REY_LoggerNUtils `V0.4a`

- `V0.4` :- WIP
    - `REY_FetchV4` :- 1. `Scout` , 2. `Submodule` , 3. `ZipLinks` , 4. `Clone/Fetch` ☺
    - WIP: StackTrace on Crash / Signal Handler / `google breakpad` + `boost stacktrace` + `StackWalker` + `google crashpad` + `sentry` + `raygun` + `BugSnag` + `RollBar`
- `V0.3` :- DONE
    - `.install` :- it's a Folder for `lib-REY_LoggerNUtils.lib` & "external libraries" installation
    - `.forge` :- 😉 a new idea for external-library management
    - added `.forge/CMakeFiles/REY_FetchV2_fmt.cmake`
    - added `.forge/CMakeFiles/REY_FetchV3.cmake`
- `V0.2` :- Prefix_Tag:- 😢 REFACTORED ["amVK" --> "REY"]
- `V0.1` :- Initial Commit: moving from [GIST ---> GITHUB]
- `V0.1beta` :- https://gist.github.com/REYNEP/14a628ab270cae461a926ba212226492

Example

```cpp
// ---------- example. 1 ----------
#include "REY_Logger.hh"
int main(void) {
    REY_LOG("Hello, World!");
}

// ---------- example. 2:- you also got access to `fmt` 😄 ----------
#include <fmt/core.h>
int main(void) {
    fmt::print("Hello, World!\n");
    return 0;
}
/** cmake configure --> will automatically Fetch / Build / Link / IncludePath of `fmt`
 *  https://github.com/fmtlib/fmt */

// ---------- example. 3 ----------
// TBA
```

Building / Using [SUMMARY]

It's basically automatically handled ☺:-

```
git clone https://github.com/REYNEP/REY_LoggerNUtils <path>
# or
git add submoule https://github.com/REYNEP/REY_LoggerNUtils <path>

Way 1
    # Open `REY_LoggerNUtils` in VSCODE
    # F1 > CMake: Configure
    # F1 > CMake: Build
    # F1 > CMake: Install [Default Folder:- REY_LoggerNUtils/.install]

        # You can optinally take a glimpse @ "REY_LoggerNUtils/CMakeLists.txt" 😜
        # for better understanding.... it's pretty small

Way 2:- add these in your CMakeLists.txt
    #      add_subdirectory( <path/to/REY_LoggerNUtils> )
    # target_link_libraries( <your_target_name>  REY_LoggerNUtils )

Way 3:- REY_FetchV4
    # copy:- `REY_FetchV4.cmake`
            #`REY_FetchV4_X_RESET.cmake`
            #`REY_FetchV4_X.REY_LoggerNUtils.cmake`
            #`REY_FetchV4.REY_LoggerNUtils.cmake`
        into wherever you keep your CMakeFiles
    # include(REY_FetchV4.REY_LoggerNUtils.cmake) in your CMakeLists.txt

Way 4:- Meson & Premake Support [TBA]
Way 5:- Ninja/MakeFiles + Python Downloader Script [TBA]
```

```
# --------------------
# OUTPUT Options:-
#       Library Name:- REY_LoggerNUtils
                Usage:- target_link_libraries(<target_name> REY_LoggerNUtils)
#
# Possible Untested Output Options:- [Might Not Work, cz of SCOPE Limitations]
        ${fmt::fmt} ---------> this is a "Target"
#                              this is How you use it:-
                                   target_link_libraries(idk fmt::fmt)
                                   target_include_directories(idk PUBLIC fmt::fmt)
# --------------------
```

External Libraries [`.forge`]

0. assuming that you did `add_subdirectory(REY_LoggerNUtils)` in your CMakeLists.txt
1. `fmt` :- automatically ***"Fetched"*** --> ***Built*** --> *"**PUBLIC** linked to `REY_LoggerNUtils` "*
    - **"PUBLIC"** Linked:-
        - i.e. `fmt` will be available to you too
        - i.e. You can just `#include <fmt/core.h>`
        - & `fmt` will be automatically linked as you are linking `REY_LoggerNUtils` in CMAKE
    - `Official Repo` :- https://github.com/fmtlib/fmt
    - `What is it?` :- https://github.com/fmtlib/fmt?tab=readme-ov-file#examples
    - `CMake / Using` :- https://fmt.dev/11.1/get-started/
    - `CheatSheet / Code Examples` :- https://hackingcpp.com/cpp/libs/fmt.html
2. `.forge` :-
    - `lib-REY_LoggerNUtils.lib` will be INSTALLED here
    - `fmt` will be fetched here & installed here
    - I store/fetch/modify/custom-build External Libraries in here
    - For the whole idea, check:- https://github.com/REYNEP/REY_LoggerNUtils/tree/main/.forge
3. `google breakpad` :- [StackTracer on Crash]
    - very hard to build on windows.
        - However I found a really cool & nice wiki & how-to about it
            - https://github.com/d1vanov/quentier/wiki/Building-and-installation-of-Quentier's-dependencies#building-google-breakpad
            - Took me Half an hour to find this guide & finally fkin build this shit
    - Building Google Breakpad on Windows:- d1vanov's wiki on github
    - BREAKPAD vs CRASHPAD
        - https://stackoverflow.com/questions/52725299/what-is-the-difference-between-googles-breakpad-and-crashpad-libraries
    - `Official Repo` :- https://chromium.googlesource.com/breakpad/breakpad
    - `What is it?` :- https://chromium.googlesource.com/breakpad/breakpad/+/HEAD/docs/breakpad.png
    - `CMake / Using` :- d1vanov's wiki on github
    - `CheatSheet / Code Examples` :- Mozilla Intro, linux [starter-guide], mac, windows, processor-design, detes on stack-tracing, chatgpt
    - `Documentation` :- HEAD/docs

Features:-

1. `REY_Logger.hh` is ***lightweight***
    - No `#include <cstdlib>` or `#include <iostream>`
    - All `#include` was done inside `#ifdef REY_LOGGER_IMPLEMENTATION`
        - Actual Implementations compiled by:- `REY_Logger.cpp`
    - So this is basically like a ***standalone*** `~500Lines` of code
        - even if you `#include REY_Logger.hh` in 1000s of files....
            - REY_Logger:- 500Lines / file
            - std::iostream:- ~20,000-50,000 Lines / file
2. So we basically had to make a ***lightweight*** wrapper around `std::cout`
    - **class REY_Logger**
    - `malloc()` --> `REY_malloc()`
    - `memcpy()` --> `REY_memcpy()`
        - `REY_Utils::merge_sort` is still template based....
        - also `REY_memcpy` is used in `REY_ArrayDYN<T>::resize`
3. License:- ***Boost License***