



# FileCloud Server Version 23.232

## Developer Guide

## Copyright Notice

©2024 CodeLathe Technologies, Inc. dba FileCloud

All rights reserved.

No reproduction without written permission.

While all reasonable care has been taken in the preparation of this document, no liability is accepted by the authors, FileCloud, for any errors, omissions or misstatements it may contain, or for any loss or damage, howsoever occasioned, to any person relying on any statement or omission in this document.

FileCloud

Phone: U.S: +1 (888) 571-6480

Fax: +1 (866) 824-9584

Email: [support@filecloud.com](mailto:support@filecloud.com)

## Table of Contents

Copyright Notice .....	2
FileCloud HTTP API .....	4
HTTP Basics .....	4
How API authentication works .....	4
Tips for using the API .....	5
In this section: .....	5
PHP Sample for FileCloud API .....	7
The sample for FileCloud API "loginguest" , "getfilelist" and "upload" is done using PHP cURL.....	7
PHP Sample For Multiple Chunk Upload .....	10
PHP Sample To Upload File With Custom Form.....	13
Curl/Bash Sample: File Upload and Download .....	16
PowerShell Samples for FileCloud API.....	18
Python Sample for FileCloud API .....	21
The sample for FileCloud API "loginguest" and "upload" is done using the python requests module. ....	21
API Changes.....	23
CSRF Token API changes .....	23
Policy Related API Changes .....	24
FileCloud API - "Hello World" Exercise.....	27
7.1 COMMAND: LOGINGUEST.....	27
7.2 COMMAND: ADMINLOGIN .....	28
FileCloud API - Authentication Exercise.....	31
FileCloud API - Logging Exercise .....	33
FileCloud API - Requirements.....	34
Meet the Requirements .....	34
FileCloud API - XML Responses.....	35

# FileCloud HTTP API

FileCloud HTTP API allows clients such as web browsers and mobile devices to programmatically connect and access FileCloud instances running on your personal computer or any other device.

- FileCloud runs on Apache server on both Windows and Linux Platforms.
- This server handles incoming HTTP calls, transforms them into internal System Messages and sends them to the appropriate internal services.

## HTTP Basics

### HTTP Basics

HTTP is the fundamental protocol of the World Wide Web. HTTP is a connectionless request response protocol, meaning there is no concept of a persistent connection between a series of requests.

## REQUEST AND RESPONSE

An HTTP request is a message sent from the client to the server. The server sends back a response. The request and response might contain content called the body. In addition, the response always contains a numeric response code, which lets us know if the request was successful. It gives more detailed information about what exactly happened (e.g. the cause of failure).

## METHODS

HTTP supports several request methods, which help the server know how to handle the request. The HTTP methods for our purposes are:

GET: Used to retrieve a resource (such as a web page or image) from a URL

POST: Used to send data to a server (such as the content of a form) based on a URL

## HEADERS

Finally, in addition to the main content of the request and response, HTTP allows additional data to be sent in the form of headers. They can be sent with the request to the server and the response from the server, and they can contain arbitrary text data. There are many standard headers, and the connection API contains methods for easily accessing some of the most common ones.

## How API authentication works

### How API authentication works

FileCloud records all API calls in the audit log, therefore, authentication cannot be done with an API token.

When you log in with `loginguest` or `adminlogin` and provide the correct password, the server creates a new session, and returns the authentication parameters in cookies which are included in all subsequent calls. The cookies allow the server to link your API call to the existing session.

More details about cookies can be seen in these examples:

[Curl/Bash Sample: File Upload and Download](#) - cookie is stored in the file `${USER}_cookie.txt`

[PowerShell Sample for FileCloud API](#) - cookie is stored in the variable `$LoginResponse.Headers['Set-Cookie']`

## Tips for using the API

### ✓ Tips for using the API

#### To view full API documentation:

FileCloud Admin API:

- Swagger: <https://fcapi-admin.filecloud.com/>

FileCloud User API:

- Swagger: <https://fcapi.filecloud.com/>

**Note:** Although the API documentation allows you to view API command formats for previous versions of FileCloud, only the API formats shown for the latest released version of FileCloud are supported.

#### To see which API call and parameters were executed:

1. Execute the action in the FileCloud admin portal or user portal.
2. In the browser, open the Developer Tools using CTRL-Shift-i.
3. Check the Developer Tools to see the API call and parameters.

For example:

The screenshot shows the FileCloud Admin Portal's 'Server Settings' page. The 'Session Timeout (Minutes)' field is highlighted with a red box and contains the value '789'. Below the field, there is a note: 'Specify user web login session timeout. Example: 15 = Default timeout of 15 minutes, 30 = 30 minutes, 60 = 1 hour. Note: Session will always expire when browser is closed unless advanced configuration is...'. A 'Save' button is visible, with a message 'You have unsaved changes.' above it.

Overlaid on the right is the Chrome Developer Tools Network tab. A request for 'loader.gif' is selected. The 'Payload' sub-tab is active, showing the request body. Under 'Form Data', the 'op' is 'setconfigsetting' and the 'params' object contains:
 

```
count: 1
param0: TONIDOCLOUD_SESSION_TIMEOUT_IN_MINUTES
value0: 789
```

 The 'value0: 789' is highlighted with a red box. An orange arrow points from the 'Save' button in the admin portal to the 'value0' field in the developer tools.

## In this section:

- [PHP Sample for FileCloud API](#)
- [PHP Sample For Multiple Chunk Upload](#)

- [PHP Sample To Upload File With Custom Form](#)
- [Curl/Bash Sample: File Upload and Download](#)
- [PowerShell Samples for FileCloud API](#)
- [Python Sample for FileCloud API](#)
- [API Changes](#)
- [FileCloud API - "Hello World" Exercise](#)
- [FileCloud API - Authentication Exercise](#)
- [FileCloud API - Logging Exercise](#)
- [FileCloud API - Requirements](#)
- [FileCloud API - XML Responses](#)

## PHP Sample for FileCloud API

**⚠ Before you try this code, make sure you have installed PHP cURL,**

The sample for FileCloud API "loginquest", "getfilelist" and "upload" is done using PHP cURL

```
<?php

function getCurlValue($file, $mimetype = '') {
    // PHP 5.5 introduced a CurlFile object that deprecates the old @filename syntax
    // See: https://wiki.php.net/rfc/curl-file-upload
    if (function_exists('curl_file_create')) {
        return curl_file_create(realpath($file), $mimetype, $file);
    }

    // Use the old style if using an older version of PHP
    $value = '@' . realpath($file);

    return $value;
}

$cookie_jar = tempnam('/tmp', 'cookie');

// OPTIONS
$username = "USER";
$password = "PASSWORD";
$serverurl = "http://URL";
$pathval = "/SHARED/fcteamfolder/HR/Policies";
$filename = 'helloworld.txt'; // name of the file to upload
$filefullpath = 'helloworld.txt'; // Full path to file to upload

//assign post data
$params = array('userid' => $username, 'password' => $password);

$api = "loginquest";
$url= $serverurl."/core/".$api;

$ch = curl_init($url);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS,$params);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_COOKIEJAR, $cookie_jar);
curl_setopt($ch, CURLOPT_COOKIEFILE, $cookie_jar);
$data = curl_exec($ch);
curl_close($ch);
```

```

$xmlstr = new \SimpleXMLElement($data);
$result = $xmlstr->command->result;
if($result == 1)
{
    echo "Profile Logged in Successfully";

    // getfilelist api call
    $api = "getfilelist";
    $path = $pathval;

    //assign post data
    $param = array('path' =>$path);

    $url= $serverurl."/core/".$api;
    $ch = curl_init($url);
    curl_setopt($ch, CURLOPT_POST, 1);
    curl_setopt($ch, CURLOPT_POSTFIELDS, $param);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
    curl_setopt($ch, CURLOPT_COOKIEJAR, $cookie_jar);
    curl_setopt($ch, CURLOPT_COOKIEFILE, $cookie_jar);
    $data = curl_exec($ch);
    curl_close($ch);

    $xmlstr = new \SimpleXMLElement($data);
    $total = $xmlstr->meta->total;
    if($total == 0)
    {
        echo "<br>". "No of file's in the system";
    }
    else
    {
        echo "<br>". "Total no. of files:". $total;
        $count = 0;
        foreach ($xmlstr->entry as $file)
        {
            echo "<br>". ++$count. ". ". $file->name;
        }
    }

    //upload api call
    $api = "upload";
    $appnamevalue = "explorer";
    $pathvalue = $pathval;
    $offset = '0';
    $complete = '1';

    $cfile = getCurlValue($filefullpath);
    $post = array('file_contents' => $cfile);

    $url= $serverurl."/core/upload?appname=explorer" . $appnamevalue . '&path=' .
    $pathvalue . '&offset=0&complete=' . $complete . '&filename=' . $filename;

```



```
$ch = curl_init($url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, $post);
curl_setopt($ch, CURLOPT_COOKIEJAR, $cookie_jar);
curl_setopt($ch, CURLOPT_COOKIEFILE, $cookie_jar);
$data = curl_exec($ch);
curl_close($ch);

if($data = "OK")
{
    echo "</br>". "File uploaded Successfully";
}
else
{
    echo $data;
}
}
else
{
    echo "Error:". $xmlstr->command->message;
}
?>
```

## PHP Sample For Multiple Chunk Upload



- This sample does multi-chunked uploading.
- Before you try this code, make sure you have installed PHP cURL,
- This sample uses FileCloud APIs "logginguest" and "upload".
- This code can be used to upload large files, splitting them into 20MB chunks.
- The following sample reads a file named sample.mp4 and uploads it into FileCloud as sample.mp4

```
<?php

function create_upload_form_for_datachunk($data) {
$headers = array();
$form[] = implode("\r\n", array(
    "Content-Disposition: form-data; name=\"filedata\"; filename=\"blob\"",
    "Content-Type: application/octet-stream",
    ""
    $data
));

// generate safe boundary
do {
    $boundary = "-----" . md5(mt_rand() . microtime());
} while (preg_grep("/{ $boundary }/", $form));

// add boundary for each parameters
array_walk($form, function (&$part) use ($boundary) {
    $part = "--{ $boundary }\r\n{ $part}";
});

// add final boundary
$form[] = "--{ $boundary }--";
$form[] = "";

// set options
$headers[] = "Content-Type: multipart/form-data; boundary={ $boundary}";
return array($headers, $form);
}

$cookie_jar = tempnam('C:\\testfolder', 'cookie');

// Input Options
$username = "testuser";
$password = "password123";
$serverurl = "https://127.0.0.1";
$pathval = "/testuser/folder";
$filename = 'sample.mp4'; // name of the file to upload
$filefullpath = 'C:\\testfolder\\'.$filename; // Full path to file to upload
//assign post data
```

```

$params = array('userid' => $username, 'password' => $password);

//login to the site
$api = "login";
$url = $serverurl . "/core/" . $api;
$ch = curl_init($url);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, $params);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_COOKIEJAR, $cookie_jar);
curl_setopt($ch, CURLOPT_COOKIEFILE, $cookie_jar);
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
$data = curl_exec($ch);
curl_close($ch);
//end of login

$xmlstr = new \SimpleXMLElement($data);
$result = $xmlstr->command->result;
if ($result == 1) {
    echo "Profile Logged in Successfully";

    //upload api call
    $api = "upload";
    $appnamevalue = "explorer";
    $pathvalue = $pathval;

    //creating a single shot upload call
    $offset = 0;
    $complete = '0';
    $file_sz = filesize($filefullpath);
    $chunk_sz = 20000000; //20MB chunks

    while($offset < $file_sz) {
        if(($offset + $chunk_sz) >= $file_sz) {
            $complete = '1';
            $sz = $file_sz - $offset;
        } else {
            $sz = $chunk_sz;
        }

        $data = file_get_contents($filefullpath, FALSE, NULL, $offset, $sz);
        $tt = strlen($data);
        list($headers, $form) = create_upload_form_for_datachunk($data);

        $url= $serverurl."/core/upload?appname=explorer" . '&path=' . $pathval .
        '&offset=' . $offset . '&complete='
            . $complete . '&filename=' . $filename;

        $ch = curl_init($url);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

```

```
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, implode("\r\n", $form));
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
curl_setopt($ch, CURLOPT_COOKIEJAR, $cookie_jar);
curl_setopt($ch, CURLOPT_COOKIEFILE, $cookie_jar);
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
$result = curl_exec($ch);
curl_close($ch);

if ($result != "OK") {
    echo "Failed to upload\n";
    exit;
}

$offset = $offset + $chunk_sz;
}

if ($result == "OK") {
    echo "</br>" . "File uploaded Successfully";
} else {
    echo $result;
}
} else {
    echo "Error:" . $xmlstr->command->message;
}
}
```

## PHP Sample To Upload File With Custom Form



- Before you try this code, make sure you have installed PHP cURL,
- This sample uses FileCloud APIs "logginguest" and "upload"
- This code creates custom upload form doing a single shot upload
- This can be modified for multiple uploads as well
- The code can upload files with same name as input or a custom name.
- The following sample reads a file named helloworld.txt and uploads into FileCloud as hellothere.txt

```
<?php

function create_upload_form_for_datachunk($data) {
    $headers = array();
    $form[] = implode("\r\n", array(
        "Content-Disposition: form-data; name=\"filedata\"; filename=\"blob\"",
        "Content-Type: application/octet-stream",
        "",
        $data,
        "blob"
    ));

    // generate safe boundary
    do {
        $boundary = "-----" . md5(mt_rand() . microtime());
    } while (preg_grep("/{ $boundary }/", $form));

    // add boundary for each parameters
    array_walk($form, function (&$part) use ($boundary) {
        $part = "--{ $boundary }\r\n{ $part}";
    });

    // add final boundary
    $form[] = "--{ $boundary }--";
    $form[] = "";

    // set options
    $headers[] = "Content-Type: multipart/form-data; boundary={ $boundary}";
    return array($headers, $form);
}

$cookie_jar = tempnam('C:\\testfolder', 'cookie');

// Input Options
$username = "testuser";
$password = "password123";
$serverurl = "http://127.0.0.1";
$pathval = "/testuser/folder";
```

```

$filename = 'hellothere.txt'; // name of the file to upload
$filefullpath = 'C:\\testfolder\\helloworld.txt'; // Full path to file to upload
//assign post data
$params = array('userid' => $username, 'password' => $password);

//login to the site
$api = "login";
$url = $serverurl . "/core/" . $api;
$ch = curl_init($url);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, $params);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_COOKIEJAR, $cookie_jar);
curl_setopt($ch, CURLOPT_COOKIEFILE, $cookie_jar);
$data = curl_exec($ch);
curl_close($ch);
//end of login

$xmlstr = new \SimpleXMLElement($data);
$result = $xmlstr->command->result;
if ($result == 1) {
    echo "Profile Logged in Successfully";

    //upload api call
    $api = "upload";
    $appnamevalue = "explorer";
    $pathvalue = $pathval;

    //creating a single shot upload call
    $offset = '0';
    $complete = '1';
    $filedata = file_get_contents($filefullpath);
    list($headers, $form) = create_upload_form_for_datachunk($filedata);

    //make the upload call
    $url = $serverurl . "/core/upload?appname=explorer" . $appnamevalue . '&path=' .
    $pathvalue . '&offset=0&complete=' . $complete . '&filename=' . $filename;
    $ch = curl_init($url);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
    curl_setopt($ch, CURLOPT_POST, 1);
    curl_setopt($ch, CURLOPT_POSTFIELDS, implode("\r\n", $form));
    curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
    curl_setopt($ch, CURLOPT_COOKIEJAR, $cookie_jar);
    curl_setopt($ch, CURLOPT_COOKIEFILE, $cookie_jar);
    $result = curl_exec($ch);
    curl_close($ch);
    //end of upload call

    if ($result == "OK") {
        echo "<br>" . "File uploaded Successfully";
    } else {

```

```
        echo $result;
    }
} else {
    echo "Error:" . $xmlstr->command->message;
}
```

## Curl/Bash Sample: File Upload and Download

This example shows how to use the FileCloud API with Curl/Bash to create a folder and upload a file.

```
#!/bin/bash
# Note: this is an example script, it does not have any error handling.

URL='https://filecloud.example.com'
USER=user1
PASSWORD=password4user

cookieFile=${USER}_cookie.txt
timestamp=`date +%Y-%m-%d_%H-%M-%S`

# user login
#-----

echo ">>> loggin in as user $USER"

endpoint='/core/loginguest'
urlParams='time=`date +%s`'          # optional parameter, simplifies troubleshooting in
the server log

postData=""
postData+='userid='$USER
postData+='&password='$PASSWORD

curl "${URL}${endpoint}?$urlParams" --cookie-jar $cookieFile --data-raw "$postData" --
compressed
echo
echo "<<<<"

# create folder
#-----

# set a name for the new folder
FOLDER="newExampleFolder_`timestamp`"

echo ">>> creating folder $FOLDER for $USER:"

endpoint='/core/createfolder'
urlParams='time=`date +%s`'          # optional parameter, simplifies troubleshooting in
the server log

postData=""
postData+="name=$FOLDER"
postData+="&path=/$USER"

curl "${URL}${endpoint}?$urlParams" --cookie $cookieFile --data-raw "$postData" --
compressed
```



```

echo
echo "<<<<"

# upload file to the new folder
#-----

UPLOADFILE="newExampleFile_`date +%s`.txt"
UPLOADPATH="/${USER}/${FOLDER}"

# create a sample file for upload
date > $UPLOADFILE

echo ">>> upload file $UPLOADFILE to folder $FOLDER for $USER"

endpoint='/upload'

urlParams='time=`date +%s`'          # optional parameter, simplifies troubleshooting in
the server log
urlParams+='&appname=explorer'
urlParams+='&path=$UPLOADPATH'
urlParams+='&offset=0'
urlParams+='&complete=1'
urlParams+='&filename=$UPLOADFILE'

curl -k -X POST -F 'image=@$UPLOADFILE "${URL}${endpoint}?$urlParams" --cookie
$cookieFile

# download the file
#-----

filepath="/${USER}/${FOLDER}/${UPLOADFILE}"
filename=$UPLOADFILE

echo ">>> downloading file $filepath"

endpoint='/core/downloadfile'
urlParams='time=`date +%s`'          # optional parameter, simplifies troubleshooting in
the server log

postData=""
postData+="filepath=$filepath"
postData+="&filename=$filename"

curl "${URL}${endpoint}?$urlParams" --cookie $cookieFile --data-raw "$postData" --
output ${filename}.downloaded
echo
echo "<<<<"

```

## PowerShell Samples for FileCloud API

### FileCloud User Portal: get file listing

```

$baseUrl = "https://fctest.ddns.net"
$username="user1"
$password="password"

$Uri = $baseUrl + "/core/"

# avoid CSFR checks, see https://www.filecloud.com/supportdocs/cloud/csrf-token-api-
changes-13502114.html
$headers = @{
    "User-Agent"="Powershell"
}

$body = @{ op = 'loginquest'
           userid = $username
           password = $password }

echo ">>> trying to login ..."
$loginResponse = Invoke-WebRequest -Method Post -Uri $Uri -Body $body -SessionVariable
WebSession -Headers $headers

echo "==== login response:
=====
"
$loginResponse
echo
"=====

$op="getfilelist"
$path = "/" + $username
$body = @{
    op = $op
    path = $path
}
echo ">>> calling" $op "..."
$response = Invoke-WebRequest -Method Post -Uri $Uri -WebSession $WebSession -Body
$body -Headers $headers
echo "<<< done."

echo "==== response: =====
"
$response
$response.Content
echo "=====

```

### FileCloud Admin Portal: get license information

```

$baseUrl = "https://fctest.ddns.net"
$adminPassword="password"

$Uri = $baseUrl + "/admin/"

# avoid CSFR checks, see https://www.filecloud.com/supportdocs/cloud/csrf-token-api-
changes-13502114.html
$headers = @{
    "User-Agent"="Powershell"
} `

$Body = @{ op = 'adminlogin'
           adminuser = 'admin'
           adminpassword = $adminPassword }

echo ">>> trying to login as admin..."
$LoginResponse = Invoke-WebRequest -Method Post -Uri $Uri -Body $Body -SessionVariable
WebSession -Headers $headers
echo "<<< done."
echo "==== login response:
=====
$LoginResponse
echo
"=====

$op="getlicense"
$Body = @{ op = $op }
echo ">>> calling" $op "...
$Response = Invoke-WebRequest -Method Post -Uri $Uri -Body $Body -WebSession $WebSession
-Headers $headers
echo "<<< done."
echo "==== response: =====
$Response
$Response.Content
echo
"=====

```

#### FileCloud Admin Portal (superadmin): get site list

```

$baseUrl = "https://fctest.ddns.net"
$superAdminPassword='password'

$Uri = $baseUrl + "/admin/"

# avoid CSFR checks, see https://www.filecloud.com/supportdocs/cloud/csrf-token-api-
changes-13502114.html
$headers = @{
    "User-Agent"="Powershell"
} `

```

```

$Body = @{ op = 'superadminlogin'
           superadminuser   = 'superadmin'
           superadminpassword = $superAdminPassword }

echo ">>> trying to login as superadmin..."
$LoginResponse = Invoke-WebRequest -Method Post -Uri $Uri -Body $Body -SessionVariable
WebSession -Headers $Headers

echo "==== login response:
=====
$LoginResponse
echo
"=====

$op="superadmingetallsites"
$Body = @{ op = $op }
echo ">>> calling" $op "...
$Response = Invoke-WebRequest -Method Post -Uri $Uri -Body $Body -WebSession $WebSession
-Headers $Headers
echo "<<< done."
echo "==== response: =====
$Response
echo $Response.Content
echo
"=====

```

## Python Sample for FileCloud API

**⚠ Before you try this code, make sure you have installed the requests module,**

The sample for FileCloud API "loginquest" and "upload" is done using the python requests module.

```
#!/usr/bin/env python3
import requests

## Path to file to be uploaded
PathToFile="D:\\Development\\python\\filecloud upload\\file_upload.txt"  #To be
defined

## Filecloud headers.
Headers = {'Accept': 'application/json'}

## Filecloud creds
Creds = {'userid': 'username', 'password': 'password'}  #To be defined

##Filecloud server API endpoints
ServerURL='https://filecloud_server_url/'  #To be defined
LoginEndPoint = 'core/loginquest'
UploadEndPoint = 'core/upload'

## Specify user path inside Filecloud.
FilecloudPath="/username"  #To be defined

## Upload API params.
UploadApiParams = {'appname': 'explorer', 'path': FilecloudPath, 'offset': 0}

if __name__ == '__main__':
    s = requests.session()
    FileToUpload={'file': (open(PathToFile,'rb'))}
    LoginCall=s.post(ServerURL+LoginEndPoint, data=Creds, headers=Headers).json()
    if LoginCall['command'][0]['result'] == 1:
        print('Login successfull, processing with File upload ...')
        UploadCall=s.post(ServerURL+UploadEndPoint, params=UploadApiParams, files =
FileToUpload, cookies = s.cookies)
        if UploadCall.text == 'OK':
            print('Upload successfull.')
```

```
    else:  
        print('Upload failed.')
```

```
else:  
    print('login failed.')
```

## API Changes

- [CSRF Token API changes](#)
- [Policy Related API Changes](#)

### CSRF Token API changes

#### INTRODUCTION

**This note on API changes affect all applications that use FileCloud API using a Web Browser user agent.**

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated.

FileCloud 14.0 (planned for release in early January 2017) introduces API changes to add additional anti CSRF measures for all Web browser client API actions.

#### What has Changed?

- 1) When a user authenticates using a web browser, an additional cookie is now set, which is called X-XSRF-TOKEN. The value of this cookie needs to be passed in as a HTTP Header 'X-XSRF-TOKEN' for all subsequent API calls.
- 2) All API calls using a web browser need to also set the X-Requested-With: XMLHttpRequest HTTP header.

#### How to modify your existing API Setup calls:

If you are using FileCloud API using a Web Browser HTTP agent here is what you have to do,

- Add the 'X-Requested-With: XMLHttpRequest' HTTP header to all calls made via a web browser
- Retrieve all the cookies when doing a login call in admin or user portal of FileCloud
- Get the Value of 'X-XSRF-TOKEN' cookie.
- On successfully getting the value of this token 'X-XSRF-TOKEN', pass this token in the headers section , while making any POST or GET calls in admin or user portal of FileCloud.

**Eg:**

```
$headers = array(
    'X-Requested-With: XMLHttpRequest',
    'X-XSRF-TOKEN: '. $this->xsrftoken,);
curl_setopt($this->curl_handle, CURLOPT_HTTPHEADER, $headers);
```

## Disabling these checks for Backwards Compatibility:

To turn off CSRF security feature, open cloudconfig.php at

Windows Location: XAMPP DIRECTORY/htdocs/config/cloudconfig.php

Linux Location: /var/www/config/cloudconfig.php

and add the following to disable the checks temporarily.

```
define("TONIDO_CLOUD_CSRF_CHECK", 0);
```

## Conclusion:

Adding these checks makes it harder for any CSRF attempts to succeed against FileCloud installations.

## Policy Related API Changes

 This change affects FileCloud versions 17.3 and greater.

## Background

Prior to FileCloud 17.3, many settings and options were either set globally or overridden on a per-user basis. This is hard to manage for hundreds of users in the system and can be error-prone. To help alleviate this issue, 17.3 introduces the concept of [policies](#) that allow administrators a great deal of flexibility in managing a large FileCloud installation. Policies are a collection of rules and settings that apply to a specific user or sets of users. Any number of policies can be created by an administrator and then can be associated to a set of users individually or to set of FileCloud Groups.

## What API is affected?

1. **Quota Settings for User Accounts:** If you are using "size" parameter in [updateuser](#) API to set the allowed user quota for managed storage, it is no longer supported and the API will ignore the setting and log an error message.
2. **System Config Settings:** If you are using the [setconfigsetting](#) API or [updatepolicyforuser](#) API, then following keys can no longer be set using this API and should be managed by the new Policy infrastructure

```
TONIDO_CLOUD_DEFAULT_SIZE_PERUSER
TONIDO_CLOUD_DISABLE_NOTIFICATION
TONIDO_CLOUD_GLOBAL_SHARE_MODE
```



```

TONIDOCLOUD_ENABLE_PRIVACY_SETTINGS
TONIDOCLOUD_ENABLE_2FA
TONIDOCLOUD_2FA_DELIVERY_MODE
TONIDOCLOUD_DISABLE_INVITES
TONIDOCLOUD_DEVICE_LOGIN_CODE_MODE
TONIDOCLOUD_ADMIN_DEVICE_APPROVAL_REQUIRED
TONIDOCLOUD_ENFORCE_SESSIONTIMEOUT_DEVICES
TONIDOCLOUD_CREATE_ACCOUNT_ON_INVITE
TONIDOCLOUD_SET_ACL

TONIDOCLOUD_REQUIRE_PASSCODE_MOBILECLIENTS
TONIDOCLOUD_DISABLE_CONNECT_MOBILECLIENTS
TONIDOCLOUD_DISABLE_PRINT_MOBILECLIENTS
TONIDOCLOUD_DISABLE_DOWNLOAD_MOBILECLIENTS
TONIDOCLOUD_DISABLE_OPENWITH_MOBILECLIENTS
TONIDOCLOUD_DISABLE_SHARE_MOBILECLIENTS
TONIDOCLOUD_DISABLE_FAV_MOBILECLIENTS
TONIDOCLOUD_DISABLE_EDITS_MOBILECLIENTS
TONIDOCLOUD_DISABLE_CONFIG_CHANGES_MOBILECLIENTS
TONIDOCLOUD_APPLY_CONFIG_MOBILECLIENTS

TONIDOCLOUD_NOTIFICATION_DISABLE_ADD
TONIDOCLOUD_NOTIFICATION_DISABLE_UPDATE
TONIDOCLOUD_NOTIFICATION_DISABLE_DELETE
TONIDOCLOUD_NOTIFICATION_DISABLE_DOWNLOAD
TONIDOCLOUD_NOTIFICATION_DISABLE_PREVIEW
TONIDOCLOUD_NOTIFICATION_DISABLE_LOCK
TONIDOCLOUD_NOTIFICATION_DISABLE_SELF_NOTIFICATION

```

## Handling Changes

If you are upgrading to 17.3 and if you are using the above APIs. Please note that you will need to handle it differently.

### **Setting Quota for Specific Users**

If you want to just a specific quota for a user, you should instead

- [create a new policy](#) for that user if policy doesn't exist and attach it to the user
- (or) if policy is already attached to that user, [retrieve that policy](#)
- [set the required quota](#) in that user specific policy

### **Setting Default Quota Globally**

If you want to set the default quota and make that change apply to all users, you should instead

- [Retrieve the current global default policy](#)
- [change the global policy's quota setting](#)

### **Setting specific user level settings**

If you want to set specific user level settings(via `updatepolicyforuser` API) that were managed by the above configuration flags (listed in section above), you should instead

- [create a new policy](#) for that user if policy doesn't exist and attach it to the user

- (or) if policy is already attached to that user, [retrieve that policy](#)
- [set the required setting](#) in that policy

### **Setting Global Settings**

If you want to set the global settings and make that change apply to all users, you should instead

- [Retrieve the current global default policy](#)
- [change the global policy's settings as needed](#)

## FileCloud API - "Hello World" Exercise



"Hello World" for FileCloud is not exactly the program that prints Hello World.

- We will see how to login and authenticate into FileCloud in a default authentication setting.
- These calls are fundamental to understand the XML messages that are used to communicate between the FileCloud server and client.

Authentication is a one step process.

- The *loginquest* command is issued with account name and password.
- When authentication is complete and successful, the FileCloud HTTP server creates a cookie and returns the cookie value to the client through the HTTP header.

### 7.1 COMMAND: LOGINGUEST

loginquest command accepts the account name and password and attempts to login into the profile. The URL follows the standard format as we discussed above.

```
HTTP://HOST:PORT/CORE/OPERATION {? OPTIONAL PARAMS }
```

When applied for loginquest call, the above URL takes the form.

<http://host:port/core/loginquest>

The input parameters (see table below) userid and password must be passed as name/value pairs from client to FileCloud HTTP server as a part of HTML body through the HTTP POST method.

From the FileCloud Client, issue an HTTP POST command with the above URL when your FileCloud Development Instance is running.

When doing HTTP POST please ensure that the content-type is set correctly to "application/x-www-form-urlencoded"

Also, if you are connecting as an agent, additional parameters such as RMC details of the agent might be needed. To avoid that, we recommend spoofing the User-Agent and passing in any web browser User Agent String.

Header Parameter	Value
content-type	application/x-www-form-urlencoded
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36

FileCloud HTTP server will return an XML result in the following format.

```

<commands>
  <command>
    <type>loginguest</type>
    <result>1</result>
    <message> </message>
  </command>
</commands>

```

In the above XML response, RESULT 1 indicates that the profile was logged in successfully. As an additional help, use the FileCloud Logs to follow the communication between your client and FileCloud HTTP server.

#### Input HTTP, Method: POST

PARAMETER	Required	DATA TYPE	DESCRIPTION
userid	Yes	String	account name
password	Yes	String	Clear Text password

#### Output

PARAMETER	Description
type	Name of the call. Possible value is "loginguest"
result	Returns 1 if success, 0 is failed. However, a RESULT 0 indicates the loginguest failed. Most likely cause will be password is not correct. password
message	Returns any error message generated in the call. 8 FILECLOUD

## 7.2 COMMAND: ADMINLOGIN

adminlogin command accepts the admin name and admin password and attempts to login into the admin site. The URL follows the standard format as we discussed above.

```
HTTP://HOST:PORT/ADMIN/?OP=OPERATION { &OPTIONAL PARAMS }
```

When applied for adminlogin call, the above URL takes the form.

<http://host:port/admin/?op=adminlogin>

The input parameters (see table below) op, adminuser and adminpassword must be passed as name/value pairs from client to FileCloud HTTP server as a part of HTML body through the HTTP POST method.

From the FileCloud Client, issue an HTTP POST command with the above URL when your FileCloud Development Instance is running.

When doing HTTP POST please ensure that the content-type is set correctly to "application/x-www-form-urlencoded"

We recommend spoofing the User-Agent and passing in any web browser User Agent String.

Header Parameter	Value
content-type	application/x-www-form-urlencoded
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36

FileCloud HTTP server will return an XML result in the following format.

```
<commands>
  <command>
    <type>adminlogin</type>
    <result>1</result>
    <message> </message>
  </command>
</commands>
```

In the above XML response, RESULT 1 indicates that the profile was logged in successfully. As an additional help, use the FileCloud Logs to follow the communication between your client and FileCloud HTTP server.

#### Input HTTP, Method: POST

PARAMETER	Required	DATA TYPE	DESCRIPTION
op	Yes	String	Operation
adminuser	Yes	String	Admin user name
adminpassword	Yes	String	Clear Text password

**Output**

<b>PARAMETER</b>	<b>Description</b>
type	Name of the call. Possible value is "adminlogin"
result	Returns 1 if success, 0 is failed. However, a RESULT 0 indicates the adminlogin failed.
message	Returns any error message generated in the call.

## FileCloud API - Authentication Exercise



After you have successfully created a new user and logged in to FileCloud User Site, use this exercise to call the `getauthenticationinfo` operation.

- You must use the following URL: `http://www.yourdomain.com/core/getauthenticationinfo` (where `www.yourdomain.com` is your URL)
- For the purpose of this exercise, Authentication Type must be set to DEFAULT → [Enabling Default Authentication](#)

- i** In FileCloud, the first fundamental requirement is the ability to identify the FileCloud user uniquely.
- The first time you install FileCloud Server and log in to the Admin Portal, you will have to → [Create a new FileCloud user account](#).
  - After account creation, you login into the User Portal with that account using the password you created. → [Log in to the User Portal](#)
  - When you login correctly, an authentication cookie is sent back to the browser which is used for all subsequent HTTP calls.
  - The client has to pass this cookie back to the server with all the calls going forward to successfully communicate with the FileCloud HTTP server.
  - Usually, passing of cookies between client and server is handled automatically in the browser.
  - However, if your client does not handle this communication automatically you have to capture and pass this cookie through the HTTP header from the client to the server.

**💡** HTTP (GET/POST) response from FileCloud Server will always be in a XML format. FileCloud web server will return the following XML response.

```
<authenticationinfo>
<info>
<profile>satad</profile>
<displayname>satad</displayname>
<peerid>satad</peerid>
<authenticated>1</authenticated>
<isguestauthenticated>0</isguestauthenticated>
<hash>sha1</hash>
<guesthash>sha1</guesthash>
<guesthashurl/>
<isremote>0</isremote>
<reasoncode>0</reasoncode>
<OS>TONIDO_CLOUD</OS>
<authtype>DEFAULT</authtype>
</info>
</authenticationinfo>
```

Most of the XML elements above are self explanatory, however do not worry if some properties are not clear. We will dive deep as we go through this document. Let us try to analyze the URL some more. The FileCloud HTTP server expects URL to be of a certain form.

```
HTTP://HOST:PORT/CORE/OPERATION {? OPTIONAL PARAMS VIA GET OR POST}
```

The "CORE" in the above URL refers to FileCloud User Site that provides programmable APIs, manages and provides HTTP connectivity. In our first example, we were using `getauthenticationinfo` as the operation that gets the basic authentication detail such as profile id, display name, authentication type etc of the profile logged into FileCloud server. The Optional Parameters can be used to pass parameters from client to server. They can either be passed via HTTP Get URL parameters or as POST parameters.



## FileCloud API - Logging Exercise



FileCloud logging offers a simple but powerful means to follow and debug the communication between your client and FileCloud HTTP server.

- Logs files are stored under the SCRATCH folder under Apache web server root folder.
- To view the log file, you can use note pad or a Log file monitoring tool such as Bare Tail by Bare Metal Soft in Windows. <http://www.baremetalsoft.com/baretail/>.

As an additional activity, repeat the Authentication exercise and follow the FileCloud logs.

➔ [FileCloud Logging](#)

➔ [Authentication Exercise](#)

## FileCloud API - Requirements









If you want to work with the FileCloud API, you will need to set up your test environment.

- You must have an instance of FileCloud Server running on a Windows or Linux server

### Meet the Requirements

Before you can proceed with the practice exercises or develop a solution to work with the FileCloud API, you must complete the items in the Requirements checklist.

	Requirements	For more information
	FileCloud Server license	Request a free trial or buy FileCloud license from <a href="http://www.filecloud.com">http://www.filecloud.com</a>
	Read the release notes	 <a href="#">Release Notes</a>
	Read the upgrade notes if applicable	 <a href="#">FileCloud Server Upgrade Notes</a>
	Install FileCloud Server	 <a href="#">Install FileCloud Server</a>
	Verify you can log in to the FileCloud Admin Portal	 <a href="#">Log in to the Admin Portal</a>
	Verify you can log in to the FileCloud User Portal	 <a href="#">Access the User Portal</a>

## FileCloud API - XML Responses



XML response received from FileCloud HTTP server follows a standard pattern with items, meta and item element.

The following is the standard XML pattern received from FileCloud server.

```
<items>
  <meta>
    <metaelement1>[Meta Element 1 value]
    </metaelement1> <metaelement2>[Meta Element 2 value]
    </metaelement2>
  </meta>
  <item>
    <itemdetail1>[Item detail 1 value]<itemdetail1>
    <itemdetail2>[Item detail 1 value]<itemdetail2>
    <itemdetail3>[Item detail 1 value]<itemdetail3>
  </item>
</items>
```

The items and item element will be renamed appropriately for each XML response received as shown below in the samples. On the other hand, meta element is optional. Following are some of the sample XML response received.

In the loggingest response below, the items and item element are replaced with commands and command element respectively. The command element has type, result and message elements as itemdetail.

```
<commands>
  <command>
    <type>loggingest</type>
    <result>1</result>
    <message> </message>
  </command>
</commands>
```

In the getfilelist response below, the items and item element are replaced with entries and entry element respectively. The entry element has path, dirpath, name, ext, isroot, type, fullfilename, size, modified, favoritelistid, favoriteid, order, fullsize, modifiedepoch elements as itemdetail. Note that, the getfilelist also returns the optional meta element with parentpath and total entry elements.

```
<entries>
  <meta>
    <parentpath>/apptester</parentpath>
    <total>133</total>
    <realpath>/apptester/Docs</realpath>
```

```

    <canupload>1</canupload>
    <isshareable>1</isshareable>
    <candownload>1</candownload>
    <cansetacls>0</cansetacls>
    <showshareoption>0</showshareoption>
    <teamfolder>0</teamfolder>
    <result>1</result>
    <message/>
    <defaultfile/>
  </meta>

  <entry>
    <path>/apptester/Docs/CA</path>
    <dirpath>/apptester/Docs/</dirpath>
    <name>CA</name>
    <ext/>
    <fullsize>0</fullsize>
    <modified>Dec 10, 2017 8:26 AM</modified>
    <type>dir</type>
    <fullfilename>/apptester/Docs/CA</fullfilename>
    <size/>
    <modifiedepoch>1512894360</modifiedepoch>
    <modifiediso>2017-12-10T08:26:00+0000</modifiediso>
    <isroot>0</isroot>
    <isshareable>1</isshareable>
    <issyncable>0</issyncable>
    <isshared/>
    <canrename>1</canrename>
    <showprev>0</showprev>
    <canfavorite>1</canfavorite>
    <canupload>1</canupload>
    <candownload>1</candownload>
    <favoritelistid>0</favoritelistid>
    <favoriteid>0</favoriteid>
    <order>0</order>
    <showquickedit>1</showquickedit>
    <showlockunlock>1</showlockunlock>
    <showshareoption>0</showshareoption>
    <cansetacls>0</cansetacls>
    <locked>0</locked>
  </entry>
</entries>

```