

Overview

This document explains python code which was written to download many files from an ftp directory. It is not a comprehensive piece of code and while it could be used as-is in some cases, it may require modification to work in your specific use case, particularly in relation to comprehensive error handling. The purpose of this document is to provide an example for how to use the “ftplib” python library and potentially speed up writing of your code by providing a well explained reference.

Code Explanation

At first, an example is show about how to use the code. Then, an explanation of each class function is given while noting the key points about each.

Usage

At first, the “ftpMirror” library is imported. The ftp URL, source directory, username, password, and destination directory must be provided. Providing an initial sub folder in the source directory definition can help to limit the size of the file download if there are other subfolders under the top folder which you do not want to download. The source directory string must begin and end with “/”.

An instance of ftpMirror is created called “ftp” by passing the 5 parameters in the order shown. Optionally, a list of file extensions to be excluded can be defined. If those type of files exist, they will be excluded in any file quantity and size computations and will not be downloaded. Finally, the file download is initiated by calling [ftp.download\(\)](#).

```
from ftpMirror import *

#FTP url, source directory, credentials
url = 'xx.xxx.xxx.x'
sourceDir = '/Folder/SubFolder/'
username = 'username'
password = 'password'

#Destination root directory
destDir = 'C:/Temp'

ftp = ftpMirror(url, username, password, sourceDir, destDir)
ftp.exclude = ['.NEF', '.avi']
ftp.download()
```

Included Libraries

The following libraries are used by the ftpMirror class:

```
import os
import time
import ftplib
```

__init__

This is a location to hold variables which can be utilized by the other functions inside ftpMirror. An instance of "FTP" is opened here with the URL, username, and password. The current ftp working directory is set to the source directory.

```
class ftpMirror:

    def __init__(self, url, username, password, sourceDir, destDir):
        """A class to mirror an FTP directory to local storage"""
        self.url = url
        self.ftp = ftplib.FTP(url, username, password)
        self.sourceDir = sourceDir
        self.ftp.cwd(sourceDir)
        self.destDir = destDir

        self.ftpDir = []
        self.ftpFiles = []
        self.ftpNumFiles = 0
        self.ftpSizeGB = 0

        self.destFiles = []
        self.destNumFiles = 0
        self.destSizeGB = 0

        self.remainingFiles = 0
        self.remainingGB = 0
        self.exclude = []
```

download

The main sequence control is performed here. Lists of files in the FTP and those existing in the local machine are made and the remaining file count and size to be downloaded is displayed. If all files have already been downloaded, the user is notified that there is nothing to download. If there are files still to be downloaded and the user chooses to download them, the FTP folder structure is created on the local machine and any remaining files are downloaded. Upon completion, total download time in minutes is displayed.

```
def download(self):
    """Main sequence control"""
    print('FTP Mirror Utility: checking directories...')
    self.buildDirList()
    self.checkSource()
    self.buildFileList()
    self.checkDest()

    self.remainingFiles = self.ftpNumFiles - self.destNumFiles
    self.remainingGB = self.ftpSizeGB - self.destSizeGB

    if self.remainingFiles == 0:
        print(f'FTP and {self.destDir} are already synchronized, nothing to download')
        self.exitPrompt()
    else:
        self.preview()

        if self.getDecision():
            self.createDirs()
            startTime = time.time()
            self.transferFiles()
            endTime = (time.time() - startTime) / 60
            print(f'Download Complete, {self.remainingFiles} file(s) in {endTime:.2f} minutes')
            self.exitPrompt()

        else:
            print('Exiting without downloading')
            time.sleep(1)
```

buildDirList

Starting from the given source directory, this function searches down each sub directory until no more folders can be found. It has dependency on the function `getNumDirs`. By appending newly found directories to the list of ftp directories and then looking at the length of ftp directories, this guarantees that it searches down as far as possible and that no sub folders are missed.

```
def buildDirList(self):
    """get all FTP folder names until there are no deeper folders"""
    prevDirNum = -1
    currDirNum = 0
    self.ftpDir = self.ftp.nlst(self.sourceDir)

    while (prevDirNum != currDirNum):
        prevDirNum = len(self.ftpDir)
        for dir in self.ftpDir:
            if self.getNumDirs(dir) != 0:
                subDirs = self.ftp.nlst(dir)
                for subDir in subDirs:
                    if subDir not in self.ftpDir:
                        self.ftpDir.append(subDir)
                        currDirNum = len(self.ftpDir)
```

getNumDirs

Given a directory, this function returns an integer representing the number of folders in that directory. It has a dependency on the function `isDir`. It checks each row of a dir response to see if that line is a directory or not.

```
def getNumDirs(self, dir):
    """Returns the number of folders in an FTP directory"""
    numDirs = 0
    rows = []
    self.ftp.dir(dir, rows.append)

    for i in range(0, len(rows)):
        if self.isDir(rows[i]):
            numDirs += 1
    return numDirs
```

isDir

Checks if a single line of a dir response contains the text "<DIR>".

```
def isDir(self, file):
    """Checks if a dir response line is a directory"""
    if '<DIR>' in file:
        return True
    else:
        return False
```

checkSource

Computes the total size of the ftp directory files not including any files whose extension is in the excluded list. This function uses the ftplib method 'dir' instead of 'size' because of the advantage in speed.

```
def checkSource(self):
    """Computes total FTP file size without excluded file types"""
    for dir in self.ftpDir:
        rows = []
        self.ftp.dir(dir, rows.append)
        for row in range(0, len(rows)):
            temp = self.removeSpaces(list(rows[row].split(' ')))
            if(temp[2] != '<DIR>' and self.noExclusions(temp[-1])):
                self.ftpSizeGB += int(temp[2]) / 1000000000
```

removeSpaces

Removes long blank sections from a 'dir' response line which has been split into a list so that desired components ending up at predictable indexes.

```
def removeSpaces(self, list):
    """Removes the blank spaces from a line of a dir response"""
    noBlanks = [value for value in list if value != '']
    return noBlanks
```

noExclusions

Checks if any of the user specified file extensions in the exclusion list are present. If no exclusions are found, 'True' is returned. In case the string originates as one component of a 'dir' line, the split is ignored since there will be no '/' characters present. In case the incoming string is a line from a 'nlst' response, the string will be like: "folder/subfolder/subfolder/file.ext". So, the split creates a list from the string and looks at the last value only.

```
def noExclusions(self, string):
    """Returns True if no excluded file extensions are present"""
    exclusions = 0
    fileName = string.split('/')
    for exclusion in self.exclude:
        if exclusion in fileName[-1]:
            exclusions += 1
    if exclusions == 0:
        return True
    else:
        return False
```

buildFileList

Builds a list of all files in the FTP source directory and its sub directories. This is not done within 'checkSource' because the 'dir' method is used there for speed but has challenges extracting the file name correctly in case the file name contains spaces. Has dependency on 'isFile'.

By using 'nlst' response lines to build the file list, the folder structure is already attached like: "folder/subfolder/subfolder/file.ext". When writing to the local machine directory it is only necessary to attach the destination directory at the beginning to have a full file path.

```
def buildFileList(self):
    """Builds a list of all files in the FTP source directory"""
    for dir in self.ftpDir:
        files = self.ftp.nlst(dir)
        for file in files:
            if self.isFile(file):
                if self.noExclusions(file):
                    self.ftpFiles.append(file)

    self.ftpNumFiles = len(self.ftpFiles)
```


isFile

Takes a single line from a nlst response, converts it to a list and looks at the last 3rd or 4th character to see if it is a ".". By doing so implies that it is a file with an extension of length 2 or 3 characters. Returns 'True' if the nlst line is determined to be a file.

```
def isFile(self, file):  
    """Checks if a nlst response line is a file"""  
    if (list(file)[-3] == '.' or list(file)[-4] == '.'):   
        return True  
    else:  
        return False
```

checkDest

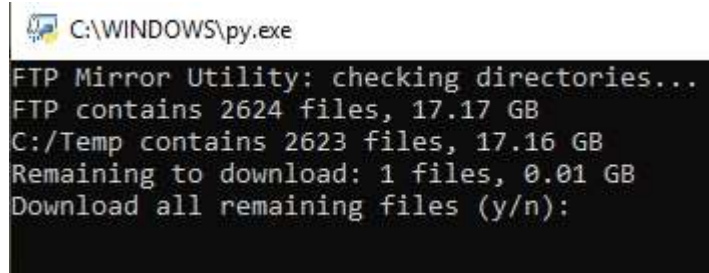
Checks which files in the FTP have already been downloaded to the local machine. The total size of pre-existing files is also computed.

```
def checkDest(self):  
    """Checks which files are already in the destination"""  
    for file in self.ftpFiles:  
        if os.path.exists(self.destDir + file):  
            self.destFiles.append(file)  
            self.destSizeGB += os.path.getsize(self.destDir + file) / 1000000000  
  
    self.destNumFiles = len(self.destFiles)
```

preview

Displays a summary to the user including file count and size in the ftp, already on the local machine, and those which still need to be downloaded.

```
def preview(self):  
    """Prints a summary of the files in FTP and local machine"""  
    print(f'FTP contains {self.ftpNumFiles} files, {self.ftpSizeGB:.2f} GB')  
    print(f'{self.destDir} contains {self.destNumFiles} files, {self.destSizeGB:.2f} GB')  
    print(f'Remaining to download: {self.remainingFiles} files, {self.remainingGB:.2f} GB')
```



```
C:\WINDOWS\py.exe  
FTP Mirror Utility: checking directories...  
FTP contains 2624 files, 17.17 GB  
C:/Temp contains 2623 files, 17.16 GB  
Remaining to download: 1 files, 0.01 GB  
Download all remaining files (y/n):
```

getDecision

Asks the user if they want to proceed to download the files. If a yes decision is given, 'True' is returned.

```
def getDecision(self):  
    """Get Decision from user to download files"""  
    decision = ''  
    while(True):  
        decision = input('Download all remaining files (y/n): ').upper()  
        if decision == 'Y':  
            return True  
        elif decision == 'N':  
            return False  
        else:  
            print('Invalid input')
```

createDirs

Re-creates the FTP folder structure in the specified local machine directory. Checks if a folder already exists before creating it.

```
def createDirs(self):  
    """Mirrors FTP folder structure on local machine"""  
    for dir in self.ftpDir:  
        if not os.path.exists(self.destDir + dir):  
            os.makedirs(self.destDir + dir)
```


transferFiles

Copies all files which are not already on the local machine from the ftp to the local machine. The entire list of files is used as a reference in case a file which was only partially written previously is not missed. If a file exists in the ftp and the local machine, each size is checked for consistency. Any files found on the local machine which have a file size which is too small are deleted and re-downloaded. Before transferring a file, a check is done to see if it already exists or not. So, if a download is interrupted, by restarting the script, the download will pick up where it left off and all existing work will still count.

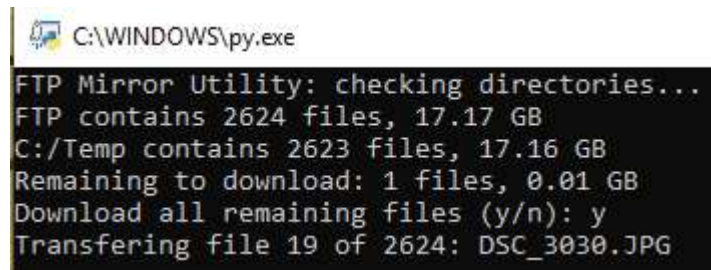
The “end = ‘\r’” at the end of the print statement is meant to show one file at a time when running the script via double click. If running the script in the python shell using “F5” this will cause the terminal to fill up and operate slowly.

```
def transferFiles(self):
    """Copies files from FTP to local machine"""
    for file in self.ftpFiles:
        path = self.destDir + file

        if os.path.isfile(path):
            if os.path.getsize(path) != self.ftp.size(file):
                os.remove(path)

        fileName = file.split('/')
        index = self.ftpFiles.index(file) + 1

        if not os.path.isfile(path):
            print(f'Transferring file {index} of {self.ftpNumFiles}: {fileName[-1]}', end = '\r')
            with open(path, 'wb') as f:
                self.ftp.retrbinary('RETR ' + file, f.write)
```

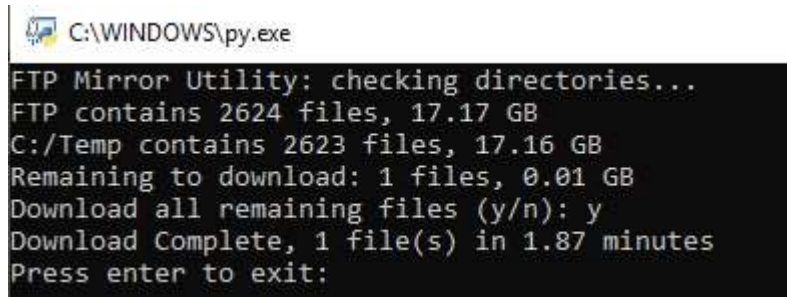


```
C:\WINDOWS\py.exe
FTP Mirror Utility: checking directories...
FTP contains 2624 files, 17.17 GB
C:/Temp contains 2623 files, 17.16 GB
Remaining to download: 1 files, 0.01 GB
Download all remaining files (y/n): y
Transferring file 19 of 2624: DSC_3030.JPG
```

exitPrompt

prompts the user to press 'enter' to exit and quits the ftp connection.

```
def exitPrompt(self):  
    """Prompts the user to press enter to exit"""  
    input('Press enter to exit: ')  
    self.ftp.quit()  
    time.sleep(1)
```



A screenshot of a Windows command prompt window. The title bar shows the icon for a Python file and the path 'C:\WINDOWS\py.exe'. The command prompt displays the following text:

```
FTP Mirror Utility: checking directories...  
FTP contains 2624 files, 17.17 GB  
C:/Temp contains 2623 files, 17.16 GB  
Remaining to download: 1 files, 0.01 GB  
Download all remaining files (y/n): y  
Download Complete, 1 file(s) in 1.87 minutes  
Press enter to exit:
```