msaSDK Module

`.utils.fileupload`

Attributes

__version__ `module-attribute`

```
__version__ = '0.1.0'
```

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
`uid`	str , the GUID of the file **TYPE:** `str`
`root_path`	str, dirname of the file **TYPE:** `str` **DEFAULT:** `os.path.join(os.path.dirname(__file__))`

PARAMETER	DESCRIPTION		
`uploads_dir`	str, the folder the file was uploaded to. **TYPE:** `str`		**DEFAULT:** `'data/uploads'`

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

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
filename	the name of the file it should be saved uner **TYPE:** `str`
ufile	UploadFile instance of the file to save **TYPE:** `UploadFile`

upload `async`

```
upload(file: UploadFile)
```

upload the file

PARAMETER	DESCRIPTION
file	The UploadFile instance of the file for upload. **TYPE:** `UploadFile`

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
`file`	is the UploadFile **TYPE:** `UploadFile`

PARAMETER	DESCRIPTION
`up`	is the FileUpload Instance **TYPE:** `FileUpload`

RETURNS	DESCRIPTION
`mf`	New MSAFile instance. **TYPE:** `MSAFile`

createMSAFileFromUnpacked `async`

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

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

PARAMETER	DESCRIPTION
`filepath`	str of the file path **TYPE:** `str`
`processuid`	str of the group process id (GUID) **TYPE:** `str`

RETURNS	DESCRIPTION
`mf`	New MSAFile instance. **TYPE:** `MSAFile`

nameGen

```
nameGen(uid, file)
```

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('i contain cool \xfcml\xe4uts.txt')
> 'i_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 **TYPE:** `str`

Last update: September 13, 2022

Created: September 13, 2022