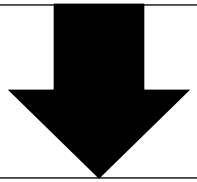
Current Release 1 Implementation – Download Page – Download Functions

Client Browser

Views/Initialize.pug/

include includes/init_rightside_nav.pug

- li: a.init_download(href='auth/start/box1', data-level='1 - Download All') All Photos & Videos
- li: a.init_download(href='auth/start/box2', data-level='2 - Download Photos') All Photos
- li: a.init_download(href='auth/start/box3', data-level='3 - Download Videos') All Videos
- li: a.init_download(href='auth/start/box4', data-level='4 - Update Added Photos & Videos') Just New Photos & Videos
- li: a.init_download(href='auth/start/box5', data-level='5 - Update Added Photos') Just New Photos
- li: a.init_download(href='auth/start/box6', data-level='6 - Update Added Videos') Just New Videos
- li: a.init_download(href='auth/start/box7', data-level='7 - Results of Last Download') Results of Last Download



mySite Node.js Server

public/js/auth.js

router.get("/start/:serviceName", (req, res) => {

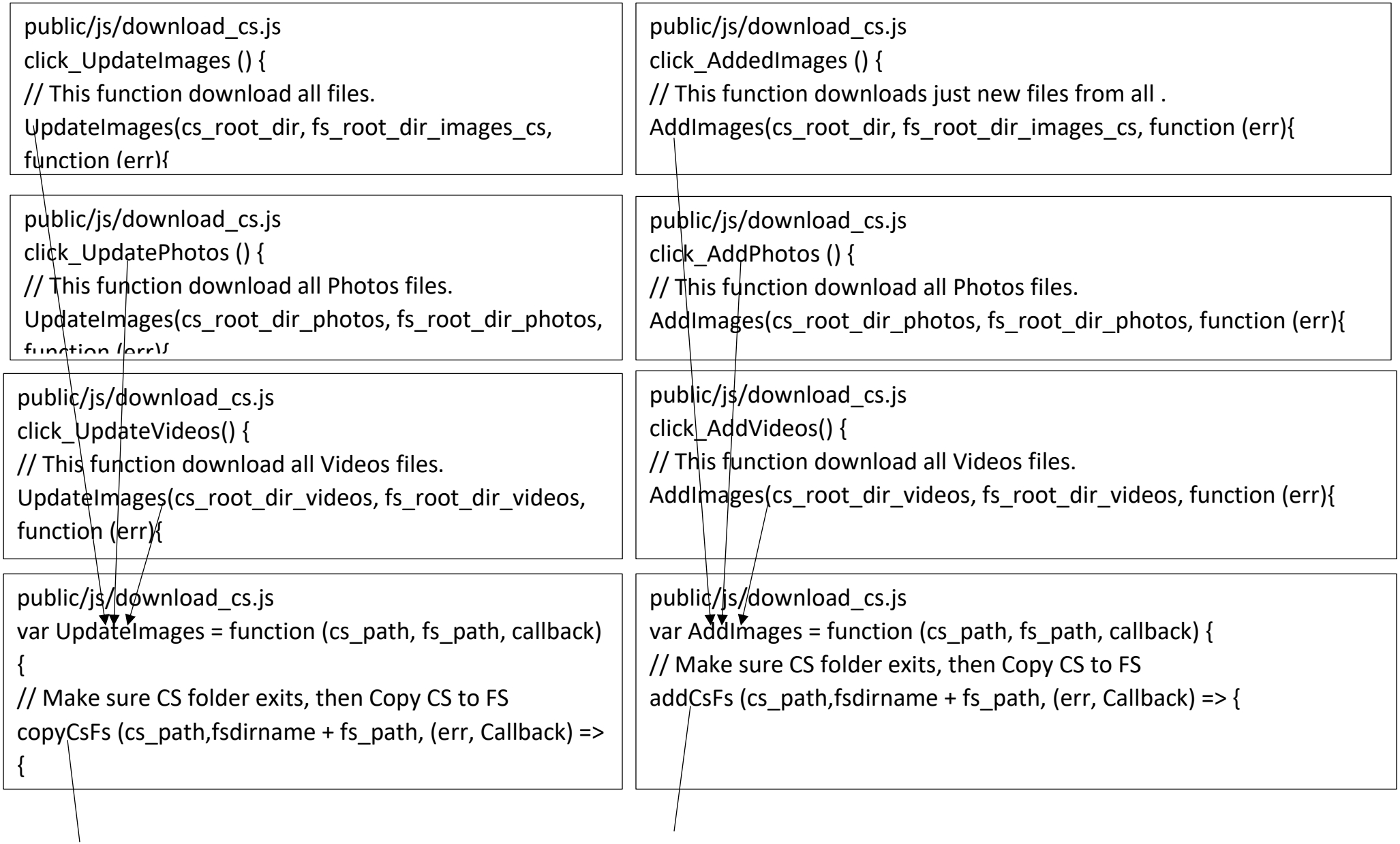
Perform CloudRail OAuth2 Authorization for Cloud Service Selected. User will be asked to enter UserName and Password.

res.redirect('/pleasewait'); // Pass control to pleaseWait.js

routes/pleaseWait.js

The function of this module is to get the Download Function Request from OAuth auth.js module, and call the proper function in the downloas_cs module. The display the pleasewait.pug screen. During the download process, socket.io is used to display progress of the download.

```
    router.get('/', function(req, res, next) {
// At this point, the OAuth2 authorization for CloudRail has completed successfully.
var csSelection = require('../public/js/download_cs');
Switch (data-level)
 1 - csSelection.click_UpdateImages(function (err) {
 2 - csSelection.click_UpdatePhotos(function (err) {
 3 - csSelection.click_UpdateVideos(function (err) {
 4 - csSelection.click_UpdateAddedImages(function (err) {
 5 - csSelection.click_UpdateAddedPhotos(function (err) {
 6 - csSelection.click_UpdateAddedVideos(function (err) {
 7 - csSelection.click_CreateJade(function (err) {
```

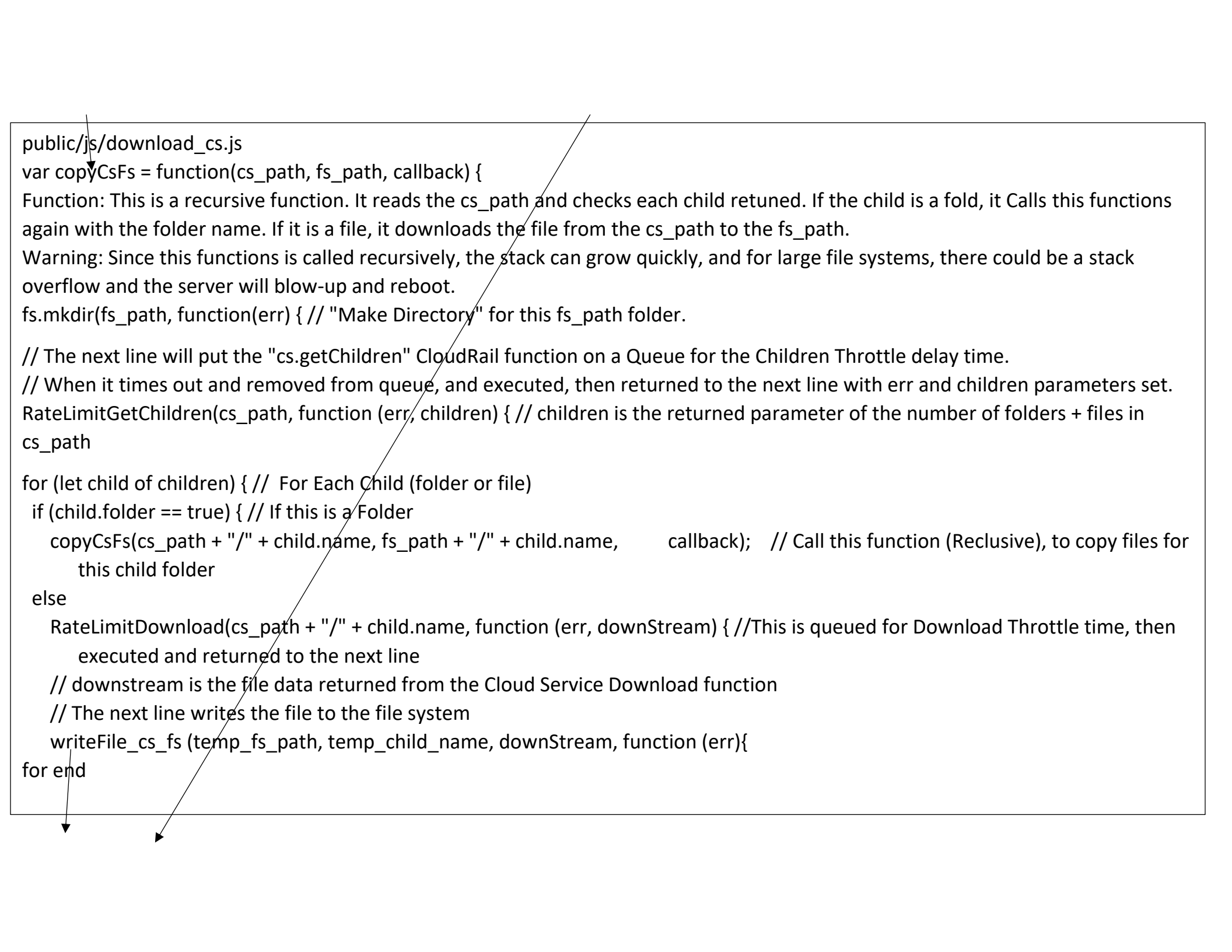

public/js/download_cs.js
click_UpdateImages () {
// This function download all files.
UpdateImages(cs_root_dir, fs_root_dir_images_cs,
function (err){
public/js/download_cs.js
click_AddedImages () {
// This function downloads just new files from all .
AddImages(cs_root_dir, fs_root_dir_images_cs, function (err){
public/js/download_cs.js
click_UpdatePhotos () {
// This function download all Photos files.
UpdateImages(cs_root_dir_photos, fs_root_dir_photos,
public/js/download_cs.js
click_AddPhotos () {
// This function download all Photos files.
AddImages(cs_root_dir_photos, fs_root_dir_photos, function (err){
public/js/download_cs.js
click_UpdateVideos() {
// This function download all Videos files.
UpdateImages(cs_root_dir_videos, fs_root_dir_videos,
function (err){
public/js/download_cs.js
click_AddVideos() {
// This function download all Videos files.
AddImages(cs_root_dir_videos, fs_root_dir_videos, function (err){
public/js/download_cs.js
var UpdateImages = function (cs_path, fs_path, callback)
{
// Make sure CS folder exits, then Copy CS to FS
copyCsFs (cs_path,fsdirname + fs_path, (err, Callback) =>
{
public/js/download_cs.js
var AddImages = function (cs_path, fs_path, callback) {
// Make sure CS folder exits, then Copy CS to FS
addCsFs (cs_path,fsdirname + fs_path, (err, Callback) => {

public/js/download_cs.js

```
var copyCsFs = function(cs_path, fs_path, callback) {
Function: This is a recursive function. It reads the cs_path and checks each child retuned. If the child is a fold, it Calls this functions again with the folder name. If it is a file, it downloads the file from the cs_path to the fs_path.
Warning: Since this functions is called recursively, the stack can grow quickly, and for large file systems, there could be a stack overflow and the server will blow-up and reboot.
fs.mkdir(fs_path, function(err) { // "Make Directory" for this fs_path folder.

// The next line will put the "cs.getChildren" CloudRail function on a Queue for the Children Throttle delay time.
// When it times out and removed from queue, and executed, then returned to the next line with err and children parameters set.
RateLimitGetChildren(cs_path, function (err, children) { // children is the returned parameter of the number of folders + files in cs_path

for (let child of children) { //  For Each Child (folder or file)
  if (child.folder == true) { // If this is a Folder
     copyCsFs(cs_path + "/" + child.name, fs_path + "/" + child.name,         callback);    // Call this function (Reclusive), to copy files for this child folder
  else
     RateLimitDownload(cs_path + "/" + child.name, function (err, downStream) { //This is queued for Download Throttle time, then executed and returned to the next line
     // downstream is the file data returned from the Cloud Service Download function
     // The next line writes the file to the file system
     writeFile_cs_fs (temp_fs_path, temp_child_name, downStream, function (err){
for end
```



```
public/js/download_cs.js
var addCsFs = function(cs_path, fs_path, callback) {
Function: This is a recursive function. It Copies the folder and  files from cs to fs
Warning: Since this function is called recursively, the stack can grow quickly, and for large file systems, there could be a stack
overflow and the server will blow-up and reboot.
// The next line will put the "cs.getChildren" CloudRail function on a Queue for the Children Throttle delay time.
// When it times out and removed from queue, and executed, then returned to the next line with err and children parameters set.
RateLimitGetChildren(cs_path, function (err, children) { // children is the returned parameter of the number of folders + files in
cs_path

for (let child of children) { //  For Each Child (folder or file)
  if (child.folder == true) { // If this is a Folder
     copyCsFs(cs_path + "/" + child.name, fs_path + "/" + child.name,         callback);    // Call this function (Reclusive), to copy files for
          this child folder
  else
     if this file does NOT exit
          RateLimitDownload(cs_path + "/" + child.name, function (err, downStream) { //This is queued for Download Throttle time,
               then executed and returned to the next line
          // downstream is the file data returned from the Cloud Service Download function
          // The next line writes the file to the file system
          writeFile_cs_fs (temp_fs_path, temp_child_name, downStream, function (err){
for end
// Note: There is no “return callback(0)” because off the recursive nature of this function.
// A “return callback(err) occurs if a fatal error occurs that prevents continuing.
// If a CS Download error occurs (probably due to a threshold being reached, it is just logged and ignores. This is not an ideal
situation. Ideally, I would like to determine the exact cause and reissue Download Request again, after another throttle delay.
```

public/js/download_cs.js
function writeFile_cs_fs(fspath, childname, downStream, callback) {
Function: Write the file to the file system. Keep track of the total number of bytes written.
Warning: The way I know the complete download operation has completed is by sampling that the throttle queue is empty and where is not more writing to the file system

```
let fileStream = fs.createWriteStream(fspath + "/" + childname); // convert downSteam to filestream
downStream.pipe(fileStream);  // Write file to file system
downStream.on('end', function () {  // wait for it to complete
downStream.on('data', function (chunk) {
      bytesWrittenToFS = bytesWrittenToFS + chunk.length; // track length written for a file
```