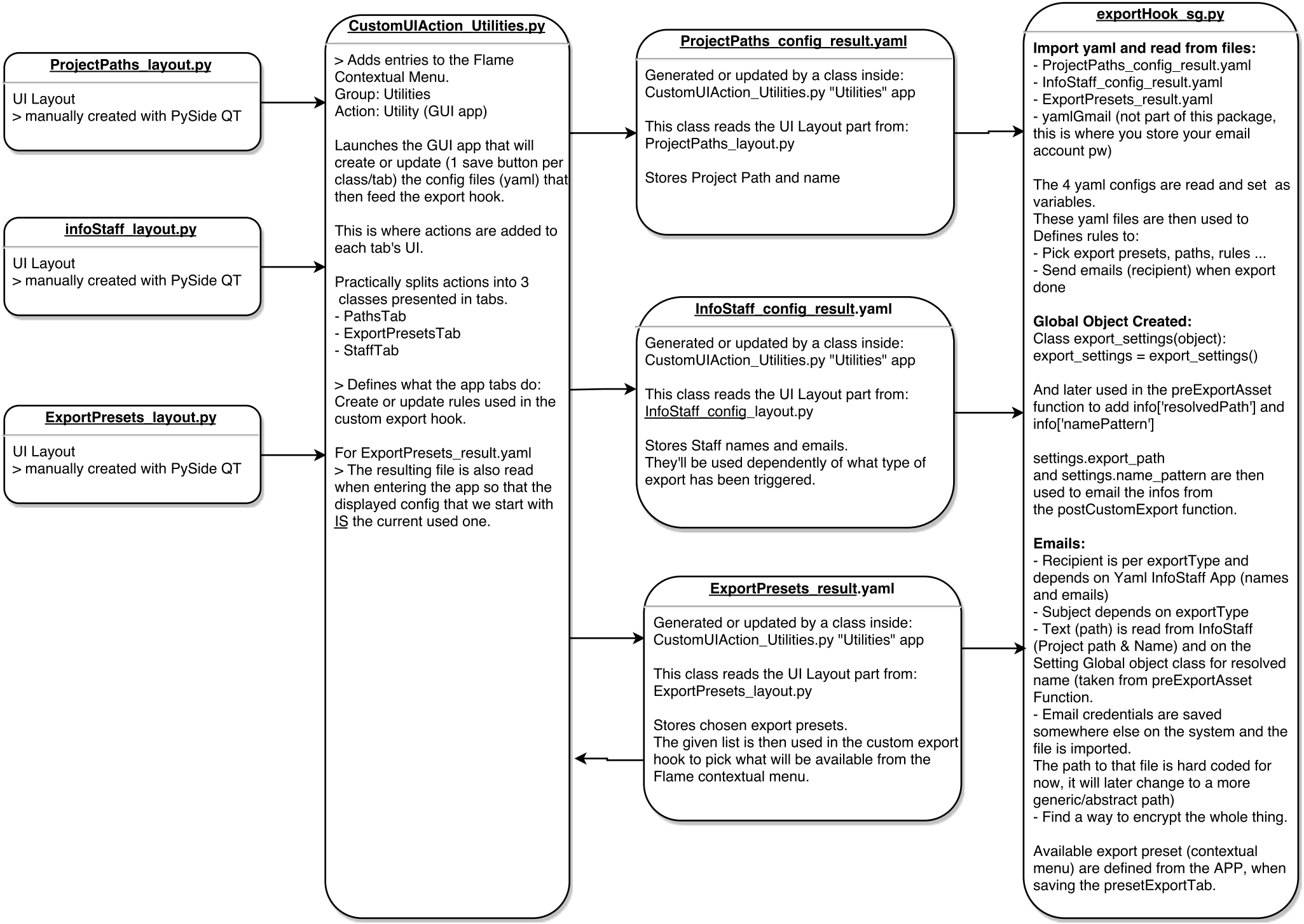
ProjectPaths_layout.py
UI Layout
> manually created with PySide QT
infoStaff_layout.py
UI Layout
> manually created with PySide QT
ExportPresets_layout.py
UI Layout
> manually created with PySide QT
CustomUIAction_Utilities.py
> Adds entries to the Flame Contextual Menu.
Group: Utilities
Action: Utility (GUI app)
Launches the GUI app that will create or update (1 save button per class/tab) the config files (yaml) that then feed the export hook.
This is where actions are added to each tab's UI.
Practically splits actions into 3 classes presented in tabs.
- PathsTab
- ExportPresetsTab
- StaffTab
> Defines what the app tabs do: Create or update rules used in the custom export hook.
For ExportPresets_result.yaml
> The resulting file is also read when entering the app so that the displayed config that we start with IS the current used one.
ProjectPaths_config_result.yaml
Generated or updated by a class inside: CustomUIAction_Utilities.py "Utilities" app
This class reads the UI Layout part from: ProjectPaths_layout.py
Stores Project Path and name
InfoStaff_config_result.yaml
Generated or updated by a class inside: CustomUIAction_Utilities.py "Utilities" app
This class reads the UI Layout part from: InfoStaff_config_layout.py
Stores Staff names and emails.
They'll be used dependently of what type of export has been triggered.
ExportPresets_result.yaml
Generated or updated by a class inside: CustomUIAction_Utilities.py "Utilities" app
This class reads the UI Layout part from: ExportPresets_layout.py
Stores chosen export presets.
The given list is then used in the custom export hook to pick what will be available from the Flame contextual menu.
exportHook_sg.py
Import yaml and read from files:
- ProjectPaths_config_result.yaml
- InfoStaff_config_result.yaml
- ExportPresets_result.yaml
- yamlGmail (not part of this package, this is where you store your email account pw)
The 4 yaml configs are read and set as variables.
These yaml files are then used to Defines rules to:
- Pick export presets, paths, rules ...
- Send emails (recipient) when export done
Global Object Created:
Class export_settings(object):
export_settings = export_settings()
And later used in the preExportAsset function to add info['resolvedPath'] and info['namePattern']
settings.export_path
and settings.name_pattern are then used to email the infos from the postCustomExport function.
Emails:
- Recipient is per exportType and depends on Yaml InfoStaff App (names and emails)
- Subject depends on exportType
- Text (path) is read from InfoStaff (Project path & Name) and on the Setting Global object class for resolved name (taken from preExportAsset Function.
- Email credentials are saved somewhere else on the system and the file is imported.
The path to that file is hard coded for now, it will later change to a more generic/abstract path)
- Find a way to encrypt the whole thing.
Available export preset (contextual menu) are defined from the APP, when saving the presetExportTab.