UEVaultManager

Laurent Ongaro

May 12, 2024

CONTENTS:

UEVaultManager is an open-source assets manager that can list assets and their data from the Epic Games Marketplace. It's developed in Python, so it can run on any platform that support this language.

Its main purpose is to list the assets (with or without user login), filter (optional) and save the list into a file that can be reused later as a data source (in an Excel sheet for instance).

Note: This project is under active development.

CHAPTER

ONE

UEVAULTMANAGER

1.1 A free and open-source Epic Games Assets Manager for Unreal Engine

An Epic Launcher Asset management alternative available on all Platforms

UEVaultManager is an open-source assets manager that can list assets and their data from the Epic Games Marketplace. It's developed in Python, so it can run on any platform that support this language.

Its main purpose is to list the assets (with or without user login), filter (optional) and save the list into a file that can be reused later as a data source (in an Excel sheet for instance).

Please read the *Configuration* and *Usage* sections before creating an issue to avoid invalid issue reports.

1.1.1 Note:

UEVaultManager can be run as a CLI (command-line interface) application, it has to be run from a terminal (e.g. a Linux Shell, a PowerShell or a Dos Console).

Since the version 1.1.0, the Application can edit the content of a previous assets scan (using the 'list' or the 'scrap' commands). This feature offers GUI interface to edit the assets list, and to add or edit the user data for each asset.

If you find a problem with this application, please note that it's a free application, and it 's made on my spare time. So be patient and comprehensive, you can try to solve it by your own means if possible.

If you're stuck, you can create an issue on GitHub, so I'll be aware of, and I'll try to fix it, as quick as I can.

All bug reports, PR, ideas, improvement suggestions, code correction… are welcome !

Released under GNU General Public License v3.0

THIS TOOL IS PROVIDED AS IT IS. NO WARRANTY . AND, PLEASE, NO COMPLAIN . THANKS

1.1.2 Implemented Features:

- **Downloading and installing the assets you own from the EPIC MARKETPLACE (since the version 1.10.0)**
- **Scanning user folders for local assets (since the version 1.9.0)**
- **Scraping the marketplace page of an asset to get all its data using the EPIC GAME SERVICE API (since the version 1.8.0)**
- Since version 1.8.0, the application can also use a sqlite database to store the data and, as it, it could list ALLthe assets available on the marketplace (and not only the ones you own like in the previous versions).

- Using a several cache systems to avoid getting data using API calls and web scraping each time the application is run. The delay of cache conservation can be set in the configuration file
- Filtering the asset list by category before their listing
- Saving the resulting list in a csv or a json file
- Saving the metadata and the extra data in individual json files (one for each asset) in sub-folders of the config folder
- **Editing the content of a result file (json or csv) using a GUI (since the version 1.1.0)**
- Preserving user data for each asset (see the *Output Format and file* section).
  - Some fields in the result file (comments, personal note…) will be protected and not overwritten by a future data update.
- Listing and getting data about assets
  - all the metadata that are usually available: name, title, id, description, UE versions…
  - extra data grabbed from the marketplace page of an asset : price, review, tags, owned or not…
- Authenticating with Epic's service

1.1.3 Planned Features

Since version 1.4.3, the features can be listed using special Labels in the GitHub issues list.

- planed enhancements
- ascetic improvments
- ideas

1.1.4 Special thanks

Legendary team

This code was initially based on a lighter and improved version of the Legendary code base, with some addition regarding the listing and the management of unreal engine marketplace assets. So Thanks to the Legendary team for the fantastic work on their tool !!

Jetbrains

I intensively use JetBrains software for developing all my projects.

Thanks to JetBrains for their support on this project through their License Program For non-commercial open source development

Their tools are great ! If you don't know them, you should give them a try.

1.2 Known bugs and limitations

CHAPTER

TWO

QUICKSTART

Tip: When using PowerShell with the standalone executable, you may need to replace `UEVaultManager` with `.\UEVaultManager` in the commands below.

2.1 Installation

To use UEVaultManager, first install it using pip:

```
pip install UEVaultManager
```

2.2 log in

```
UEVaultManager auth
```

If the pywebview package is installed (that is done by the installation process), this should open a new window with the Epic Login.

Otherwise, authentication is a little finicky since we have to go through the Epic website and manually copy a code. The login page should open in your browser and after login you should be presented with a JSON response that contains a code (*authorizationCode*), just copy the code into the terminal and hit enter.

Alternatively you can use the *–import* flag to import the authentication from the Epic Games Launcher

Note that this will log you out of the Epic Launcher.

2.3 Listing your asset

```
UEVaultManager list
```

This will fetch a list of asset available on your account (only the assets you OWNED), the first time may take a while depending on how many asset you have.

2.4 Saving the list into a CSV file

```
UEVaultManager list -o "D:/ue_asset_list.csv"
```

You can manually edit some data in this file And you can update the data extrated from the Marketplace (new version, new release, desciption update...) by running the same command again. The changes you made manually will be preserved, depending on what fields (aka. columns) has been changed (see *Output Format and file* section).

2.5 Editing the list with the new GUI (since 1.6.0)

```
UEVaultManager edit -i "D:/ue_asset_list.csv"
```

For more details, please read the *The (new) Gui* section.

2.6 Scraping all the asset from the marketplace and store them in a database file (since 1.8.0)

```
UEVaultManager scrap
```

2.7 Editing the database content with the new GUI (since 1.8.0)

```
UEVaultManager edit -db "D:/scraping/assets.db"
```

Note that the folder `D:/scraping` is set in `<config folder>/config_gui.ini`. By default this folder is located in the application installation folder (`<python folder>/Lib/site-packages/UEVaultManager`).

CHAPTER

THREE

HOW TO RUN/INSTALL

3.1 Requirements

- Windows (8.1+), Linux, or macOS (12.0+)
  - 32-bit operating systems are not supported
- Python 3.9 or newer
- PyPI packages:
  - `requests`
  - *beautifulsoup4*, *Pillow*, `tkinter`, *pandas*, *pandastable* and their dependencies for the GUI
  - (optional) `setuptools` and `wheel` for setup/building
  - (optional but recommended) *pywebview* for webview-based login

  Note: We describe how to do things for Ubuntu 22.04. But it should be similar for other Linux distributions and macOS.

  We only make tests and installations on Windows 11, Ubuntu 22.04 (WSL) and Pop!OS 22.04. So we can't guarantee that it will work on other OS.

3.2 Prerequisites

- Install python3.9
- You can manually install all the python packages listed before, but it will be done when running the command `pip install .` as indicated bellow.
- The tkinter package is not installed by default on Ubuntu and won't be installed by pip. So you need to install it manually with the command `sudo apt install python3-tk`.
- Be sure that pip is installed by running
  - for Linux or macOS (12.0+): `sudo apt install python3-pip` or `python -m ensurepip` or `python3 -m ensurepip` (depending on you os version)
  - for Windows: `python -m ensurepip`
- To prevent problems with permissions during installation, please upgrade your `pip` by running `python -m pip install -U pip --user`.

  Tip: You may need to replace *python* in the above command with *python3* on Linux/macOS.

3.3 Directly from the repo

3.3.1 Windows example

1. First install the Python language (3.9 minimal version required) as explained on the official python website
2. create a folder for storing the source files
3. open a command prompt or a terminal from this folder.
4. run the following commands:

```
git clone https://github.com/LaurentOngaro/UEVaultManager.git
cd UEVaultManager
pip install .
```

3.3.2 Ubuntu 22.04 example

Ubuntu standard repositories include everything needed to install UEVaultManager:

```
sudo apt install python3 python3-requests python3-setuptools-git
sudo apt install python3-tk
git clone https://github.com/LaurentOngaro/UEVaultManager.git
cd UEVaultManager
pip install .
```

If the `UEVaultManager` executable is not available after installation, you may need to configure your *PATH* correctly. You can do this by running the command:

```
echo 'export PATH=$PATH:~/.local/bin' >> ~/.profile && source ~/.profile
```

3.4 Direct installation (any)

3.4.1 Python Package on pypi (any)

```
pip install UEVaultManager
```

3.4.2 Windows Binaries from repos (since 1.6.2)

Download the *UEvm.exe* binary from the latest release and move it to somewhere in your path. The simpliest way is to put the executable into your windows folder (aka. *C:Windows*). But you can also create a folder and add the folder to your path.

The Windows executable was created with PyInstaller and will run standalone even without python being installed.

Note the executable is not signed, so you could get a warning from Windows SmartScreen when you run it. The executable will be decompressed in your temp folder and run from there. **So the first runs will be slow**.

CHAPTER

FOUR

USAGE

```
usage: UEVaultManager [-h] [-H] [-d] [-y] [-V] [-c <file>] [-J] [-A <seconds>] <command>␣
↪...

  exemple:

    UEVaultManager list --csv -fc "plugin" -o "D:\testing\list.csv"

    Will list all the assets of the marketplace that have "plugin" it their category␣
↪field (on the marketplace) and save the
    result using a csv format into the "D:\testing\list.csv" file

optional arguments:
  -h, --help                  Show this help message and exit
  -H, --full-help             Show full help (including individual command help)
  -d, --debug                 Set loglevel to debug
  -y, --yes                   Default to yes for all prompts
  -V, --version               Print version and exit
  -c, --config-file <file>    Overwrite the default configuration file name to use
  -J, --pretty-json           Pretty-print JSON. Improve readability
  -A, --api-timeout <seconds> Connection and read timeout API HTTP request (default: 7␣
↪seconds for each)
  -g, --gui                   Display the help in a windows instead of using the console

Commands:
    <command>
     auth             Authenticate with the Epic Games Store
     cleanup          Remove old temporary, metadata, and manifest files
     info             Prints info about specified Asset or manifest
     list             List the assets you OWNED (and only them). The process could take␣
↪some time.
     list-files       List files in manifest
     status           Show UEVaultManager status information. Will update the assets␣
↪list and could take some time.
     edit             Display a GUI to Edit the file that contains a list of assets.␣
↪Mainly use in conjunction with the list command that could
                        produce a list of assets in a file.
     scrap            Will use the EPIC API to retreive the data of ALL THE AVAILABLE␣
↪assets in the EPIC marketplace (including the ones you owned)
                        and store them in an sqlite database file. The process could␣
↪take some time.
```

(continues on next page)

(continued from previous page)

```
    install          Download and install or not an asset by name or manifest URI in a␣
↪project Folder.

Individual command help:

Command: auth
  usage: UEVaultManager auth [-h] [--import] [--code <exchange code>] [--token <exchange␣
↪token>]
                             [--sid <session id>] [--delete] [--disable-webview]

  optional arguments:
    -h, --help                  Show this help message and exit
    --import                    Import Epic Games Launcher authentication data (logs out␣
↪of EGL)
    --code <authorization code> Use specified authorization code instead of interactive␣
↪authentication
    --token <exchange token>    Use specified exchange token instead of interactive␣
↪authentication
    --sid <session id>          Use specified session id instead of interactive␣
↪authentication
    --delete                    Remove existing authentication (log out)
    --disable-webview           Do not use embedded browser for login


Command: cleanup
  usage: UEVaultManager cleanup [-h] [-cs] [-cc] [-g]

  optional arguments:
    -h,  --help                  Show this help message and exit
    -cs, --delete-scraping-data  Also delete scraping data files. They are kept by␣
↪default
    -cc, --delete-cache-data     Also delete image asset previews. They are usefull and␣
↪should be kept. They are kept by default
    -g,  --gui                   Display the output in a windows instead of using the␣
↪console


Command: info
  usage: UEVaultManager info [-h] [--offline] [--json] [-a] [-g] <Asset Name/Manifest␣
↪URI>

  positional arguments:
    <Asset Name/Manifest URI> Asset Name or manifest path/URI

  optional arguments:
    -h, --help              Show this help message and exit
    --offline               Only print info available offline. It will use files saved␣
↪previously, do not log in
    --json                  Output information in JSON format
    -a, --all               Display all the information even if non-relevant for an asset
    -g, --gui               Display the output in a windows instead of using the console
```

(continues on next page)

(continued from previous page)

```
Command: list
  usage: UEVaultManager list [-h] [--csv] [--tsv] [--json] [-f] [--offline]
                         [-fc <text_to_search>] [-o <file_name_with_path>]
                         [-g]

  optional arguments:
    -h,  --help             Show this help message and exit
    --csv                   List assets in CSV format
    --tsv                   List assets in TSV format
    --json                  List assets in JSON format
    -f,  --force-refresh    Force a refresh of all asset metadata. It could take some␣
↪time ! If not forced, the cached data will be used
    --offline               Only print info available offline. It will use files saved␣
↪previously, do not log in
    -fc, --filter-category  Filter assets by category. Search against the asset category␣
↪in the marketplace. Search is case-insensitive
                              and can be partial
    -o,  --output <file>    The file name (with path) where the list should be written to
    -g,  --gui              Display additional informations using gui elements like␣
↪dialog boxes or progress window


Command: list-files
  usage: UEVaultManager list-files [-h] [--manifest <url>] [--csv] [--tsv] [--json]
                         [--hashlist] [-g] [<Asset Name>]

  positional arguments:
    <Asset Name>          Name of the asset

  optional arguments:
    -h, --help            Show this help message and exit
    --manifest <url>      Manifest URL or path to use instead of the CDN one
    --csv                 Output in CSV format
    --tsv                 Output in TSV format
    --json                Output in JSON format
    --hashlist            Output file hash list in hashcheck/sha1sum -c compatible format
    -g, --gui             Display the output in a windows instead of using the console


Command: status
  usage: UEVaultManager status [-h] [--offline] [--json] [-g]

  optional arguments:
    -h, --help            Show this help message and exit
    --offline             Only print offline status information, do not login
    --json                Show status in JSON format
    -g, --gui             Display the output in a windows instead of using the console


Command: edit
  usage: UEVaultManager edit [-h] [--input] [--database]
```

(continues on next page)

(continued from previous page)

```
  optional arguments:
    -h,  --help            Show this help message and exit
    -i,  --input <file>    The file name (with path) where the list should be read from␣
↪(it exludes the --database option)
    -db, --database <file> The sqlite file name (with path) where the list should be␣
↪read from (it exludes the --input option)

Command: scrap
  usage: UEVaultManager scrap [-h] [-f] [--offline] [-g]

  optional arguments:
    -h, --help            Show this help message and exit
    -f, --force-refresh   Force a refresh of all asset metadata. It could take some time␣
↪! If not forced, the cached data in json files will be used
    --offline             Use previous saved data files (json) instead of scapping and␣
↪new data, do not log in
    -fc, --filter-category  Filter assets by category. Search against the asset category␣
↪in the marketplace. Search is case-insensitive
                              and can be partial
    -g, --gui             Display the output in a windows instead of using the console

Command: install
  usage: UEVaultManager install [-h] [...see arguments bellow...] [<Asset Name>]

  positional arguments:
    <Asset Name>                   Name of the asset

  optional arguments:
    -h,  --help                    Show this help message and exit
    -dp, --download-path <path>    Path where the Asset will be downloaded. If empty,␣
↪the Epic launcher Vault cache will be used.
    -f,  --force-refresh           Force a refresh of all asset's data. It could take␣
↪some time ! If not forced, the cached data will be used
    -vc, --vault-cache             Use the vault cache folder to store the downloaded␣
↪asset. It uses Epic Game Launcher setting to get this value. In that case, the␣
↪download_path option will be ignored
    -c,  --clean-dowloaded-data    Delete the folder with dowloaded data. Keep the␣
↪installed version if it has been installed.
    --max-shared-memory <Mib>      Maximum amount of shared memory to use (in MiB),␣
↪default: 1 GiB
    --max-workers <workers>        Maximum amount of download workers, default: min(2 *␣
↪CPUs, 16)
    --manifest <url>               Manifest URL or path to use instead of the CDN one (e.
↪g. for downgrading)
    --base-url <url>               Base URL to download from (e.g. to test or switch to␣
↪a different CDNs)
    --download-only, --no-install  Do not install the Asset after download
    -r,  --reuse-last-install      If the asset has been previouly installed, the␣
↪installation folder will be reused. In that case, the install-path option will be␣
↪ignored
    --enable-reordering            Enable reordering optimization to reduce RAM␣
↪requirements during download (may have adverse results for some titles
```

(continues on next page)

(continued from previous page)

```
    --timeout                       Connection and read timeout for downloader (default:␣
↪7 seconds for each)
    --preferred-cdn <cdn>           Set the hostname of the preferred CDN to use when␣
↪available
```

CHAPTER

FIVE

CONFIGURATION

5.1 Config folder

Configuration file, log files and results files are stored by default in data folders in the user home directory.

The location is:

- for Linux: `~/.config/UEVaultManager/`
- for Windows: `C:\users\<you_login_name>\.config\UEVaultManager\`

5.2 Config files

5.2.1 For the Cli Application settings

UEVaultManager supports some settings in its config file `<config folder>/config.ini`:

This is an example of this file content and the settings you can change:

```
[UEVaultManager]
;Set to True to start the Application in Edit mode (since v1.4.4) with the GUI
start_in_edit_mode = False
;Set to True to disable the automatic update check
disable_update_check = False
; Set to True to disable the notice about an available update on exit
disable_update_notice = False
; Create a backup of the output file (when using the --output option) suffixed by a␣
↪timestamp before creating a new file
create_output_backup = True
; Set to True to create a backup of the log files that store asset analysis. It is␣
↪suffixed by a timestamp
create_log_backup = True
; Set to True to print more information during long operations
verbose_mode = False
; File name (and path) to log issues with assets when running the list or scrap commands
; use "~/" at the start of the filename to store it relatively to the user directory
ignored_assets_filename_log = ~/.config/ignored_assets.log
notfound_assets_filename_log = ~/.config/notfound_assets.log
scan_assets_filename_log = ~/.config/scan_assets.log
scrap_assets_filename_log = ~/.config/scrap_assets.log
```

(continues on next page)

(continued from previous page)

```
; Minimal unreal engine version to check for obsolete assets (default is 4.26)
engine_version_for_obsolete_assets = 4.26
```

5.2.2 For the GUI settings

Since version 1.6.0, UEVaultManager also supports some settings specific to the new GUI, in a config file <config folder>/config_gui.ini:

This is an example of this file content and the settings you can change:

```
[UEVaultManager]
;List of Folders to scan for assets. Their content will be added to the list
folders_to_scan = ["G:/Assets/pour UE/01 Acquis", "G:/Assets/pour UE/00 A trier"]
;Set to True to print debug information (GUI related only)
debug_mode = False
;Set to True to use multiple threads when scraping/grabbing data for UE assets
use_threads = True
;timeout in second when scraping several assets in once. This value should not be too␣
↪low to limit timeout issues and scraping cancellation.
timeout_for_scraping = 30
;Number of grouped assets to scrap with one url. Since 2023-10-31 a value bigger than 75␣
↪will be refused by UE API
scraped_assets_per_page = 75
;Set to True to re-open the last file at startup if no input file is given
reopen_last_file = True
;Set to True to speed the update process by not updating the metadata files. FOR TESTING␣
↪ONLY
never_update_data_files = False
;Set to True to enable cell coloring depending on its content.It could slow down data␣
↪and display refreshing
use_colors_for_data = True
;Set to True to check and clean invalid asset folders when scraping or rebuilding data␣
↪for UE assets
check_asset_folders = True
;Set to True to browse for a folder when adding a new row. If false, an empty row will␣
↪be added
browse_when_add_row = True
;Number of Rows displayed or scraped per page.If this value is changed all the scraped␣
↪files must be updated to match the new value
rows_per_page = 37
;Number of backup files version to keep in the folder for backups. The oldest will be␣
↪deleted. Set to 0 to keep all the backups
backup_files_to_keep = 30
;Delay in seconds when image cache will be invalidated. Default value represent 15 days
image_cache_max_time = 1296000
;Folder (relative or absolute) to store images for assets
asset_images_folder = K:/UE/UEVM/asset_images
;Folder (relative or absolute) to store the scraped files for the assets in markeplace
scraping_folder = K:/UE/UEVM/scraping
;Folder (relative or absolute) to store result files to read and save data from
results_folder = K:/UE/UEVM/results
```

(continues on next page)

(continued from previous page)

```
;Minimal score required when looking for an url file comparing to an asset name. MUST BE␣
↪LOWERCASE
;minimal_fuzzy_score_by_name = {"default": 70, "brushify": 90, "elite_landscapes": 90,
↪"realistic landscapes": 100, "girl modular": 90}
minimal_fuzzy_score_by_name = {"default": 70, "brushify": 90, "elite_landscapes": 90,
↪"realistic landscapes": 100, "girl modular": 90, "mw dune desert landscape": 90}
;The name of the groups where the selected rows can be added to.
group_names = ["group1", "group2", "group3"]
;The name of the current group where the selected rows can be added to.
current_group_name = group2
;X position of the main windows. Set to 0 to center the window. Automatically saved on␣
↪quit
x_pos = -1915
;Y position of the main windows. Set to 0 to center the window. Automatically saved on␣
↪quit
y_pos = 13
;Width of the main windows. Automatically saved on quit
width = 1889
;Height of the main windows. Automatically saved on quit
height = 972
;File name of the last opened file. Automatically saved on quit
last_opened_file = K:\UE\UEVM\scraping\assets.db
;The last opened Folder name. Automatically saved when browsing a folder
last_opened_folder = G:\Assets\pour UE\02 Warez\Characters\Female\FurryS1 Fantasy Warrior
;The last opened project name. Automatically saved when browsing a project folder
last_opened_project = U:\UE_Big\UE_BigProjets\_EmptyForInstallTestsBBB
;The last opened Folder name. Automatically saved when browsing an engine folder
last_opened_engine = R:\UnrealEngine\UE_5.1
;The last opened filter file name.Automatically saved when loading a filter.Leave empty␣
↪to load no filter at start.Contains the file name only, not the path
last_opened_filter = landscape_owned_or_local_not_brushify.json
;List of columns names that will be hidden when applying columns width. Note that the
↪"Index_copy" will be hidden by default
hidden_column_names = ["Uid", "Release info"]
;Infos about columns of the table in SQLITE mode. Automatically saved on quit. Leave␣
↪empty for default
;column_infos_sqlite = {"Owned": {"width": 57, "pos": 0}, "App name": {"width": 222, "pos
↪": 1}, "Category": {"width": 112, "pos": 2}, "Comment": {"width": 265, "pos": 3},
↪"Description": {"width": 205, "pos": 4}, "Discount price": {"width": 61, "pos": 5},
↪"Origin": {"width": 641, "pos": 6}, "Tags": {"width": 228, "pos": 7}, "Discount␣
↪percentage": {"width": 58, "pos": 8}, "Review": {"width": 54, "pos": 9}, "Discounted":
↪{"width": 71, "pos": 10}, "Is new": {"width": 48, "pos": 11}, "Free": {"width": 44,
↪"pos": 12}, "Obsolete": {"width": 62, "pos": 13}, "Must buy": {"width": 59, "pos": 14},
↪ "Added manually": {"width": 44, "pos": 15}, "Grab result": {"width": 79, "pos": 16},
↪"Price": {"width": 50, "pos": 17}, "Asset_id": {"width": 174, "pos": 18}, "Review count
↪": {"width": 83, "pos": 19}, "Can purchase": {"width": -1, "pos": 20}, "Status": {
↪"width": 56, "pos": 21}, "Old price": {"width": 59, "pos": 22}, "Developer": {"width":␣
↪-1, "pos": 23}, "Stars": {"width": 42, "pos": 24}, "Test result": {"width": 69, "pos":␣
↪25}, "Alternative": {"width": -1, "pos": 26}, "Custom attributes": {"width": 105, "pos
↪": 27}, "Downloaded size": {"width": 82, "pos": 28}, "Page title": {"width": 161, "pos
↪": 29}, "Image": {"width": 50, "pos": 30}, "Url": {"width": 44, "pos": 31}, "Date added
↪": {"width": 75, "pos": 32}, "Creation date": {"width": 86, "pos": 33}, "Update date":
```

(continues on next page)

(continued from previous page)

```
↪{"width": 79, "pos": 34}, "Asset slug": {"width": 65, "pos": 35}, "Installed folders":
↪{"width": 150, "pos": 36}, "Uid": {"width": 2, "pos": 37}, "Supported versions": {
↪"width": 2, "pos": 38}, "Release info": {"width": 2, "pos": 39}, "Index copy": {"pos":␣
↪40, "width": 2}}
column_infos_sqlite = {"Owned": {"width": 57, "pos": 0}, "App name": {"width": 222, "pos
↪": 1}, "Category": {"width": 112, "pos": 2}, "Comment": {"width": 212, "pos": 3},
↪"Description": {"width": 306, "pos": 4}, "Discount price": {"width": 61, "pos": 5},
↪"Origin": {"width": 255, "pos": 6}, "Tags": {"width": 228, "pos": 7}, "Discount␣
↪percentage": {"width": 58, "pos": 8}, "Review": {"width": 54, "pos": 9}, "Discounted":
↪{"width": 71, "pos": 10}, "Is new": {"width": 48, "pos": 11}, "Free": {"width": 44,
↪"pos": 12}, "Obsolete": {"width": 62, "pos": 13}, "Must buy": {"width": 59, "pos": 14},
↪ "Added manually": {"width": 44, "pos": 15}, "Grab result": {"width": 79, "pos": 16},
↪"Price": {"width": 50, "pos": 17}, "Asset_id": {"width": 174, "pos": 18}, "Review count
↪": {"width": 83, "pos": 19}, "Can purchase": {"width": -1, "pos": 20}, "Status": {
↪"width": 56, "pos": 21}, "Old price": {"width": 59, "pos": 22}, "Developer": {"width":␣
↪-1, "pos": 23}, "Stars": {"width": 42, "pos": 24}, "Test result": {"width": 69, "pos":␣
↪25}, "Alternative": {"width": -1, "pos": 26}, "Custom attributes": {"width": 105, "pos
↪": 27}, "Downloaded size": {"width": 82, "pos": 28}, "Page title": {"width": 161, "pos
↪": 29}, "Image": {"width": 50, "pos": 30}, "Url": {"width": 44, "pos": 31}, "Date added
↪": {"width": 75, "pos": 32}, "Creation date": {"width": 86, "pos": 33}, "Update date":
↪{"width": 79, "pos": 34}, "Asset slug": {"width": 65, "pos": 35}, "Installed folders":
↪{"width": 150, "pos": 36}, "Uid": {"width": 2, "pos": 37}, "Supported versions": {
↪"width": 2, "pos": 38}, "Release info": {"width": 2, "pos": 39}, "Index copy": {"pos":␣
↪40, "width": 2}}
;Infos about columns of the table in FILE mode. Automatically saved on quit. Leave empty␣
↪for default
column_infos_file = {"Owned": {"width": 57, "pos": 0}, "App name": {"width": 222, "pos":␣
↪1}, "Category": {"width": 112, "pos": 2}, "Comment": {"width": 265, "pos": 3},
↪"Description": {"width": 205, "pos": 4}, "Discount price": {"width": 61, "pos": 5},
↪"Origin": {"width": 311, "pos": 6}, "Tags": {"width": 228, "pos": 7}, "Discount␣
↪percentage": {"width": 58, "pos": 8}, "Review": {"width": 54, "pos": 9}, "Discounted":
↪{"width": 71, "pos": 10}, "Is new": {"width": 48, "pos": 11}, "Free": {"width": 44,
↪"pos": 12}, "Obsolete": {"width": 62, "pos": 13}, "Must buy": {"width": 59, "pos": 14},
↪ "Added manually": {"width": 44, "pos": 15}, "Grab result": {"width": 79, "pos": 16},
↪"Price": {"width": 50, "pos": 17}, "Asset_id": {"width": 174, "pos": 18}, "Review count
↪": {"width": 83, "pos": 19}, "Can purchase": {"width": -1, "pos": 20}, "Status": {
↪"width": 56, "pos": 21}, "Old price": {"width": 59, "pos": 22}, "Developer": {"width":␣
↪-1, "pos": 23}, "Stars": {"width": 42, "pos": 24}, "Test result": {"width": 69, "pos":␣
↪25}, "Alternative": {"width": -1, "pos": 26}, "Custom attributes": {"width": 105, "pos
↪": 27}, "Downloaded size": {"width": 82, "pos": 28}, "Page title": {"width": 161, "pos
↪": 29}, "Image": {"width": 50, "pos": 30}, "Url": {"width": 44, "pos": 31}, "Date added
↪": {"width": 75, "pos": 32}, "Creation date": {"width": 86, "pos": 33}, "Update date":
↪{"width": 79, "pos": 34}, "Asset slug": {"width": 65, "pos": 35}, "Installed folders":
↪{"width": 150, "pos": 36}, "Uid": {"width": 2, "pos": 37}, "Supported versions": {
↪"width": 2, "pos": 38}, "Release info": {"width": 2, "pos": 39}, "App title": {"width
↪": 52, "pos": 40}, "urlSlug": {"width": 2, "pos": 41}, "Index copy": {"pos": 42, "width
↪": 2}}
;DEV ONLY. NO CHANGE UNLESS YOU KNOW WHAT YOU ARE DOING. Column name to sort the assets␣
↪from the database followed by ASC or DESC (optional).
;assets_order_col = asset_id ASC
assets_order_col = date_added DESC
;DEV ONLY. NO CHANGE UNLESS YOU KNOW WHAT YOU ARE DOING. Value that can be changed in␣
```

(continues on next page)

(continued from previous page)

```
↪live to switch some behaviours whithout quitting.
testing_switch = 0
```

Note that some other settings for the new GUI are managed by a dedicated python file <python install folder>/<source folder of the package>/tkgui/modules/GuiSettingsClass.py

For instance, the location is:

- for Linux: ~/.local/lib/python3.10/site-packages/UEVaultManager/tkgui/modules/GuiSettingsClass.py
- for Windows: c:\python3.10\site-packages\UEVaultManager\tkgui\modules\GuiSettingsClass.py

The final path can depend on your installation.

CHAPTER

SIX

OUTPUT FORMAT AND FILE

6.1 Log files and debug

3 different log files could be used during the process Use the config file to set their file name (and path). If a file name is missing, empty or set to '' the corresponding log feature will be disabled.

- ignored assets file log
  - file is defined by the setting: *ignored_assets_filename_log* (default is `~/.config/ignored_assets.log`)
  - each asset listed in the file has been ignored during the process. Possible reasons are: asset filtered by category (list with -fc option)
- not found assets log
  - file is defined by the setting: *notfound_assets_filename_log* (default is `~/.config/notfound_assets.log`)
  - each asset listed in the file has not been found during the scraping process. Possible reasons are: invalid url, obsolete or removed from the marketplace
- folder scanning for assets log
  - file is defined by the setting: *scan_assets_filename_log* (default is `~/.config/scan_assets_filename_log.log`)
  - each scanned folder is listed here whith the result of the scan
- assets scraping log
  - file is defined by the setting: *scrap_assets_filename_log* (default is `~/.config/scan_assets.log`)
  - each scraped asset is listed here whith the result of the scan

6.2 The output file

The result of the listing can be displayed on the console where the application has been launched. This is done by default. But it can also be saved in a csv or a json file for a future use.

The script use a (hardcoded) boolean value to know if the content of the field is *protected* and must be preserved before overwriting an existing output file.

This feature goal is to avoid overwriting data that could have been manually changed by the user in the output file between successive runs. As it, if the user manually change the content of some data in the file, by adding a comment for instance, this data WON'T be overwritten. Also Note that if *create_output_backup = true* is set in the config file, the application will create a backup of the output file suffixed by a timestamp before overwriting the result file.

These are the fields (or column headings) that will be written in that order into the CSV file (or the names of the fields ins the Json file). The value is False if its content is not preserved, and True if it is preserved (and can be used to store persistant data).

These value are defined by the csv_sql_fields variable at the beginning of the core.py file:

```
csv_sql_fields = {
    # fields mapping from csv to sql
    # key: csv field name, value: {sql name, state }
    # some field are intentionnaly duplicated because
    #   several CSV fields could come from a same database field
    #   a csv field with this name must exist to get the value
    'Asset_id': {
        'sql_name': 'asset_id',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.STR
    },
    'App name': {
        'sql_name': 'title',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.STR
    },
    'App title': {
        # intentionnaly duplicated
        'sql_name': 'title',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.STR
    },
    'Category': {
        'sql_name': 'category',
        'state': CSVFieldState.CHANGED,
        'field_type': CSVFieldType.LIST
    },
    'Review': {
        'sql_name': 'review',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.FLOAT
    },
    'Review count': {
        # not in "standard/result" csv file
        'sql_name': 'review_count',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.INT
    },
    'Developer': {
        'sql_name': 'author',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.STR
    },
    'Description': {
        'sql_name': 'description',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.TEXT
    },
```

(continues on next page)

(continued from previous page)

```
    'Status': {
        'sql_name': 'status',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.STR
    },
    'Discount price': {
        'sql_name': 'discount_price',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.FLOAT
    },
    'Discount percentage': {
        'sql_name': 'discount_percentage',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.INT
    },
    'Discounted': {
        'sql_name': 'discounted',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.BOOL
    },
    'Is new': {
        # not in "standard/result" csv file
        'sql_name': 'is_new',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.BOOL
    },
    'Free': {
        # not in "standard/result" csv file
        'sql_name': 'free',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.BOOL
    },
    'Can purchase': {
        # not in "standard/result" csv file
        'sql_name': 'can_purchase',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.BOOL
    },
    'Owned': {
        'sql_name': 'owned',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.BOOL
    },
    'Obsolete': {
        'sql_name': 'obsolete',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.BOOL
    },
    'Supported versions': {
        'sql_name': 'supported_versions',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.STR
```

(continues on next page)

(continued from previous page)

```
    },
    'Grab result': {
        'sql_name': 'grab_result',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.LIST
    },
    'Price': {
        'sql_name': 'price',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.FLOAT
    },
    'Old price': {
        'sql_name': 'old_price',
        'state': CSVFieldState.CHANGED,
        'field_type': CSVFieldType.FLOAT
    },
    # ## User Fields
    'Comment': {
        'sql_name': 'comment',
        'state': CSVFieldState.USER,
        'field_type': CSVFieldType.TEXT
    },
    'Stars': {
        'sql_name': 'stars',
        'state': CSVFieldState.USER,
        'field_type': CSVFieldType.INT
    },
    'Must buy': {
        'sql_name': 'must_buy',
        'state': CSVFieldState.USER,
        'field_type': CSVFieldType.BOOL
    },
    'Test result': {
        'sql_name': 'test_result',
        'state': CSVFieldState.USER,
        'field_type': CSVFieldType.STR
    },
    'Installed folders': {
        'sql_name': 'installed_folders',
        'state': CSVFieldState.USER,
        'field_type': CSVFieldType.STR
    },
    'Alternative': {
        'sql_name': 'alternative',
        'state': CSVFieldState.USER,
        'field_type': CSVFieldType.STR
    },
    'Origin': {
        'sql_name': 'origin',
        'state': CSVFieldState.CHANGED,
        'field_type': CSVFieldType.STR
    },
```

(continues on next page)

(continued from previous page)

```
    'Added manually': {
        'sql_name': 'added_manually',
        'state': CSVFieldState.USER,
        'field_type': CSVFieldType.BOOL
    },
    # ## less important fields
    'Custom attributes': {
        # not in "standard/result" csv file
        'sql_name': 'custom_attributes',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.STR
    },
    'Page title': {
        'sql_name': 'page_title',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.STR
    },
    'Image': {
        'sql_name': 'thumbnail_url',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.STR
    },
    'Url': {
        'sql_name': 'asset_url',
        'state': CSVFieldState.CHANGED,
        'field_type': CSVFieldType.STR
    },
    'Compatible versions': {
        # not in database
        'sql_name': None,
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.STR
    },
    'Date added': {
        'sql_name': 'date_added',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.DATETIME
    },
    'Creation date': {
        'sql_name': 'creation_date',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.DATETIME
    },
    'Update date': {
        'sql_name': 'update_date',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.DATETIME
    },
    'UE version': {
        # not in database
        'sql_name': None,
        'state': CSVFieldState.NORMAL,
```

(continues on next page)

(continued from previous page)

```
        'field_type': CSVFieldType.STR
    },
    'Uid': {
        'sql_name': 'id',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.STR
    },
    # ## UE asset class field only
    'Namespace': {
        'sql_name': 'namespace',
        'state': CSVFieldState.ASSET_ONLY,
        'field_type': CSVFieldType.STR
    },
    'Catalog itemid': {
        'sql_name': 'catalog_item_id',
        'state': CSVFieldState.ASSET_ONLY,
        'field_type': CSVFieldType.STR
    },
    'Asset slug': {
        'sql_name': 'asset_slug',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.STR
    },
    'Currency code': {
        'sql_name': 'currency_code',
        'state': CSVFieldState.ASSET_ONLY,
        'field_type': CSVFieldType.STR
    },
    'Technical details': {
        'sql_name': 'technical_details',
        'state': CSVFieldState.ASSET_ONLY,
        'field_type': CSVFieldType.STR
    },
    'Long description': {
        'sql_name': 'long_description',
        'state': CSVFieldState.ASSET_ONLY,
        'field_type': CSVFieldType.TEXT
    },
    'Tags': {
        'sql_name': 'tags',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.STR
    },
    'Comment rating id': {
        'sql_name': 'comment_rating_id',
        'state': CSVFieldState.ASSET_ONLY,
        'field_type': CSVFieldType.STR
    },
    'Rating id': {
        'sql_name': 'rating_id',
        'state': CSVFieldState.ASSET_ONLY,
        'field_type': CSVFieldType.STR
```

(continues on next page)

(continued from previous page)

```
    },
    'Is catalog item': {
        'sql_name': 'is_catalog_item',
        'state': CSVFieldState.ASSET_ONLY,
        'field_type': CSVFieldType.BOOL
    },
    'Thumbnail': {
        # intentionnaly duplicated
        'sql_name': 'thumbnail_url',
        'state': CSVFieldState.ASSET_ONLY,
        'field_type': CSVFieldType.STR
    },
    'Release info': {
        'sql_name': 'release_info',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.STR
    },
    'Downloaded size': {
        'sql_name': 'downloaded_size',
        'state': CSVFieldState.NORMAL,
        'field_type': CSVFieldType.STR
    },
}
```

6.3 The individual json files

Each asset will also have its data saved in to different json files:

- for the all the assets available in the marketplace (including the owned ones):
- the folder <`Scraping folder`>`/assets`: contains a json file for each asset (identified by its *asset_id* is the asset has one) to store its metadata (get from a call to the epic API). The <Scraping folder> can be set in the <`config folder`>`/config_gui.ini` configuration file
- for the assets OWNED by the user
- the folder <`Scraping folder`>`/owned`: contains a json file for each asset (identified by its *asset_id* is the asset has one) to store its metadata (get from a call to the epic API). The <Scraping folder> can be set in the <`config folder`>`/config_gui.ini` configuration file

6.4 how to fix invalid search result during the scraping process

The INDIVIDUAL scraping process (i.e. click on the "Scrap" or "Scrap range" buttons some a text based search (partial and case-insensitive) can be used if the url of the asset is invalid. By default, only the first result of this search is taken as the corresponding asset. When the asset name, which must be converted to be used as a search keyword, is ambiguous,the search could provide several results or even a wrong result (an asset that don't correspond).

So, in that case, the asset page that is analyzed could be the bad one and grabbed data could be taken for the wrong asset.

To limit this error, a text comparison is done between the asset title in the metadata and the title in the asset page. If the values are different, its *Grab Result* field will contain a value different from NO_ERROR. Each value correspond

to a specific status code (see *possible values in the error Field*)

To fix that, the search of the correct url for the asset must be done and validated manually.

Once validated, the correct URL could be added into the result file, inside the Url field. As this field is marked as *USER*, it won't be overwritten on the next data update and will be used as a source url for the page to be grabbed instead of making a new search for the asset page.

Please Note that the user is responsable for respecting the attended format of the result file when modifying its content. Breaking its structure will probably result in losing the data the user has modified in the file when the application will be executed next time.

Making a backup before any manual modification is certainly a good idea. Using a tool (e.g. a linter) to check if the structure of the file (json or CSV) is still correct before running the application again is also a very good idea.

6.5 possible values in the error Field

The *Grab result* field of each asset contains a value that indicate how the process has run. These code are defined by the following enum at the beginning of the api/egs.py file:

```
class GrabResult(Enum):
    NO_ERROR = 0
    INCONSISTANT_DATA = 1
    PAGE_NOT_FOUND = 2
    CONTENT_NOT_FOUND = 3
    TIMEOUT = 4
    PARTIAL = 5  # when asset has been added when owned asset data only (less complete␣
↪that "standard" asset data)
    NO_APPID = 6  # no appid found in the data (will produce a file name like '_no_appId_
↪asset_1e10acc0cca34d5c8ff7f0ab57e7f89f
    NO_RESPONSE = 7  # the url does not return HTTP 200
```

CHAPTER

SEVEN

THE (NEW) GUI

Since the 1.1.0 version , the application provides a graphical user interface (GUI) based on the Tkinter library.

The GUI is designed to edit the result file of the *list* command of the application. I should be self explanatory. If not, please report a bug.

The GUI is also available as an option from some commands of the application. Usually, just add the '-g' or '–gui' option to the command arguments. Please read the *Usage* section to see what command are supported.

The 'edit' command always uses the new GUI, so no need to add the option.

7.1 Usage for editing

```
usage: UEVaultManager edit [-h] [--input]

optional arguments:
  -h, --help            Show this help message and exit
  -i, --input           The file name (with path) where the list should be read from

  exemple:

    UEVaultManager edit --input "D:\testing\list.csv"
```

Note: if you run the application from its sources, you can also start the new edit mode by executing the `./UEVaultManager/tkgui/main.py` file.

7.2 Use it as default starting mode

If you want to start the application without arguments or start it in edit mode by default (aka with the new GUI). you can set the line start_in_edit_mode=true in the configuration file. In this case, the application will use the default data file. If the file does not exists, a new one will be created and its content will be rebuilt from scratch.

7.3 Some Screenshots

Here some screen shot of windows used by the application. They could have evolved since the time this documentation was written.

7.3.1 The main window

It displays a Listing of all the assets by row, as a standard data table. You can use pagination or not, filter rows, edit cells …

Coloring is used to highlight the status of the asset or special cell values.

Rows can be selected and edited or filtered by a search string, a category or a status.

Double-clic on a cell to open the *edit cell window*.

Some fields are not editable, like the 'asset_id', other are editable directly by changing the value in the cell (mainly boolean and categorical values).

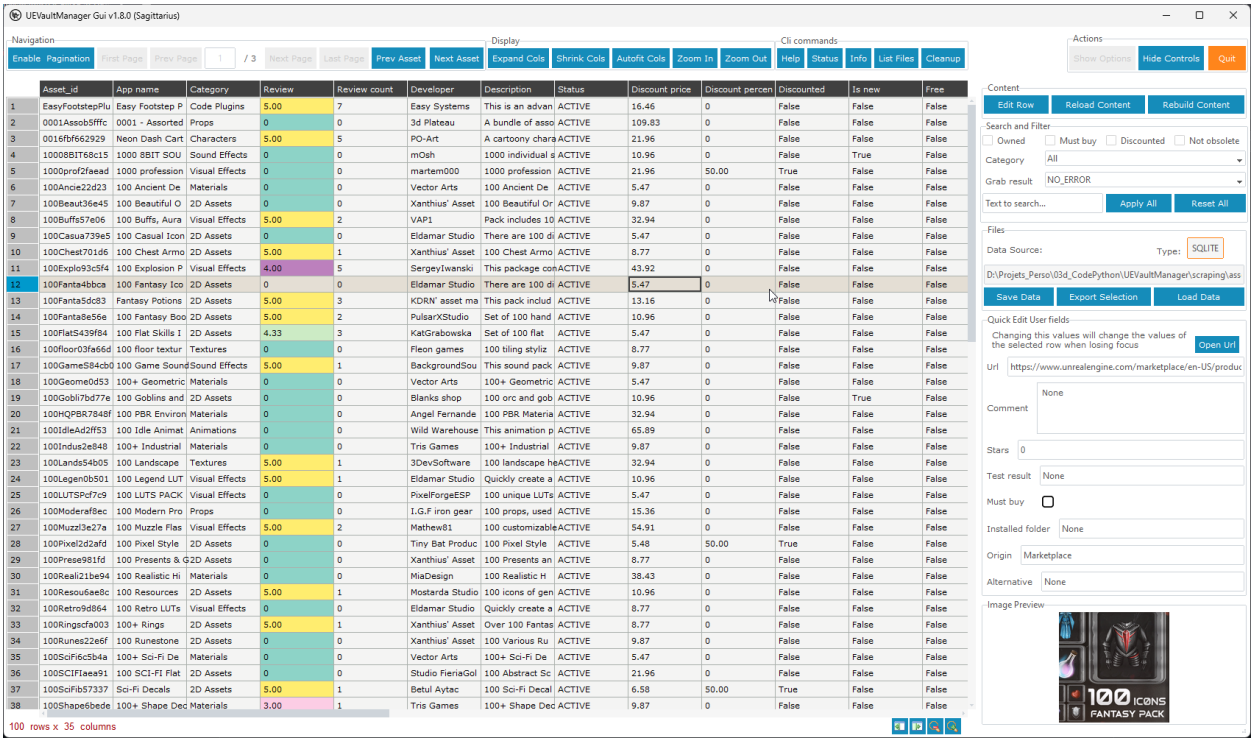
Users fields values can be quick edited by changing their value in the *Quick Edit User Fields* panel. Change will be applied on focus out of the editing widget.

A new file can be created from scratch or loaded from an existing file.

Data can be exported to a CSV file and saved to the current loaded file or to a new one.

Data can be rebuilt from the previous stored metadata files by clicking on the *Rebuild file content* button. It could take some time, so a progress window will be displayed and the process can be stopped.

Some commands can be executed from the toolbar and their result will be displayed in a Result window.

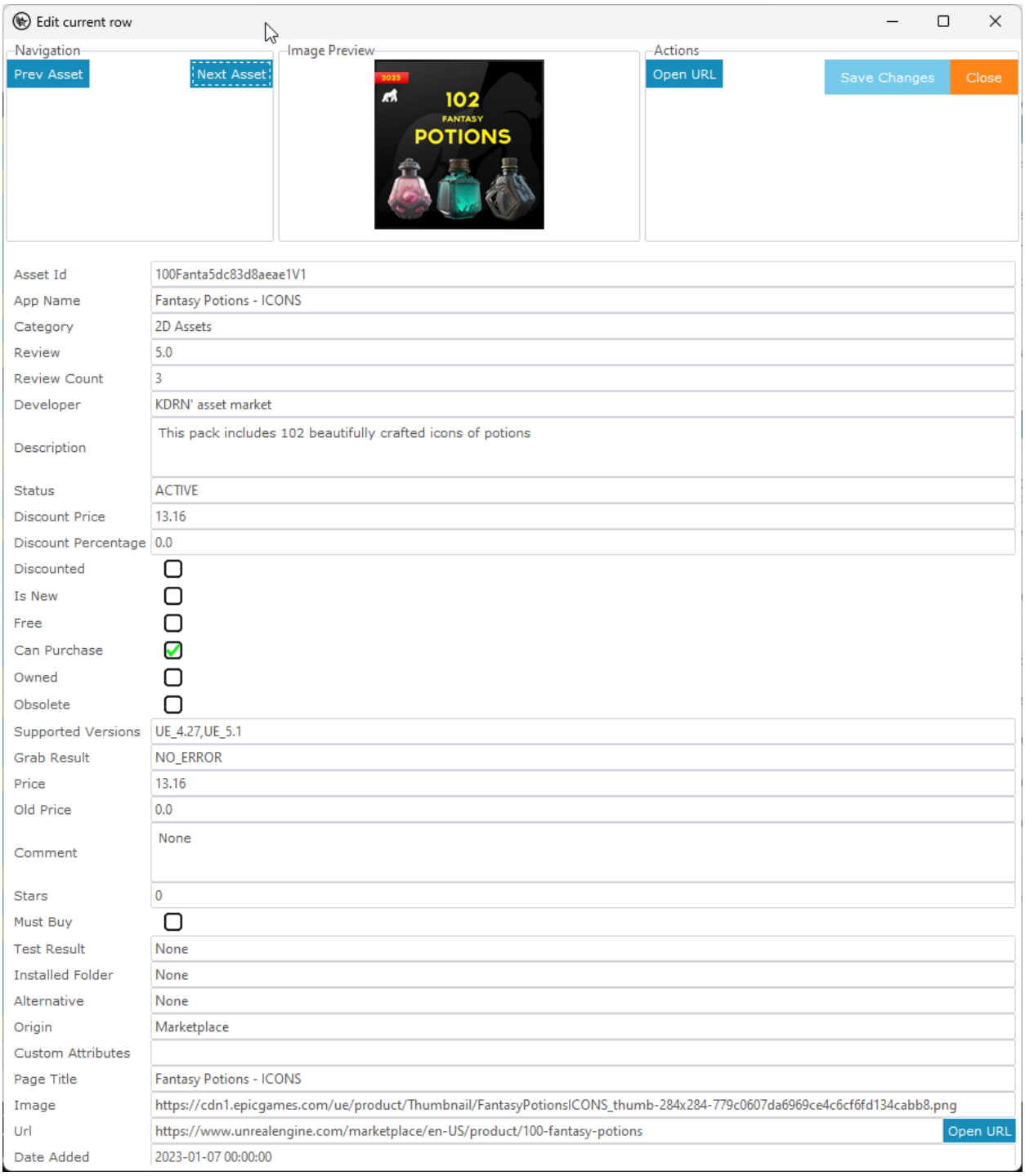


7.3.2 The Edit row window

It allows to edit the value of all the fields of a row.

Note that the data are raw and not formatted as in the main window, exception for boolean values that are displayed as checkboxes.

The changes made to a value must respect the initial format of the field to avoid errors on save.



7.3.3 The Edit cell window

It alloaw to change the value of a single cell of a row.

Note that the data are raw and not formatted as in the main window, exception for boolean values that are displayed as checkboxes.

The changes made to a value must respect the initial format of the field to avoid errors on save.