```
            Set the logging level used by the library.
            â€˜quiet, -8â€™
            Show nothing at all; be silent.
            â€˜panic, 0â€™
            Only show fatal errors which could lead the process to crash, such as an assertion failure. This is not currently used for anything.
            â€˜fatal, 8â€™
            Only show fatal errors. These are errors after which the process absolutely cannot continue.
            â€˜error, 16â€™
            Show all errors, including ones which can be recovered from.
            â€˜warning, 24â€™
            Show all warnings and errors. Any message related to possibly incorrect or unexpected events will be shown.
            â€˜info, 32â€™
            Show informative messages during processing. This is in addition to warnings and errors. This is the default value.
            â€˜verbose, 40â€™
            Same as info, except more verbose.
            â€˜debug, 48â€™
            Show everything, including debugging information.
            â€˜trace, 56â€™
        """
        subprocess.call(['ffmpeg','-v', 'warning', '-y', '-i', self.path_name+'/'+filename,self.path_name+'/wav/'+filename+'.wav'])
        return

    def nameFileByName(self, soundObject):
        if not (self.state):
            return False
        fullfilepath = self.path_name+"/"+soundObject.name
        if (os.path.isfile(fullfilepath)):
            if config.DEBUG:
                print ("dateiname vorhanden: "+soundObject.name)
        else:
            if config.DEBUG:
                print ("datei muss geladen werden:")
                print("\t\tDownloading:", soundObject.name)
            #if sound.name.endswith(sound.type):
            filename = soundObject.name
            soundObject.retrieve_preview(self.path_name, name=filename)
            #else:
            #   filename = "%s.%s" % (sound.name, sound.type)
            #   sound.retrieve_preview(self.path_name, name=filename)
        return

    def filterByDuration(self, soundsObject, minDuration, maxDuration):
        if not (self.state):
            return False
        sounds = soundsObject
        soundList = []
        tmp = time.time()
        for sound in sounds:
            soundList += [sound]
        random.shuffle(soundList)
        filteredObjects = []
        i = 0
        if config.DEBUG:
            print ("dauer fÃ¼r shuffle: ")
            print (time.time()-tmp)
            print ("Preselected Sounds:")
            print (soundList)
        for sound in soundList:
            if (i >= 0) & (i < config.COUNT):
                if (int(sound.duration) >= minDuration) & (int(sound.duration) <= maxDuration):
                    filteredObjects += [sound]
                    i += 1
        if config.DEBUG:
            print ("selected sounds after filtering:")
            print (filteredObjects)
        return filteredObjects

```