# Module **`grouper`**

## Functions

**`def AppendNestedSubgroups(parentGroupsDict, subgroupName, grouper: Grouper)`**

### Description

Uses the specified subgroupName to create the dict of grouped files for that particular subgroup, the result is appended to the previously constructed parentGroupsDict.

The groupItems in the parentGroupsDict are either a list of files or dicts containing further groups. If they are files, this function performs the desired grouping on those files. If they are dicts, this function is then called recursively in order to create all nested subsubgroups within this particular subgroup, until files are found and thus grouped.

### Args:

```
 parentGroupsDict : dict[str] = (dict[str] = list[File] ... or list[file] )
    #: The already constructed dict containing parent groups
subgroupName : str
    #: The name of the subgroup at the current level in the hierarchy
grouper : Grouper
    #: The grouper object defining the hierarchy
```

### Outputs

```
 parentGroupsDict : dict[str] = (dict[str] = list[File] ... or list[file] )
    #: The already constructed dict containing parent groups and appended subgroup
```

**`def BuildPreviewJsTreeJson(grouper: Grouper)`**

### Description

Given the previewPathDict and groupStructureDicts within the Grouper object, this function creates the JSON strings required to build the preview jsTree instance.

### Args:

```
grouper : Grouper
    #: The grouper object defining the hierarchy and storing grouping results
```

### Outputs:

```
allJson : [str]
    #: String array containing all JSON strings for the preview instance
```

**`def CreateAllNestedFolders(nestedGroupsDict, grouper: Grouper)`**

### Description

Uses the dict containing the nested groups to create empty folders for each group.

This step is completed before the user has verified changes, but no files are moved at this stage - only empty folders created. If the changes are not accepted these empty folders are removed.

The proposed changes (new locations of grouped files) are stored in the grouper.previewPathDict object. The subgroup -> parent group folder structure is stored in the grouper.groupStructureDict object (required for building jsTree instance)

### Args:

```
 nestedGroupsDict : dict[str] = (dict[str] = list[File] ... or list[file] )
    #: The dict containing nested groups
grouper : Grouper
    #: The grouper object defining the hierarchy
```

**`def CreateAllNestedGroups(files: list, grouper: Grouper)`**

### Description

Uses the defined hierarchy in the Grouper object to create a dict containing all nested groups.

The nested dict is of the form groupName -> groupItems, where the groupItems may be more dicts containing further groups.

### Args:

```
 files : list[File or pathlike]
    #: The list of files to be grouped
grouper : Grouper
    #: The grouper object defining the hierarchy
```

### Outputs

```
 nestedGroupsDict : dict[str] = (dict[str] = list[File] ... or list[file] )
    #: Nested dict containing groupName -> groupItemsDict
```

**`def CreateGrouperBackup(previewPathDict)`**

### Description

Creates a backup of the original locations of the files to be grouped, by storing a dict (as JSON string) with new proposed location -> old location.

This JSON string can additionaly be downloaded by the user and used to restore from backup at a later time after navigating away from the page.

### Args:

```
 previewPathDict : dict[File] = Path
    #: The dict with the proposed locations of the grouped files
```

### Outputs:

```
 backupJson : [str]
    #: JSON string serialising the backup dict
```

**`def CreateNestedGroupFolders(groupName, groupItems, parent: pathlib.Path, previewPathDict, groupStructureDict)`**

### Description

Recursively created the subfolders for each subgroup within the specified group with groupName.

The groupItems are either a list of files or dicts containing further groups. If they are files, their new proposed locations are stored in the previewPathDict. If they are dicts, this function is then called recursively in order to create all nested subgroups within this particular group.

### Args:

```
groupName : str
    #: The name of the current group
groupItems : dict[str] = (dict[str] = list[File] ... or list[file] ) or list[file]
    #: The items in the group - either more group dicts or a list of files
parent : Path
    #: The path of the parent folder that will contain this subgroup folder
previewPathDict : dict[Path] = list[File]
    #: The dict of Path -> files proposed for this new location
groupStructureDict : dict[Path] = Path
    #: The dict of subgroup Path -> parent group Path
```

### Outputs

```
previewPathDict : dict[Path] = list[File]
    #: The dict of Path -> files proposed for this new location
groupStructureDict : dict[Path] = Path
    #: The dict of subgroup Path -> parent group Path
```

**`def FileGrouperMain_Preview(groupOrder: list, includedFolders: list, destination: pathlib.Path, keywords=None)`**

### Description

The main function used to build the preview of the desired grouping. The group hierarchy is passed as a string list and used to construct the Grouper object for this grouping operation.

All files within the includedFolders will be considered for grouping. No files will be moved until the user verifies the proposed changes.

### Args:

```
groupOrder : list[str]
    #: The group order list (of group names) parsed from the user input
includedFolders : list[Folder]
    #: List of folders within which files will be grouped
destination : Path
    #: The destination for the groups
```

### Kwargs:

```
keywords : list[str] : default=None
```

### Outputs:

```
allJson : [str]
    #: String array containing all JSON strings for the preview instance
```

**`def FileGrouperMain_Verified(accepted: bool, destination, previewPathDict, groupStructureDict)`**

### Description

The main function used to move files (or cancel) once the user has verified the changes.

If the changes were not accepted the empty group folders previously created will be removed.

This does not accept a grouper instance (since it is only local to the FileGrouperMain_Preview scope) but rather the destination path and dicts that were previously stored in the cache by the grouping_utility view.

### Args:

```
accepted : bool
    #: True if the changes were accepted
destination : Path
    #: The destination for the groups
        previewPathDict : dict[Path] = list[File]
    #: The dict of Path -> files proposed for this new location
groupStructureDict : dict[Path] = Path
    #: The dict of subgroup Path -> parent group Path
```

### Kwargs:

```
keywords : list[str] : default=None
```

### Outputs:

```
allJson : [str]
    #: String array containing all JSON strings for the preview instance
```

**`def GroupByKeyword(files: list, keywords: list)`**

### Description

Group the list of files based on the supplied list of keywords. Files are placed into a dataframe which is searched using the MultiTermFuzzySearch method.

The matching files for each keyword are thus grouped - NOTE this does not yet account for the situation where a single file matches multiple keywords, which would cause the file to be placed into multiple locations (not possible on filesystem).

Files that are not matched to any keyword are placed into the "Other" group. If a supplied keyword is already named "Other", the unmatched group is renamed to "_Other".

Returns a dict of keyword -> list of files matching that keyword

### Args:

```
files : list[File or pathlike]
    #: The list of files to be grouped
keywords : list[str]
    #: The list of keyword strings to search for
```

### Outputs

```
groupDict: dict[str] = list[File]
    #: Dict with keyword -> list of matched files
```

**`def GroupBySubtype(files: list)`**

### Description

Group the list of files based on their extension subtype tag. Returns a dict of extension subtype (tag name, e.g. "Spreadsheet") -> list of files of that subtype

### Args:

```
files : list[File or pathlike]
    #: The list of files to be grouped
```

### Outputs

```
groupDict: dict[str] = list[File]
    #: Dict with subtype tag name -> list of files
```

**`def GroupByType(files: list)`**

### Description

Group the list of files based on their extension Type tag. Returns a dict of extension type (tag name, e.g. "Documents") -> list of files of that type

### Args:

```
files : list[File or pathlike]
    #: The list of files to be grouped
```

### Outputs

```
typeFilesDict: dict[str] = list[File]
    #: Dict with type tag name -> list of files
```

### Notes:

Since files only have the subtype tag attribute, the type tag is accessed using the subtypeTag.parentTag attribute.

**`def MoveFilesIntoGroups(previewPathDict)`**

### Description

This function moves files from their original locations to the newly proposed grouped locations, once the user has accepted the changes to be made.

### Args:

```
previewPathDict : dict[File] = Path
    #: The dict with the proposed locations of the grouped files
```

**`def RestoreGroupedFilesFromBackup(backupJson)`**

### Description

From the backup JSON string this will restore grouped files to their original locations.

NOTE: Any subsequent changes to the new locations of the grouped files will thus be missed by such a backup, which only stores the locations immediately after grouping has been performed.

### Args:

```
backupJson : [str]
    #: JSON string serialising the backup dict
```

**`def TestGroup(files: list)`**

## Classes

**`class Grouper (destination: pathlib.Path, groupOrder: list, keywords=None)`**

### Class:

This class handles the grouping operations requested by the user. It defines references to the possible grouping methods in the groupMethods dict (methods referenced by name, e.g GroupByType = "Type") The class is constructed with the grouping order supplied from the interface, the desired destination of the grouped files and any keywords (if relevant). The first level of groups defined in the hierarchy will be created as subdirectories of the specified destination.

## Description

Constructor for the Grouper class. Takes the group order as a list of named grouping methods, the desired destination for the grouped files and any keywords if relevant.

## Args:

```
 destination : Path or pathlike
    #: The destination for the grouped files.
groupOrder : list[str]
    #: The list of group names, first element is outermost group.
```

## Kwargs:

```
keywords : list[str] : default=None
    #: The list of keyword strings to search for.
```

# Index

## Super-module

`fileorganiser`

## Global variables

`SORT_GROUPS`

## Functions

`AppendNestedSubgroups`
`BuildPreviewJsTreeJson`
`CreateAllNestedFolders`
`CreateAllNestedGroups`
`CreateGrouperBackup`
`CreateNestedGroupFolders`
`FileGrouperMain_Preview`
`FileGrouperMain_Verified`
`GroupByKeyword`
`GroupBySubtype`
`GroupByType`
`MoveFilesIntoGroups`
`RestoreGroupedFilesFromBackup`
`TestGroup`

## Classes

**`Grouper`**