the headers returned from the request. The `/post/raw` request returns the upload status in the `X-Agile-Status` header. A value of 0 means success.

```
>>> print (r.headers
{'x-agile-status': '0', 'content-length': '0', 'access-control-allow-headers': 'X-
Agile-Authorization, X-Content-Type', 'x-agile-checksum':
'f5d56bded9d953c31d4e257ecf606e152435af1c8139d32a5b367e9070ac783f', 'keep-alive':
'timeout=5, max=100', 'server': 'Apache', 'x-agile-size': '594472', 'connection':
'Keep-Alive', 'date': 'Mon, 17 Aug 2015 17:10:23 GMT', 'x-agile-path': '/{Account
Name}/test.mp3', 'access-control-allow-origin': '*', 'access-control-allow-methods':
'OPTIONS', 'content-type': 'text/json;charset=utf-8'}
```

You can also issue a `HEAD` request against the object and look at the headers returned and the overall response code (200=OK).

Here is an example using the Python `requests` library:

```
>>> r2 = requests.head('http://{Account name}.upload.llnw.net/{Account Name}/' +
basename)
>>> print (r2)
<Response [200]>
>>> print (r2.headers)
{'content-length': '594472', 'x-agile-checksum':
'f5d56bded9d953c31d4e257ecf606e152435af1c8139d32a5b367e9070ac783f', 'accept-ranges':
'bytes', 'server': 'nginx/1.6.2', 'last-modified': 'Mon, 17 Aug 2015 17:10:25 GMT',
'connection': 'keep-alive', 'date': 'Mon, 17 Aug 2015 17:15:07 GMT', 'content-type':
'audio/mpeg'}
```

JSON-RPC Example

Use the `stat` function to get information on the uploaded file. The `stat` function has an optional ‘detail’ parameter. Pass `True` for extended information, or pass `False` (or omit) for an abbreviated response. A value of 0 in the returned `code` field indicate success. Other values indicate failure. You can also look at other fields such as `size` and `checksum` and compare them to known values.

```
>>> api.stat(token, '/test.mp3', True)
```

```
{u'mimetype': u'audio/mpeg', u'code': 0, u'uid': 12020, u'checksum':
u'f5d56bded9d953c31d4e257ecf606e152435af1c8139d32a5b367e9070ac783f', u'gid': 100,
u'mtime': 1439831425, u'size': 594472, u'type': 2, u'ctime': 1439831425}
```

For more information about the `stat` call, see Obtain File or Directory Metadata.

Complete HTTP Example

This section ties information together, providing a complete Python working sample that you can quickly and easily copy and paste into a Python file.

> **Note:**
> Always use https for logging in to prevent sniffer attacks that can detect your credentials and token.

```
import jsonrpclib
import requests
api = jsonrpclib.Server('https://{Account Name}.upload.llnw.net/jsonrpc')
token, user = api.login('yourUser', 'yourPassword')

basename = 'test.mp3'
file_to_upload ='/' + basename
upload_headers = { 'X-Agile-Authorization': token, 'X-Agile-Basename': file_to_
upload, 'X-Agile-Directory': '/' }
with open(file_to_upload, 'rb') as filesrc:
  r = requests.post('http://{Account name}.upload.llnw.net/post/raw', data=filesrc,
headers=upload_headers)
print (r.headers
r2 = requests.head('http://global.mt.lldns.net/{Account Name}/' + basename)
print (r2)
print (r2.headers)
api.stat(token, "/" + basename, True)
```

Multipart Example

This section presents a simple scenario for initiating a multipart upload, adding a piece to the upload, and completing the upload. For complete information about multipart uploads, see Uploading Files – Multipart.

Calculate the SHA-256 Checksum For the File

You can use the value later to ensure the file was uploaded successfully.

Initiate the Upload

```
>>> import requests
>>> serverDirectory = '/multipart'
>>> uploadBasename = 'text-file1.txt'
>>> headers = { 'X-Agile-Authorization': token, 'X-Agile-Basename': uploadBasename,
'X-Agile-Directory': serverDirectory}
>>> r = requests.post('http://{Acct name}.upload.llnw.net/multipart/create',
headers=headers, verify=False)
```

Verify That the Initiation Was Successful

Look at the headers returned from the request. The `/multipart/create` request returns the upload status in the `X-Agile-Status` header. A value of 0 means success.

```
>>> print (r.headers)
{'x-agile-status': '0', 'content-length': '0', 'x-agile-meta': 'content_type=None
mtime=0', 'keep-alive': 'timeout=5, max=100', 'server': 'Apache', 'connection':
'Keep-Alive', 'x-agile-multipart': 'e20207caa6234b53a93f671320ad7be6', 'date': 'Wed,
19 Aug 2015 21:23:41 GMT', 'x-agile-path': '/{Account Name}/multipart/text-
file1.txt', 'access-control-allow-origin': '*', 'access-control-allow-methods':
'OPTIONS', 'content-type': 'text/json;charset=utf-8', 'access-control-allow-
headers': 'X-Agile-Authorization, X-Content-Type'}
```

Note:
A multipart upload is not available until you have added pieces to it and completed it, so a HEAD request or a JSON-RPC stat request on the object at this point will fail.

Add a Piece To the Multipart Upload

This is a simple example that adds a complete file to the multipart upload. In real life you would stream the object, break the stream into chunks and upload each chunk as a piece.

```
>>> mpid = r.headers['X-Agile-Multipart']
>>> pieceBasename = 'file1.txt'
>>> file_to_upload ='/' + pieceBasename
>>> headers = { 'X-Agile-Authorization': token, 'X-Agile-Multipart': mpid, 'X-Agile-
Part': '1'}
>>> with open(file_to_upload, 'rb') as filesrc:
>>> r2 = requests.post('http://{Acct name}.upload.llnw.net/multipart/piece',
data=filesrc, headers=headers, verify=False)
```

Verify That the Piece Was Uploaded Successfully

Look at the headers returned from the request. The upload status is in the `X-Agile-Status` header. A value of 0 means success.

Compare the checksum (`X-Agile-Checksum` header) with the value you calculated earlier.

```
>>> print (r2.headers)
```

```
{'x-agile-status': '0', 'content-length': '0', 'access-control-allow-headers': 'X-Agile-Authorization, X-Content-Type', 'x-agile-checksum': 'c28a4e896970557bea969cc591299183b341d90dad44ace3684994be80e8d07c', 'keep-alive': 'timeout=5, max=100', 'server': 'Apache', 'x-agile-size': '70', 'connection': 'Keep-Alive', 'date': 'Wed, 19 Aug 2015 21:23:46 GMT', 'access-control-allow-origin': '*', 'access-control-allow-methods': 'OPTIONS', 'content-type': 'text/json;charset=utf-8'}
```

Complete the Upload

```
>>> headers = { 'X-Agile-Authorization': token, 'X-Agile-Multipart': mpid}
>>> r3 = requests.post('http://{Acct name}.upload.llnw.net/multipart/complete',
headers=headers, verify=False)
```

Verify That the Upload Was Successful

You can use HTTP or JSON-RPC.

HTTP Example

Look at the `X-Agile-Status` header. A value of 0 means success.

```
>>> print (r3.headers)
{'x-agile-status': '0', 'content-length': '0', 'x-agile-parts': '1', 'keep-alive': 'timeout=5, max=100', 'server': 'Apache', 'connection': 'Keep-Alive', 'X-Agile-Multipart': 'e20207caa6234b53a93f671320ad7be6', 'date': 'Wed, 19 Aug 2015 21:23:50 GMT', 'access-control-allow-origin': '*', 'access-control-allow-methods': 'OPTIONS', 'content-type': 'text/json;charset=utf-8', 'access-control-allow-headers': 'X-Agile-Authorization, X-Content-Type'}
```

You can also issue a `HEAD` request against the object and look at the headers returned and the overall response code (200=OK).

```
>>> r4 = requests.head('http://{Account name}.upload.llnw.net' + serverDirectory +
'/' + uploadBasename)
>>> print (r4)
<Response [200]>
>>> print (r4 headers)
{'content-length': '70', 'x-agile-checksum':
'c28a4e896970557bea969cc591299183b341d90dad44ace3684994be80e8d07c', 'accept-ranges':
'bytes', 'server': 'nginx/1.6.2', 'last-modified': 'Wed, 19 Aug 2015 21:22:44 GMT',
'connection': 'keep-alive', 'date': 'Wed, 19 Aug 2015 21:23:53 GMT', 'content-type':
'text/plain'}
```

Note:
There is often a delay between the time you complete the multipart upload and the time it is ready for a HEAD request. Issuing a HEAD request too soon results in a 404 (not found) error.

JSON-RPC Example

Use the stat function to get information on the uploaded file. The stat function has an optional 'detail' parameter. Pass True for extended information, or pass False (or omit) for an abbreviated response. A value of 0 in the returned code field indicate success. Other values indicate failure. You can also look at other fields such as size and checksum and compare them to known values.

```
>>> api.stat(token, serverDirectory + '/' + uploadBasename, True)
{u'mimetype': u'text/plain', u'code': 0, u'uid': 12020, u'checksum':
u'c28a4e896970557bea969cc591299183b341d90dad44ace3684994be80e8d07c', u'gid': 100,
u'mtime': 1440025478, u'size': 70, u'type': 2, u'ctime': 1440025489}
```

For more information about the stat call, see Obtain File or Directory Metadata.

Complete HTTP Example

This section ties information together, providing a complete Python working sample that you can quickly and easily copy and paste into a Python file.

Note:
Always use https for logging in to prevent sniffer attacks that can detect your credentials and token.

```
import jsonrpclib
import requests
import os

#######################################
# Log in
#######################################
api = jsonrpclib.Server('https://{Account Name}.upload.llnw.net/jsonrpc')
token, user = api.login('yourUser', 'yourPassword')

#######################################
# Set Upload Variables
#######################################
serverDirectory = '/multipart'
localDirectory = '/Temp/'
fileName = 'flat-irons3.jpg'
chunkSize = 200000

print ('==================')
print ('Upload Information')
print ('==================')
print ('Upload file: ' + fileName)
print (' Server Dir: ' + serverDirectory)
print (' Piece Size: ' + str(chunkSize))
print ('')
```

```
#######################################
# Create multipart
#######################################
print ('================')
print ('Multipart Create')
print ('================')

headers = { 'X-Agile-Authorization': token, 'X-Agile-Basename': fileName, 'X-Agile-
Directory': serverDirectory}
r = requests.post('http://{Account Name}.upload.llnw.net/multipart/create',
headers=headers, verify=False)

print ('Request Result')
print ('--------------')
print (str (r.status_code ) + ' (' + r.reason + ')')

print ('')
print ('Multipart ID')
print ('------------')
mpid = r.headers['X-Agile-Multipart']
print (mpid)

#######################################
# Multipart Piece
#######################################
print ('')
print ('===============')
print ('Multipart Piece')
print ('===============')
file_to_upload = localDirectory + fileName
pieceNum = 1
f = open(file_to_upload, 'rb')
while True:
  chunkBuff = f.read( chunkSize )
  if not chunkBuff:
    break
  else:
    headers = { 'X-Agile-Authorization': token, 'X-Agile-Multipart': mpid, 'X-Agile-
Part': str( pieceNum )}
    print (' Sending piece: ' + str( pieceNum ) )
    r2 = requests.post('http://{Account Name}.upload.llnw.net/multipart/piece',
data=chunkBuff, headers=headers, verify=False)
    print (' Result: ' + str (r2.status_code ) + ' (' + r2.reason + ')')
    status = r2.headers['X-Agile-Status']
    print ('X-Agile-Status: ' + str( status ))
    print ('-------------------------')
    if status != "0":
       break
    else:
      pieceNum = pieceNum +1
f.close()

#######################################
# Multipart Complete
#######################################
print ('')
print ('==================')
print ('Multipart Complete')
print ('==================')
headers = { 'X-Agile-Authorization': token, 'X-Agile-Multipart': mpid}
r3 = requests.post('http://{Account Name}.upload.llnw.net/multipart/complete',
headers=headers, verify=False)
print ('Request Result')
print ('---------------')
print (str (r3.status_code ) + ' (' + r3.reason + ')')
print ('X-Agile-Status: ' + r3.headers['x-agile-status'])

#######################################
# Sleep
#######################################
from time import sleep
secs = 5
print ('')
print ('**sleeping for ' + str(secs) + ' seconds...')
sleep( secs )

#######################################
# HEAD Request
#######################################
print ('==================')
print ('HEAD Request')
print ('==================')
print ('HEAD request on ' + serverDirectory + '/' + fileName)
r4 = requests.head('http://{Account name}.upload.llnw.net' + serverDirectory + '/' +
fileName)
print ('Request Result')
print ('---------------')
print (str (r4.status_code ) + ' (' + r4.reason + ')')
print ('')
print ('Checksum')
print ('---------')
print (r4.headers['x-agile-checksum'])
print ('')
print ('Content Length')
print ('---------------')
print (r4.headers['content-length'])
```

```
#######################################
# JSON-RPC stat Request
#######################################
print ('')
print ('=====================')
print ('JSON-RPC stat Request')
print ('=====================')
print ('Requesting stat on ' + serverDirectory + '/' + fileName)
results = api.stat(token, serverDirectory + '/' + fileName, True))
print ('')
print ('Request Result')
print ('--------------')
print (str (results['code'] ))
statChecksum = results['checksum']
print ('')
print ('Checksum')
print ('--------')
print (statChecksum)
print ('')
print ('File Size')
print ('---------')
print (results['size'])

#########################################
# Get the SHA-256 hash of the local file
#########################################
import hashlib
fh = None
try:
  fh = open(file_to_upload, 'rb')
except IOError, e:
  log.warn("checksum_file IOError: %s: %s" % (file_to_upload, e))
  fh = None
if fh != None:
  m = hashlib.sha256()
  while True:
    buf = fh.read(4096)
    if not buf:
      break
    m.update(buf)
  fh.close()
  localFileCkSum = m.hexdigest()
  print ('')
  print ('==========')
  print ('Local file')
  print ('==========')
  print ('Checksum')
  print ('--------')
  print (localFileCkSum)

fileSize = os.path.getsize(file_to_upload)
print ('')
print ('File Size')
print ('---------')
print (fileSize)
print ('')
if statChecksum == localFileCkSum:
  print ('Local file checksum matches checksum from stat call.')
else:
  print ('Local file checksum does not match checksum returned from stat call.')

########
#Log out
########
api.logout(token)
```

Global Error Codes

The following error codes, not specific to any method, sometimes occur while you are using the *Origin Storage* APIs.

Error Number	Error Classification	Description
-10001	Limellight-specific	Invalid authentication credentials
-32000	Limellight-specific	Unspecified server error. Limellight-specific error
-32099	Limellight-specific	Batch error. Limellight-specific error
-32600	JSON Org predefined	Invalid request. Occurs in any of the following conditions: A JSON-RPC 2.0 request did not contain `2.0` in the jsonrpc member of the `Request` object. A JSON-RPC 2.0 request did not have a `method` member in the `Request` object.
-32601	JSON Org predefined	Method not found. Occurs in any of the following conditions: A JSON-RPC 2.0 request included a method name that started with '_' A request contained a method that is not part of the *Origin Storage* API set.
-32602	JSON Org predefined	Bad parameters A request contained invalid parameters
-32603	JSON Org predefined	Unspecified internal error. Sometimes returned when you pass invalid parameters to a JSON-RPC call.
-32700	JSON Org predefined	Parsing error

Content Types

Origin Storage supports the following content types:

MIME Type	Extension(s)
application/atom+xml	atom
application/java-archive	jar war ear
application/json	json
application/mac-binhex40	hqx
application/msword	doc
application/octet-stream	bin exe dll
application/octet-stream	deb
application/octet-stream	dmg
application/octet-stream	eot
application/octet-stream	iso img
application/octet-stream	msi msp msm
application/ogg	ogx
application/pdf	pdf
application/postscript	ps eps ai
application/rtf	rtf
application/vnd.google-earth.kml+xml	kml
application/vnd.google-earth.kmz	kmz
application/vnd.ms-excel	xls
application/vnd.ms-powerpoint	ppt
application/vnd.wap.wmlc	wmlc
application/x-7z-compressed	7z
application/x-cocoa	cco
application/xhtml+xml	xhtml
application/x-java-archive-diff	jardiff
application/x-java-jnlp-file	jnlp
application/x-javascript	js
application/x-makeself	run
application/x-perl	pl pm

MIME Type	Extension(s)
application/x-pilot	prc pdb
application/x-rar-compressed	rar
application/x-redhat-package-manager	rpm
application/x-sea	sea
application/x-stuffit	sit
application/x-tcl	tcl tk
application/x-x509-ca-cert	der pem crt
application/x-xpinstall	xpi
application/zip	zip
audio/midi	mid midi kar
audio/mpeg	mpga mpega mp2 mp3 m4a
audio/ogg	oga ogg spx
audio/webm	weba
audio/x-realaudio	ra
image/gif	gif
image/jpeg	jpeg jpg
image/png	png
image/svg+xml	svg svgz
image/tiff	tif tiff
image/vnd.wap.wbmp	wbmp
image/x-icon	ico
image/x-jng	jng
image/x-ms-bmp	bmp
text/css	css
text/html	html htm shtml
text/mathml	mml
text/plain	txt
text/vnd.sun.j2me.app-descriptor	jad
text/vnd.wap.wml	wml
text/x-component	htc

MIME Type	Extension(s)
text/xml	xml rss
video/3gpp	3gpp 3gp
video/mp4	mp4
video/mpeg	mpeg mpg mpe
video/ogg	ogv
video/quicktime	mov
video/webm	webm
video/x-flv	flv
video/x-mng	mng
video/x-ms-asf	asx asf
video/x-msvideo	avi
video/x-ms-wmv	wmv

Multipart State Codes

Both multipart uploads and multipart pieces have lifecycles along with states that indicate where they are within their lifecycle. Each state has an associated integer code and a status string.

State codes for mutlpart uploads are visible in the output from calls to `getMultipartStatus` and `listMultipart` (see Get Status for a Multipart Upload and List Your Multipart Uploads).

State codes for mulipart pieces are visible in the output from calls to `listMultipartPiece` (see List Pieces in a Multipart Upload).

The following sections describe the state codes:

- Multipart Upload State Codes
- Multipart Piece State Codes

Note:
You can programmatically obtain the status strings by calling `getMultipartStatusMap` (see Get String Equivalents of Multipart Status Codes). This makes it easy to obtain status strings at runtime.

Note:
Allowable state transitions are listed in Multipart State Transitions.

Multipart Upload State Codes

The following table shows the status string and description for each code.

Code	Status String	Description
0	UNKOWN	Impossible state
1	NEW	Just created and contains no pieces
2	READY	One or more pieces have been added. Ready to be completed.
3	COMPLETE	Multipart upload has been completed
5	JOIN	Join in progress. (The pieces are being joined in order to create the final file.)
6	SUCCESS	Join completed and the final file has been created on disk.
8	DELETED	Pieces have been deleted from all Landing Pads
9	ERROR	An internal error occurred
10	ABORT	Aborted by user
11	EXPIRED	When a multipart upload that is in NEW or FAILED state reaches an age older than a defined amount of time, the upload is transitioned to EXPIRED state if all associated pieces are also in EXPIRED state. The age limit defaults to 1 day. Contact your Account Manager if

Code	Status String	Description
		you want to disable the expiration of aged multipart uploads, or if you want to set the expiry to other than the default.
12	FAILED	An error occurred during the merge process, such as a 404 or a checksum error.

Multipart Piece State Codes

The following table shows the status string and description for each code.

Code	Status String	Description
0	UNKOWN	Impossible state
1	NEW	Piece has been added to a multipart upload.
5	JOIN	Piece is being joined (merged) to a multipart upload.
6	SUCCESS	Piece successfully joined
8	DELETED	Piece successfully deleted from all Landing Pads
9	ERROR	An internal error occurred
10	SKIPPED	A piece enters SKIPPED state if it is located outside of the local domains on which LLPs run.
11	ABORT	Aborted by user
12	EXPIRED	When a multipart piece that is in NEW or FAILED state reaches an age older than a defined amount of time, the piece is automatically transitioned to EXPIRED state. The age limit defaults to 1 day. Contact your Account Manager if you want to disable the automatic expiration of aged multipart uploads, or if you want to set the expiry to other than the default.
13	FAILED	Piece error; example: piece not found

Multipart State Transitions

Both multipart uploads and multipart pieces follow well-defined state transitions. See the following sections:

Multipart State Transitions

Multipart Piece State Transitions

Multipart State Transitions

Following are the possible state transitions:

NEW -> ABORT

NEW -> DELETED

NEW -> READY

NEW -> EXPIRED

READY -> ABORT

READY -> COMPLETE

COMPLETE -> JOIN

JOIN -> FAILED

JOIN -> SUCCESS

SUCCESS -> DELETED

ABORT -> DELETED

FAILED -> DELETED

FAILED -> EXPIRED

EXPIRED (*goes directly to EXPIRED state*)

Multipart Piece State Transitions

Following are the possible state transitions:

NEW -> JOIN

NEW -> ERROR

NEW -> ABORT

NEW -> EXPIRED

NEW -> SKIPPED

ABORT -> SKIPPED

ABORT -> DELETED

ABORT -> ERROR

SUCCESS -> SKIPPED

SUCCESS -> DELETED

SUCCESS -> ERROR

JOIN -> SUCCESS

JOIN -> FAILED

FAILED -> ERROR

FAILED -> EXPIRED

FAILED -> SKIPPED

EXPIRED (*goes directly to EXPIRED state*)

SKIPPED (*goes directly to SKIPPED state*)