

# msaUtils Module

## .file

### Attributes

#### archive\_pack\_formats module-attribute

```
archive_pack_formats = shutil.get_archive_formats()
```

#### archive\_unpack\_formats module-attribute

```
archive_unpack_formats = shutil.get_unpack_formats()
```

### Classes

#### FileDelete

File Delete Class

PARAMETER	DESCRIPTION
<code>uid</code>	str, the GUID of the file <b>TYPE:</b> <code>str</code>
<code>root_path</code>	str, dir name of the file <b>TYPE:</b> <code>str</code> <b>DEFAULT:</b> <code>os.path.join(os.path.dirname(__file__))</code>
<code>uploads_dir</code>	str, the folder the file was uploaded to. <b>TYPE:</b> <code>str</code> <b>DEFAULT:</b> <code>'data/uploads'</code>

#### Attributes

`root_path` instance-attribute

```
root_path = root_path
```

**uid** instance-attribute

```
uid = uid
```

**uploads\_dir** instance-attribute

```
uploads_dir = uploads_dir
```

## Functions

**`__init__`**

```
__init__(  
    uid: str,  
    root_path: str = os.path.join(  
        os.path.dirname(__file__)  
    ),  
    uploads_dir: str = "data/uploads",  
)
```

**`delete_files`** async

```
delete_files()
```

## FileExtNotAllowed

Bases: ServerHTTPException

raise when the upload file ext not allowed

## FileMaxSizeLimit

Bases: ServerHTTPException

raise when the upload file exceeds the max size

## FileUpload

FileUpload Class

PARAMETER	DESCRIPTION
<code>filesize</code>	int, size in bytes <b>TYPE:</b> <code>int</code>
<code>root_path</code>	str = dirname of the file <b>TYPE:</b> <code>str</code> <b>DEFAULT:</b> <code>os.path.join(os.path.dirname(__file__))</code>
<code>uploads_dir</code>	str = "data/uploads", where to store the file <b>TYPE:</b> <code>str</code> <b>DEFAULT:</b> <code>'data/uploads'</code>
<code>not_allow_extensions</code>	Optional[List[str]] = None, exclude file extensions from upload ability <b>TYPE:</b> <code>Optional[List[str]]</code> <b>DEFAULT:</b> <code>None</code>
<code>max_size</code>	int = 150000000, max allowed filesize in bytes for upload <b>TYPE:</b> <code>int</code> <b>DEFAULT:</b> <code>150000000</code>
<code>createUIDSubFolders</code>	bool = False, if enabled the system creates Subfolders by the UID <b>TYPE:</b> <code>bool</code> <b>DEFAULT:</b> <code>False</code>

### Attributes

`content_type` instance-attribute

```
content_type = ''
```

`createSubFolders` instance-attribute

```
createSubFolders = createUIDSubFolders
```

`file_size` instance-attribute

```
file_size = filesize
```

`filename_generator` instance-attribute

```
filename_generator = nameGen
```

`fullpath` instance-attribute

```
fullpath = ''
```

magic\_desc instance-attribute

```
magic_desc = ''
```

magic\_type instance-attribute

```
magic_type = ''
```

max\_size instance-attribute

```
max_size = max_size
```

name instance-attribute

```
name = ''
```

not\_allow\_extensions instance-attribute

```
not_allow_extensions = not_allow_extensions
```

root\_path instance-attribute

```
root_path = root_path
```

uid instance-attribute

```
uid = str(uuid4())
```

uploads\_dir instance-attribute

```
uploads_dir = uploads_dir
```

## Functions

`__init__`

```
__init__(  
    filesize: int,
```

```

root_path: str = os.path.join(
    os.path.dirname(__file__)
),
uploads_dir: str = "data/uploads",
not_allow_extensions: Optional[List[str]] = None,
max_size: int = 150000000,
createUIDSubFolders: bool = False,
)

```

### save\_file async

```
save_file(filename: str, ufile: UploadFile)
```

Save the file

PARAMETER	DESCRIPTION
<code>filename</code>	the name of the file it should be saved uner <b>TYPE:</b> <code>str</code>
<code>ufile</code>	UploadFile instance of the file to save <b>TYPE:</b> <code>UploadFile</code>

### upload async

```

upload(
    file: UploadFile, chunk_size: int = 1024 * 1024 * 50
) -> str

```

upload the file

PARAMETER	DESCRIPTION
<code>chunk_size</code>	1024 * 1024 * 50 = 50 megabytes <b>DEFAULT:</b> <code>1024 * 1024 * 50</code>
<code>file</code>	The UploadFile instance of the file for upload. <b>TYPE:</b> <code>UploadFile</code>

RETURNS	DESCRIPTION

RETURNS	DESCRIPTION
<code>filename</code>	str complete filename after created safe version <b>TYPE:</b> <code>str</code>

## InvalidResource

Bases: `ServerHTTPException`

raise when has invalid resource

## NoSuchFieldFound

Bases: `ServerHTTPException`

raise when no such field for the given

## ServerHTTPException

Bases: `HTTPException`

### Functions

`__init__`

```
__init__(error: str = None)
```

## Functions

### checkIfFileIsArchive `async`

```
checkIfFileIsArchive(file: UploadFile)
```

Check if File is an Archive like zip or tar

### createMSAFile `async`

```
createMSAFile(file: UploadFile, up: FileUpload) -> MSAFile
```

Create an MSAFile Instance for the provided file

PARAMETER	DESCRIPTION
<code>file</code>	is the UploadFile <b>TYPE:</b> <code>UploadFile</code>
<code>up</code>	is the FileUpload Instance <b>TYPE:</b> <code>FileUpload</code>

RETURNS	DESCRIPTION
<code>mf</code>	New MSAFile instance. <b>TYPE:</b> <code>MSAFile</code>

### createMSAFileFromUnpacked `async`

```
createMSAFileFromUnpacked(
    filepath: str, process_uid: str
) -> MSAFile
```

Create an MSAFile Instance for a file from an archive, and keep them under one group by the process\_uid

PARAMETER	DESCRIPTION
<code>filepath</code>	str of the file path <b>TYPE:</b> <code>str</code>
<code>process_uid</code>	str of the group process id (GUID) <b>TYPE:</b> <code>str</code>

RETURNS	DESCRIPTION
<code>mf</code>	New MSAFile instance. <b>TYPE:</b> <code>MSAFile</code>

### get\_all\_dirs `async`

```
get_all_dirs(path: str) -> List
```

Get all Directories of a specific root path

PARAMETER	DESCRIPTION
<code>path</code>	The root path to wlk through <b>TYPE:</b> <code>str</code>

RETURNS	DESCRIPTION
<code>result</code>	The list of directory names <b>TYPE:</b> <code>List</code>

### get\_all\_dirs\_with\_subdirs `async`

```
get_all_dirs_with_subdirs(  
    path: str, subdirs: set[str]  
) -> List[str]
```

Get all Directories which have a specific set of Sub Dirs

PARAMETER	DESCRIPTION
<code>path</code>	The root path to wlk through <b>TYPE:</b> <code>str</code>
<code>subdirs</code>	a set of subdir names which need to be found to add the dirs <b>TYPE:</b> <code>set[str]</code>

RETURNS	DESCRIPTION
<code>result</code>	The list of directory names <b>TYPE:</b> <code>List[str]</code>

### load\_bytes\_from\_file `async`

```
load_bytes_from_file(path_file_name: str) -> bytes
```

Load a bytes/binary content from a file

PARAMETER	DESCRIPTION
<code>path_file_name</code>	The pathfilename to load the content from <b>TYPE:</b> <code>str</code>

RETURNS	DESCRIPTION
<code>bytes</code>	The content of the file <b>TYPE:</b> <code>bytes</code>

### load\_text\_from\_file `async`

```
load_text_from_file(path_file_name: str) -> str
```

Load a text content from a file

PARAMETER	DESCRIPTION
<code>path_file_name</code>	The pathfilename to load the content from <b>TYPE:</b> <code>str</code>

RETURNS	DESCRIPTION
<code>str</code>	The content of the file <b>TYPE:</b> <code>str</code>

### nameGen

```
nameGen(uid: uuid, file_name: str) -> str
```

Gets an uuid and file\_name combined name back

### save\_bytes\_to\_file `async`

```
save_bytes_to_file(
    path_file_name: str, binary_content: bytes
) -> None
```

Saves a bytes content to file

PARAMETER	DESCRIPTION
<code>path_file_name</code>	The pathfilename to save the content to <b>TYPE:</b> <code>str</code>
<code>binary_content</code>	The bytes content to store inside the file <b>TYPE:</b> <code>bytes</code>

RETURNS	DESCRIPTION
<code>None</code>	None

`save_text_to_file` `async`

```
save_text_to_file(
    path_file_name: str, text_content: str
) -> None
```

Saves a str content to file

PARAMETER	DESCRIPTION
<code>path_file_name</code>	The pathfilename to save the content to <b>TYPE:</b> <code>str</code>
<code>text_content</code>	The str content to store inside the file <b>TYPE:</b> <code>str</code>

RETURNS	DESCRIPTION
<code>None</code>	None

`secure_filename`

```
secure_filename(filename: str) -> str
```

Pass it a filename, and it will return a secure version of it. This filename can then safely be stored on a regular file system and passed to :func:os.path.join. The filename returned is an ASCII only string for maximum portability. On Windows systems the function also makes sure that the file is not named after one of the special device files.

**Example**

```
secure_filename("My cool movie.mov")
'My_cool_movie.mov'
secure_filename("../../../etc/passwd")
'etc_passwd'
secure_filename('xyz contain cool \xfcm1\xe4uts.txt')
'xyz_contain_cool_umlauts.txt'
```

The function might return an empty filename. It's your responsibility to ensure that the filename is unique and that you abort or generate a random filename if the function returned an empty one. ...

versionadded:: 0.5

PARAMETER	DESCRIPTION
filename	the filename to secure <b>TYPE:</b> str

Last update: September 24, 2022

Created: September 24, 2022