__init__.py

```
package_name = "EnneadTab"
version = "2.1"

import os
import traceback

import os
for module in os.listdir(os.path.dirname(__file__)):
    #print (module)
    if module == '__init__.py':
        continue
    if module in ["RHINO", "REVIT"]:
        __import__(module, locals(), globals())
        continue
    if module[-3:] != '.py':
        continue
    try:
        __import__(module[:-3], locals(), globals())
    except Exception as e:
        # to-do: add try because Rhino 8 traceback is not working peoperly. This
should be recheck in future Rhino 8.
        try:
            print ("Cannot import {} becasue\n\n{}".format(module,
traceback.format_exc()))
        except:
            print ("Cannot import {} becasue\n\n{}".format(module, e))
del module# delete this varible becaue it is refering to last item on the for
loop
```

ENVIRONMENT.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""Utility functions for checking the current application environment.
Sets environment variables and paths for EnneadTab."""

import os
import sys


IS_PY3 = sys.version.startswith("3")
IS_PY2 = not IS_PY3


# this is the repo folder if you are a developer, or EA_dist if you are a normal
user
ROOT = os.path.dirname(
    os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
)


INSTALLATION_FOLDER = os.path.join(ROOT, "Installation")


APP_FOLDER = os.path.join(ROOT, "Apps")
REVIT_FOLDER = os.path.join(APP_FOLDER, "_revit")
RHINO_FOLDER = os.path.join(APP_FOLDER, "_rhino")
PRIMARY_EXTENSION_NAME = "EnneaDuck"
REVIT_PRIMARY_EXTENSION = os.path.join(
    REVIT_FOLDER, "{}.extension".format(PRIMARY_EXTENSION_NAME)
)
```

```
REVIT_PRIMARY_TAB = os.path.join(REVIT_PRIMARY_EXTENSION, "Ennead.tab")
REVIT_LIBRARY_TAB = os.path.join(REVIT_PRIMARY_EXTENSION, "Ennead Library.tab")
REVIT_TAILOR_TAB = os.path.join(REVIT_PRIMARY_EXTENSION, "Ennead Tailor.tab")

L_DRIVE_HOST_FOLDER = "L:\\4b_Applied Computing"
DB_FOLDER = "{}\\EnneadTab-DB".format(L_DRIVE_HOST_FOLDER)
SHARED_DUMP_FOLDER = DB_FOLDER + "\\Shared Data Dump"


LIB_FOLDER = os.path.join(APP_FOLDER, "lib")
CORE_FOLDER = os.path.join(LIB_FOLDER, "EnneadTab")
IMAGE_FOLDER = os.path.join(CORE_FOLDER, "images")
AUDIO_FOLDER = os.path.join(CORE_FOLDER, "audios")
DOCUMENT_FOLDER = os.path.join(CORE_FOLDER, "documents")


EXE_PRODUCT_FOLDER = os.path.join(LIB_FOLDER, "ExeProducts")

DEPENDENCY_FOLDER = os.path.join(LIB_FOLDER, "dependency")
if IS_PY2:
    DEPENDENCY_FOLDER += "\\py2"
```

COLOR.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""Utilities for color manipulation and conversion"""

import random
import json

try:
    from System.Drawing import Color # pyright: ignore
    BLACK = Color.FromArgb(0,0,0)
    from colorsys import hsv_to_rgb, rgb_to_hsv

except:
    pass

# define before other customs import to avoid circular reference break in
ironpython.
# #cpython seem to be ok with this kind of circular ref
class TextColorEnum:
    Red = "red"
    Green = "green"
    Blue = "blue"
    Yellow = "yellow"
    Magenta = "magenta"
    Cyan = "cyan"
    White = "white"
ACCENT_COLOR = 70,70,70
PRIMARY_BACKGROUND = 100, 100, 100
DARKER_BACKGROUND = 70,70,70
PRIMARY_TEXT = 218,232,253


import ENVIRONMENT
if ENVIRONMENT.IS_REVIT_ENVIRONMENT:
    from Autodesk.Revit.DB import Color as DB_Color # pyright: ignore
if ENVIRONMENT.IS_RHINO_ENVIRONMENT:
    import Eto # pyright: ignore
import NOTIFICATION
import FOLDER
def from_rgb(r, g, b):
    """Generate a color object from rgb values.
```

```
    Args:
```

CONFIG.py

```
"""Get and set the global settings for EnneadTab."""


import os
import DATA_FILE


GLOBAL_SETTING_FILE = "setting_{}.sexyDuck".format(
    os.environ["USERPROFILE"].split("\\")[-1]
)


def get_setting(key, default_value=None):
    """If no key provided, will return the whole dict.
    Otherwise, return the default value of this key.

        key_default_value (tuple): (key, default value), a tuple of default
result, this is used to get the key of value looking for. If do not provide this
tuple, then return the raw while data"""

    data = DATA_FILE.get_data(GLOBAL_SETTING_FILE)
    return data.get(key, default_value)


def set_setting(key, value):
    """Set the key and value to the Revit UI setting.

    Args:
        key (str): The key of the setting.
        value (str): The value of the setting.
    """

    with DATA_FILE.update_data(GLOBAL_SETTING_FILE) as data:
        data[key] = value


# simply rename the addin file register file by
# add/remove .disabled at end of the .addin file
# note need to search for all valid version folder
def enable_revit_addin(addin):
    pass

    # reload pyrevit


def disable_revit_addin(addin):
    pass

    # reload pyrevit
```

CONTROL.py

```
pass
```

COPY.py

```
"""
The main purpose of this moudle to is to handle Rhino 8 situation.
Native shutil.copyfile will fail in some cases, so we use dotnet to copy the
file.

"""
try:
    import shutil
except:
    from System.IO import File



def copyfile(src, dst):
    try:
        # Attempt to copy the file using shutil.copyfile
        shutil.copyfile(src, dst)
    except Exception as e:
        copyfile_with_dotnet(src, dst)

def copyfile_with_dotnet(src, dst):
    try:
        File.Copy(src, dst, True)  # True to overwrite if exists
        return True
    except Exception as e:
        return False


if __name__ == "__main__":
    copyfile()
```

DATA_CONVERSION.py

```
"""Utilities for data conversions and comparisons."""

import ENVIRONMENT


class DataType:
    ElementId = "ElementId"
    Curve = "Curve"
    CurveLoop = "CurveLoop"
    Point3d = "Point3d"
    TableCellCombinedParameterData = "TableCellCombinedParameterData"
    XYZ = "XYZ"
    Double = "Double"


def list_to_system_list(list, type=DataType.ElementId, use_IList=False):
    """Convert a python list to a System collection List.
    In many occasions it is necessary to cast a python list to a .NET List
object

    Args:
        list (python list): _description_
```

```
        type (str, optional): the description for target data type. Defaults to
"ElementId".
        use_IList (bool, optional): Whether to use IList interface instead of
list instance. Defaults to False.

    Returns:
        System.Collections.Generic.List: The converted list object.
    """

    import System  # pyright: ignore

    if ENVIRONMENT.is_Revit_environment():
        from Autodesk.Revit import DB  # pyright: ignore
    if ENVIRONMENT.is_Rhino_environment():
        import Rhino  # pyright: ignore

    if use_IList:
        if type == DataType.CurveLoop:
            return System.Collections.Generic.IList[DB.CurveLoop](list)

        if type == DataType.Curve:
            return System.Collections.Generic.IList[DB.Curve](list)

        if type == DataType.TableCellCombinedParameterData:
            return
System.Collections.Generic.IList[DB.TableCellCombinedParameterData](
                list
            )

        return System.Collections.Generic.IList[type](list)

    if type == DataType.Point3d:
```

DATA_FILE.py

```
"""Utilities for writing and reading data to and from JSON files as well as
persistent sticky data."""

import shutil
import json
import io
import os
import traceback
from contextlib import contextmanager


import FOLDER


def _read_json_file_safely(filepath, use_encode=True,
create_if_not_exist=False):
    """Duplicate a JSON file then read it to avoid holding the file open status

    Args:
        filepath (str): The path of the file to read.
        use_encode (bool, optional): Might need encoding if there are Chinese
characters in the file. Defaults to False.
        create_if_not_exist (bool, optional): Create the file if it does not
exist. Defaults to False.

    Returns:
        dict: The contents of the file as a dictionary.
    """
    if not os.path.exists(filepath):
        return dict()
```

```
    local_path = FOLDER.get_EA_dump_folder_file("temp.sexyDuck")
    try:
        shutil.copyfile(filepath, local_path)
    except IOError:
        local_path = FOLDER.get_EA_dump_folder_file("temp_additional.sexyDuck")
        shutil.copyfile(filepath, local_path)

    content = _read_json_as_dict(local_path, use_encode, create_if_not_exist)
    return content


def _read_json_as_dict(filepath, use_encode=True, create_if_not_exist=False):
    """Get the data from a JSON file and return it as a dictionary.

    Args:
        filepath (str): The path of the file to read.
        use_encode (bool, optional): Might need encoding if there are Chinese
characters in the file. Defaults to False.
        create_if_not_exist (bool, optional): Create the file if it does not
exist. Defaults to False.

    Returns:
        dict: The contents of the file as a dictionary.
    """
    if create_if_not_exist:
        if not os.path.exists(filepath):
```

DOCUMENTATION.py

```
"""Utilities for showing tips and documentation for tools."""

import os
import random
import imp
import traceback


import ENVIRONMENT
import FOLDER
import USER
import OUTPUT



def get_text_path_by_name(file_name):
    """Get the full path of a text file in the documents library by its name.

    Args:
        file_name (str): _description

    Returns:
        str: Full path of the text file
    """
    path = "{}\\text\\{}".format(ENVIRONMENT.DOCUMENT_FOLDER, file_name)
    if os.path.exists(path):
        return path
    print ("A ha! {} is not valid or accessible. Better luck next
time.".format(path))



TIP_KEY = "__tip__"
SCOTT_TIPS = ["https://ei.ennead.com/post/3046/revit-legends-legend-components",
              "https://ei.ennead.com/post/2777/revit-short-subject-purge-cad-
```

```
before-linking-into-revit",
              "https://ei.ennead.com/post/3007/don-t-use-groups-as-single-
entities",
              "https://ei.ennead.com/post/2981/keeping-enough-c--drive-free-
space",
              "https://ei.ennead.com/post/2824/revit-short-subject-acc-
bim-360-requirements",
              "https://ei.ennead.com/post/2673/revit-short-subject-worksharing-
etiquette",
              "https://ei.ennead.com/post/31/revit-short-subject-design-
options",
              "https://ei.ennead.com/post/19/revit-short-subject-keyboard-
shortcuts",
              "https://ei.ennead.com/post/47/revit-short-subject-3d-view-
navigation",
              "https://ei.ennead.com/post/64/revit-short-subject-don-t-manually-
hide-elements",
              "https://ei.ennead.com/post/75/revit-short-subject-limit-use-of-
groups",
              "https://ei.ennead.com/post/99/revit-short-subject-the-cad-query-
tool",
              "https://ei.ennead.com/post/114/revit-short-subject-best-practice-
for-new-views"]

def show_scott_tip():

    """Show a random tip from Scott's EI posts."""
```

DUCK.py

```
"""the dancing call duck"""
import EXE


def quack ():
    EXE.try_open_app("EnneaDuck.exe")
```

EMAIL.py

```
"""This module is for sending email. It is a wrapper for the Emailer app."""

import time
import EXE
import DATA_FILE
import IMAGE
import EXE
import DATA_FILE
import USER
import ENVIRONMENT
import TIME
import SPEAK

if ENVIRONMENT.IS_REVIT_ENVIRONMENT:
    from REVIT import REVIT_APPLICATION


def email(
    receiver_email_list,
```

```
    body,
    subject="EnneadTab Auto Email",
    body_folder_link_list=None,
    body_image_link_list=None,
    attachment_list=None,
):
    """Send email using the Emailer app.

    Args:
        receiver_email_list (list): List of email addresses.
        body (str): Body of the email.
        subject (str, optional): Subject of the email. Defaults to "EnneadTab
Auto Email".
        body_folder_link_list (list, optional): List of folder links to be
included in the email body. Defaults to None.
        body_image_link_list (list, optional): List of image links to be
included in the email body. Defaults to None.
        attachment_list (list, optional): List of file paths to be attached to
the email. Defaults to None.
    """

    if not body:
        print("Missing body of the email.....")
        return

    if not receiver_email_list:
        print("missing email receivers....")
        return

    if isinstance(receiver_email_list, str):
        print("Prefer list but ok.")
        receiver_email_list = receiver_email_list.rstrip().split(";")

    body = body.replace("\n", "<br>")
```

EMOJI.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""Get emojis from the emoji library."""

import io
import random
import DOCUMENTATION

def get_all_emojis():
    """Get all emojis from the emoji library.

    Returns:
        list: List of emojis.
    """
    with io.open(DOCUMENTATION.get_text_path_by_name('_emoji_text.txt'), "r",
encoding = "utf8") as f:
        lines = f.readlines()
    return [x.replace("\n", "") for x  in lines if x != "\n"]


def pick_emoji_text():
    """Pick an emoji text from the displayed list and copy it to the clipboard.
    """
    lines = get_all_emojis()
    from pyrevit import forms
    sel = forms.SelectFromList.show(lines, select_multiple = False, title = "Go
wild")
    if not sel:
```

```
        return

    forms.ask_for_string(default = sel,
                         prompt = 'Copy below text to anywhere, maybe SheetName
or Schedule',
                         title = 'pick_emoji_text')


def random_emoji():
    """Pick a random emoji.
    """
    lines = get_all_emojis()

    random.shuffle(lines)
    return lines[0].replace("\n", "")
```

ENCOURAGING.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""Display encouraging messages."""

import io
import random
import textwrap


import NOTIFICATION
import DOCUMENTATION
import ENVIRONMENT
import CONFIG

def is_hate_encouraging():
    """Check if the user has enabled encouraging messages.

    Returns:
        bool: True if the user has enabled encouraging messages.
    """
    return not CONFIG.get_setting("radio_bt_popup_full", False)

def get_all_warming_quotes():
    """Get all encouraging quotes from the quote library.

    Returns:
        list: All encouraging quotes.
    """
    with io.open(DOCUMENTATION.get_text_path_by_name('_warming_quotes.txt'),
"r", encoding = "utf8") as f:
        lines = f.readlines()
    return [x.replace("\n", "") for x  in lines if x != "\n"]


def random_warming_quote():
    """Get a random encouraging quote from the quote library.

    Returns:
        str: A random encouraging quote
    """
    lines = get_all_warming_quotes()
    random.shuffle(lines)
    return lines[0].replace("\n", "")


def warming_quote():
    """Display a random encouraging quote.
```

```
    """
    quote = random_warming_quote()
```

ERROR_HANDLE.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""Defines the primary error handling function for EnneadTab."""

import traceback
import ENVIRONMENT
import FOLDER
import EMAIL
import USER
import TIME
import NOTIFICATION
import OUTPUT


def try_catch_error(is_silent=False, is_pass = False):
    """Decorator for catching exceptions and sending automated error log emails.

    Args:
        is_silent (bool, optional): If True, email will be sent but no e msg
will be visible to user. Defaults to False.
        is_pass (bool, optional): If True, the error will be ignored, user will
not be paused and no email will be sent. Defaults to False.
    """
    def decorator(func):
        def error_wrapper(*args, **kwargs):

            try:
                out = func(*args, **kwargs)
                return out
            except Exception as e:
                if is_pass:
                    return
                print_note(str(e))
                print_note("error_Wrapper func for EA Log -- Error: " + str(e))
                error_time = "Oops at
{}\n\n".format(TIME.get_formatted_current_time())
                try:
                    error = traceback.format_exc()
                except:
                    error = str(e)

                subject_line = "EnneadTab Auto Error Log"
                if is_silent:
                    subject_line += "(Silent)"
                try:
                    EMAIL.email_error(error_time + error, func.__name__,
USER.USER_NAME, subject_line=subject_line)
                except Exception as e:
                    print_note("Cannot send email: {}".format(e))

                if not is_silent:

                    error += "\n\n######If you have EnneadTab UI window open,
just close the original EnneadTab window. Do no more action, otherwise the
program might crash.##########\n#########Not sure what to do? Msg Sen Zhang, you
have dicovered a important bug and we need to fix it ASAP!!!!!########"
                    error_file =
FOLDER.get_EA_dump_folder_file("error_general_log.txt")
```

EXCEL.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""Utilities for working with Excel files.
Check formulas, read data, save data, etc."""

import os
import sys

try:
    # ■■■■■■■■■■utf8
    sys.setdefaultencoding("utf-8")
except:
    pass


import ENVIRONMENT
import EXE
import NOTIFICATION
import COLOR
import FOLDER
import UNIT_TEST

sys.path.append(ENVIRONMENT.DEPENDENCY_FOLDER)
import xlrd
import xlsxwriter


def letter_to_index(letter):
    """Get the index of a letter in the alphabet.
    A -> 0, B -> 1, C -> 2, etc.

    Args:
        letter (str): A single letter.

    Returns:
        int: The index of the letter in the alphabet.
    """
    try:
        return ord(letter.upper()) - ord("A")
    except TypeError:
        return None


def get_column_index(letter):
    """Get the index of an Excel column.

    Args:
        letter (str): The column letter.

    Returns:
```

EXE.py

```
"""Run apps from the EnneadTab app library."""
import shutil
```

```
import os
import time
import ENVIRONMENT
import FOLDER
import USER
import NOTIFICATION
import COPY

def try_open_app(exe_name, legacy_name = None, safe_open = False):
    """Attempt to open an exe file from the app libary.

    Args:
        exe_name (str): The name of the exe file to open.
        legacy_name (str): The name of the legacy exe file to open (optional).
        safe_open (bool): Whether to open the exe file using safe mode.

    Note:
        When using safe open, a temporary copy of the exe file will be created
in the dump folder.
        This is to address the issue that the exe file cannot be updated while
it is being used.
        The temporary copy will be purged after a certain period of time.
    """

    abs_name = exe_name.lower()
    if abs_name.endswith(".3dm") or abs_name.endswith(".xlsx") or
abs_name.endswith(".xls") or abs_name.endswith(".pdf") or
abs_name.endswith(".png") or abs_name.endswith(".jpg"):
        os.startfile(exe_name)
        return True



    exe_name = exe_name.replace(".exe", "")
    exe = ENVIRONMENT.EXE_PRODUCT_FOLDER + "\\{}.exe".format(exe_name)


    def get_ignore_age(file):
        if "OS_Installer" in file or "AutoStartup" in file:
            return 60*5
        return 60*60*24
    if safe_open:
        if not os.path.exists(exe):
            raise Exception("Only work for stanfle along exe, not for foldered
exe.[{}] not exist".format(exe))
        temp_exe_name = "_temp_exe_{}_{}.exe".format(exe_name, int(time.time()))
        temp_exe = FOLDER.DUMP_FOLDER + "\\" + temp_exe_name
        # print (temp_exe)
        COPY.copyfile(exe, temp_exe)
        os.startfile(temp_exe)
        for file in os.listdir(FOLDER.DUMP_FOLDER):
            if file.startswith("_temp_exe_"):
```

FOLDER.py

```
"""Utility functions for file and folder operations. Read and write to local and
shared dump folders. Format filenames within a folder."""

import time
import os

from ENVIRONMENT import DUMP_FOLDER, USER_DESKTOP_FOLDER, SHARED_DUMP_FOLDER
import COPY
def copy_file(original_path, new_path):
    """Copy file from original path to new path. If the new path does not exist,
```

```
it will be created.

    Args:
        original_path (str): The path of the original file.
        new_path (str): The path of the new file.
    """
    target_folder = os.path.dirname(new_path)
    if not os.path.exists(target_folder):
        os.mkdir(target_folder)
    COPY.copyfile(original_path, new_path)


def copy_file_to_folder(original_path, target_folder):
    """Copy a file to a specified folder. If the folder does not exist, it will
be created.

    Args:
        original_path (str): The path of the original file.
        target_folder (str): The path of the target folder.

    Returns:
        str: The new path of the copied file.
    """

    new_path = original_path.replace(os.path.dirname(original_path),
target_folder)
    try:
        COPY.copyfile(original_path, new_path)
    except Exception as e:
        print(e)

    return new_path


def secure_folder(folder):
    """Create a folder if it does not exist.

    Args:
        folder (str): The path of the folder to secure.

    Returns:
        str: The path of the folder.
```

GUI.py

```
"""Manipulate the GUI of apps by searching for image patterns."""

import EXE
import DATA_FILE

def simulate_click_on_image(image):
    """Add search json to find this image on screen and try to click on it."""

    with DATA_FILE.update_data("auto_click_data.sexyDuck") as data:
        if "ref_images" not in data:
            data["ref_images"] = []
        data["ref_images"].append(image)

    EXE.try_open_app("AutoClicker.exe")
```

HOLIDAY.py

```
# https://python-holidays.readthedocs.io/en/latest/index.html

"""Used to make customized greetings for all the employees in the office."""

import datetime


try:
    from pyrevit import script
except:
    pass

import os
import sys
import random


import FOLDER
import EXE
import SOUND
import ENVIRONMENT
import NOTIFICATION


def festival_greeting():
    greeting_chinese_new_year()
    greeting_mid_moon()
    greeting_xmas()
    greeting_pi()
    greeting_april_fools()
    greeting_may_force()


def is_valid_date(date_tuple_start, date_tuple_end):
    d0 = datetime.datetime(
        date_tuple_start[0], date_tuple_start[1], date_tuple_start[2]
    )
    today = datetime.datetime.now()
    d1 = datetime.datetime(date_tuple_end[0], date_tuple_end[1],
date_tuple_end[2])

    return d0 <= today <= d1


def greeting_april_fools():
    if not is_valid_date((2024, 3, 31), (2024, 4, 1)):
        return

    import JOKE

    for _ in range(random.randint(1, 5)):
```

IMAGE.py

```
"""Utilities for image retrieval and manipulation."""

import os
import random
import ENVIRONMENT

try:
```

```
    import System.Drawing as SD  # pyright: ignore
except Exception as e:
    pass


def get_image_path_by_name(file_name):
    """Get the full path for a specified image in the EnneadTab image library.

    Args:
        file_name (str): The name of the image file to retrieve, including
extension.

    Returns:
        str: The full path to the image file.
    """
    path = "{}\\{}".format(ENVIRONMENT.IMAGE_FOLDER, file_name)
    if os.path.exists(path):
        return path
    print("A ha! {} is not valid or accessibile. Better luck next
time.".format(path))


def get_one_image_path_by_prefix(prefix):
    """Will return a random image file from the EnneadTab image library that
starts with the specified prefix.

    Args:
        prefix (str): The prefix to search for in the image file names.

    Returns:
        str: The full path to the image file.
    """
    files = [
        os.path.join(ENVIRONMENT.IMAGE_FOLDER, f)
        for f in os.listdir(ENVIRONMENT.IMAGE_FOLDER)
        if f.startswith(prefix)
    ]
    file = random.choice(files)
    return file


def average_RGB(R, G, B):
    """Average the RGB values of a pixel to simplify it to greyscale.

    Args:
        R (int): Red. 0-255.
```

JOKE.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import os
import sys
import random
import io




import FOLDER
import NOTIFICATION
import SPEAK
import ENVIRONMENT
import USER
```

```
import EXE
import TIME
import SOUND
import EMOJI
import CONFIG
import DUCK
import DOCUMENTATION


TARGETS = ['fsun',
           'eshaw']


def is_hate_fun():
    return not CONFIG.get_setting("radio_bt_popup_full", False)

def get_all_jokes():
    with io.open(DOCUMENTATION.get_text_path_by_name('_dad_jokes.txt'), "r",
encoding = "utf8") as f:
        lines = f.readlines()
    return [x.replace("\n", "") for x  in lines if x != "\n"]

def get_all_loading_screen_message():
    with
open(DOCUMENTATION.get_text_path_by_name('_loading_screen_message.txt'), "r") as
f:
        lines = f.readlines()
    return [x.replace("\n", "") for x  in lines if x != "\n"]

def random_joke():

    lines = get_all_jokes()


    random.shuffle(lines)
    return lines[0].replace("\n", "")
```

KEYBOARD.py

```
pass
```

LEADER_BOARD.py

```
import DATA_FILE
import FOLDER
import USER

from CONFIG import GLOBAL_SETTING_FILE

PRICE = {
    "daily_reward": {
        "money_delta":99,
        "description":"xxx"
        },
    "sync_queue_cut": {
        "money_delta":-500,
        "description":"xxx"
        },
```

```
    "sync_queue_stay": {
        "money_delta":200,
        "description":"xxx"
        },
}


@FOLDER.backup_data(GLOBAL_SETTING_FILE , "setting")
def update_account(event_key):
    event_data = PRICE.get(event_key)
    if not event_data:
        raise "Cannot find event key {}".format(event_key)
    with DATA_FILE.update_data(GLOBAL_SETTING_FILE) as data:

        if "money" not in data.keys():
            data["money"] = 100
        data["money"] += event_data.get("money_delta")
        return data["money"]

def get_money():
    return update_account(0)






def print_leader_board():
    pass

def print_history():
    pass

def manual_transaction():
```

LOG.py

```
"""Logging functions for recording the usage of EnneadTab scripts."""

import time
from contextlib import contextmanager

import USER
import TIME
import FOLDER
import USER
import DATA_FILE
import ENVIRONMENT

LOG_FILE_NAME = "log_{}.sexyDuck".format(USER.USER_NAME)


@contextmanager
def log_usage(func, *args):
    """Context manager to log the usage of a function.

    Args:
        func (_type_): The function to log.
        *args (_type_): The arguments to pass to the function

    Yields:
        _type_: The result of the function.

    """
    t_start = time.time()
```

```
    res = func(*args)
    yield res
    t_end = time.time()
    duration = TIME.get_readable_time(t_end - t_start)
    with open(FOLDER.get_EA_dump_folder_file(LOG_FILE_NAME), "a") as f:
        f.writelines("\nRun at {}".format(TIME.get_formatted_time(t_start)))
        f.writelines("\nDuration: {}".format(duration))
        f.writelines("\nFunction name: {}".format(func.__name__))
        f.writelines("\nArguments: {}".format(args))
        f.writelines("\nResult: {}".format(res))


# with log_usage(LOG_FILE_NAME) as f:
#     f.writelines('\nYang is writing!')


"""log and log is break down becasue rhino need a wrapper to direct run script
directly
whereas revit need to look at local func run"""


@FOLDER.backup_data(LOG_FILE_NAME, "log")
def log(script_path, func_name_as_record):
```

MATH.py

```
pass
```

MODULE_HELPER.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""Utilities for running functions in other Python modules, Rhino, or Revit."""

import os
import imp
import sys
import FOLDER
import ENVIRONMENT
import ERROR_HANDLE
import NOTIFICATION


def run_func_in_module(module_path, func_name, *args):
    """Run a specified function in a specified python file.

    Args:
        module_path (str): Path to the python file.
        func_name (str): Name of function to run.
        *args: Positional arguments to pass to the function.
    """
    module_name = FOLDER.get_file_name_from_path(module_path).replace(".py", "")
    ref_module = imp.load_source(module_name, module_path)
    func = getattr(ref_module, func_name, None) or getattr(
        ref_module, module_name, None
    )
    if func is None:
```

```
        NOTIFICATION.messenger(
            main_text="Oooops, cannot find the the source code.\nSen Zhang is no
longer working for EnneadTab unluckly."
        )
    else:
        func(*args)


@ERROR_HANDLE.try_catch_error(is_silent=True)
def run_revit_script(script_subfolder_or_fullpath, func_name, *args, **kwargs):
    """Run a specified function in a specified file, for use with Revit buttons.

    Args:
        script_subfolder (str): such as
            "XX.tab\\YY.panel\\ZZ.pulldown" or
            "XX.tab\\YY.panel" or
            end with .py
        func_name (str): name of the func to run
    """

    folder_or_fullpath = "{}\\{}".format(
        ENVIRONMENT.REVIT_PRIMARY_EXTENSION, script_subfolder_or_fullpath
    )
```

NOTIFICATION.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""Notifications for the user, such as popups and sounds."""

import SOUND
import DATA_FILE
import EXE
import IMAGE
import CONFIG

def is_hate_messenger():
    """Check to see if the user has disabled the messenger.

    Returns:
        bool: True if the user has disabled the messenger.
    """
    return  CONFIG.get_setting("radio_bt_popup_minimal", False)

def is_hate_duck_pop():
    """Check to see if the user has disabled the duck pop.

    Returns:
        bool: True if the user has disabled the duck pop.
    """
    return not CONFIG.get_setting("toggle_bt_is_duck_allowed", False)

def messenger(main_text,
              width = 1200,
              height = None,
              image = None,
              animation_stay_duration = 5,
              x_offset = 0):
    """Pop a simple message to the user, which disappears after a few seconds.

       It can be used in place of the Windows notification, which is more
annoying and has a sound .

    Args:
```

```
        main_text (str): the message to show. Better within 2 return lines. If
too long, please use line return.
        width (int, optional): how width is the message max width. Defaults to
1200.
        height (int, optional): how tall is the message max height. Defaults to
150.
    """

    if is_hate_messenger():
        return

    if not isinstance(main_text, str):
        main_text = str(main_text)

    data = {}
```

OUTPUT.py

```
import os
import io

import webbrowser


import FOLDER
import ENVIRONMENT
import NOTIFICATION
import TIME
import IMAGE

FUNCS = """
<script>
function sample_func(btn) {
  alert(btn.innerText);
  prompt("Type anything:");
  confirm("Do you want to continue?");
}

function highlightSearch() {
  var input, filter, body, p, h1, h2, li, i, txtValue;
  input = document.getElementById('searchBox');
  filter = input.value.toLowerCase();
  body = document.getElementsByTagName('body')[0];

  // Highlight paragraphs
  p = body.getElementsByTagName('p');
  for (i = 0; i < p.length; i++) {
    txtValue = p[i].textContent || p[i].innerText;
    if (filter === "") {
      p[i].style.backgroundColor = '';
    } else if (txtValue.toLowerCase().indexOf(filter) > -1) {
      p[i].style.backgroundColor = 'lightgreen';
    } else {
      p[i].style.backgroundColor = '';
    }
  }

  // Highlight titles
  h1 = body.getElementsByTagName('h1');
  for (i = 0; i < h1.length; i++) {
    txtValue = h1[i].textContent || h1[i].innerText;
    if (filter === "") {
      h1[i].style.backgroundColor = '';
    } else if (txtValue.toLowerCase().indexOf(filter) > -1) {
      h1[i].style.backgroundColor = 'lightgreen';
```

```
    } else {
      h1[i].style.backgroundColor = '';
    }
```

OVERLOAD.py

```
"""example on overload handling

import System.Collections.Generic.IEnumerable as IEnumerable


for srf in srfs:
    splitBrep = srf.Split.Overloads[IEnumerable[Rhino.Geometry.Curve],
System.Double](cutters, tol)



wrong example.....
if you have a out parameter, iron python will return as tuple instead using it
in args
success, family_ref = project_doc.LoadFamily.Overloads[str,
DB.IFamilyLoadOptions](temp_path, loading_opt, family_ref)
"""

import os
```

PDF.py

```
import os

def pdf2img(pdf_path, output_path = None):
    pass


def img2pdf(image_path, output_path = None):
    """convert image to pdf.

    Args:
        image_path (str): path for the input image
        pdf_path (str): path for the output pdf
    """
    from PIL import Image


    if pdf_path is None:
        # replace any image extension with pdf extension
        extension = os.path.splitext(image_path)[1]
        pdf_path = image_path.replace(extension, ".pdf")

    # Open the image
    with Image.open(image_path) as img:
        # Converting the image to RGB, to ensure compatibility
        if img.mode != 'RGB':
            img = img.convert('RGB')

        # Save as PDF
        img.save(pdf_path, "PDF", resolution=100.0)
    return pdf_path
```

```
def pdfs2pdf(combined_pdf_file_path, list_of_filepaths, reorder = False):
    """merge multiple pdfs to single pdf.

    Args:
        combined_pdf_file_path (str): path for final product
        list_of_filepaths (list): list of l=path for the input pdfs
        reorder (bool, optional): reorder the pdf alphabetically. Defaults to
False.
    """
    from PyPDF2 import PdfFileMerger

    merger = PdfFileMerger()

    if reorder:
        list_of_filepaths.sort()

    for filepath in list_of_filepaths:
        merger.append(filepath)
```

SAMPLE_FILE.py

```
"""used to retrive sample file in rhino, reivt and excel.
Good the sharing template"""

import os
import ENVIRONMENT
import NOTIFICATION

def get_file(file_name):

    path = "{}\\{}\\{}".format(ENVIRONMENT.DOCUMENT_FOLDER,
ENVIRONMENT.get_app_name(), file_name)
    if os.path.exists(path):
        return path
    NOTIFICATION.messenger("Cannot find [{}]".format(file_name))
```

SECRET.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""Utilities for referencing secrets, such as API keys and developer identity
information."""

import os
import DATA_FILE
import ENVIRONMENT


def get_api_key(app_name):
    """Returns the API key for the specified app.
    Accepted keys:
    "chatgpt_api_key"
    "translator_api_key"
    "reporter_api_key"
    "clone_helper"
    "miro_oauth"

    Args:
```

```
        app_name (string): The name of the app to get the API key for.

    Returns:
        string: The API key for the specified app.
    """
    api_key_file = "EA_API_KEY.secret"
    L_drive_file_path = os.path.join(ENVIRONMENT.DB_FOLDER, api_key_file)
    if ENVIRONMENT.IS_OFFLINE_MODE:
        return DATA_FILE.get_data(api_key_file)
    data = DATA_FILE.get_data(L_drive_file_path)
    return data.get(app_name)


def get_dev_info(developer_name, key):
    """Get developer information from the secret file.

    Args:
        developer_name (string): The name of the developer.
        key (string): The key to get the value for.

    Returns:
        string: The value of the key for the developer.
    """
    data = get_dev_dict()
    developer_data = data.get(developer_name)
    if not developer_data:
        return
    return developer_data.get(key)


def get_dev_dict():
```

SHORT_CUT.py

```
    pass
```

SOUND.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import os
import threading
import time
import random
import ENVIRONMENT
import TEXT

def get_audio_path_by_name(file_name):
    if not file_name.endswith(".wav"):
        file_name = file_name + ".wav"
    if os.path.exists(file_name):
        return file_name
    path = "{}\\{}".format(ENVIRONMENT.AUDIO_FOLDER, file_name)
    if os.path.exists(path):
        return path
    print ("A ha! {} is not valid or accessibile. Better luck next
time.".format(path))
    return False
```

```
def get_one_audio_path_by_prefix(prefix):
    files = [os.path.join(ENVIRONMENT.AUDIO_FOLDER, f) for f in
os.listdir(ENVIRONMENT.AUDIO_FOLDER) if f.startswith(prefix)]
    file = random.choice(files)
    return file


def play_sound(file = "sound_effect_popup_msg3"):
    file = get_audio_path_by_name(file)
    if not file:
        return

    try:
        from System.Media import SoundPlayer # pyright: ignore
        sp = SoundPlayer()
        sp.SoundLocation = file
        sp.Play()
        return True
    except Exception as e:
        # print ("Cannot use system media becasue: " + str(e))
        pass

    try:
        import playsound # pyright : ignore
        playsound.playsound(file)
        return True
    except Exception as e:
        # print ("Cannot use native playsound  becasue: " + str(e))
        pass
    try:
```

SPEAK.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import random
import DATA_FILE
import EXE
import CONFIG

def random_speak(lines, chance=1.0):
    if random.random() <= chance:
        random.shuffle(lines)
        speak(lines[0])


def is_hate_talkie():
    return not CONFIG.get_setting("toggle_bt_is_talkie", False)


def speak(text, language='en', accent='com'):
    """
    #language = 'zh-CN'
    #language = 'zh-TW'
    #language = 'en'

    #accent = 'co.uk'
    #accent = 'co.in'
    #accent = 'com'
    """
    if is_hate_talkie():
        return

    if not text:
        return
```

```
    data = {}
    data["text"] = text
    data["language"] = language
    data["accent"] = accent
    DATA_FILE.set_data(data, "text2speech.sexyDuck")



    EXE.try_open_app("Speaker")


def unit_test():
    speak("I like to move it move it!")


if __name__ == "__main__":
    speak("This is a test?")
```

TASK.py

```
import subprocess

class TaskScheduler:


    def add_scheduled_task(self, task_name, exe_path):
        # Format the command to add to the Task Scheduler
        command = 'schtasks /create /tn "{}" /tr "{}" /sc daily /st
00:00'.format(task_name, exe_path)

        try:
            # Run the command
            subprocess.check_call(command, shell=True)
            print ("Task scheduled successfully: {}".format(task_name))
        except subprocess.CalledProcessError as e:
            print ("Failed to schedule task:", e)

    def remove_scheduled_task(self, task_name):
        # Format the command to delete the task from the Task Scheduler
        command = 'schtasks /delete /tn "{}" /f'.format(task_name)

        try:
            # Run the command
            subprocess.check_call(command, shell=True)
            print ("Task '{}' removed successfully.".format(task_name))
        except subprocess.CalledProcessError as e:
            print ("Failed to remove scheduled task:", e)
```

TEXT.py

```
import ENVIRONMENT

import sys
sys.path.append(ENVIRONMENT.DEPENDENCY_FOLDER)

from termcolor import colored # pyright: ignore
from COLOR import TextColorEnum
def colored_text(text, color = TextColorEnum.Cyan, on_color=None, attrs=None):
```

```
    """Colorize text.

    Available text colors:
        red, green, yellow, blue, magenta, cyan, white.

    Available text highlights:
        on_red, on_green, on_yellow, on_blue, on_magenta, on_cyan, on_white.

    Available attributes:
        bold, dark, underline, blink, reverse, concealed.

    Example:
        colored('Hello, World!', 'red', 'on_grey', ['blue', 'blink'])
        colored('Hello, World!', 'green')
    """


    if "colored" not in globals():
        # in some terminal run, it cannot read the dependecy folder so cannot
load the colored moudle
        return text
    return colored(text, color, on_color, attrs)



def unit_test():
    print (colored_text("Test dfault color text"))
    print (colored_text("test green", TextColorEnum.Green))#,
attrs=[TextColorEnum.Blue, 'blink']))




if __name__ == "__main__":
    unit_test()
```

TIME.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import datetime
import time
import threading

"""
'{:02d}' means minimal 2 digit long, use 0 at left padding.
For 4 digit number like years, this has no effect
"""

def get_YYYYMMDD():
    """2023-02-17 output as 230217"""
    now = datetime.datetime.now()
    year, month, day = '{:02d}'.format(now.year), '{:02d}'.format(now.month),
'{:02d}'.format(now.day)

    return "{}{}{}".format(year, month, day)

def get_YYYY_MM_DD():
    """2023-02-17 output as 2023-02-17"""
    now = datetime.datetime.now()
    year, month, day = '{:02d}'.format(now.year), '{:02d}'.format(now.month),
'{:02d}'.format(now.day)

    return "{}-{}-{}".format(year, month, day)
def get_date_as_tuple(return_string = True):
```

```
    """2023-02-17 output as (2023,02,17)"""
    now = datetime.datetime.now()
    year, month, day = '{:02d}'.format(now.year), '{:02d}'.format(now.month),
'{:02d}'.format(now.day)
    if return_string:return year, month, day
    return int(year), int(month), int(day)

def timer(func):
    def wrapper(*args, **kwargs):
        time_start = time.time()
        out = func(*args, **kwargs)
        used_time = time.time() - time_start
        try:
            print ("Function: {} use {}".format(func.__name__,
get_readable_time(used_time)))
        except:
            print (used_time)
        return out
    return wrapper


def get_formatted_current_time():
    """-->2023-05-16_11-33-55"""
    now = datetime.datetime.now()
    return get_formatted_time(now)
```

TIMESHEET.py

```
import time


import DATA_FILE
import ENVIRONMENT
import TIME
import OUTPUT
import FOLDER
import USER

TIMESHEET_DATA_FILE = "timesheet_{}.sexyDuck".format(USER.USER_NAME)


@FOLDER.backup_data(TIMESHEET_DATA_FILE , "timesheet")
def update_timesheet(doc_name):
    app_name = ENVIRONMENT.get_app_name()
    _update_time_sheet_by_software(doc_name, app_name)

def print_timesheet_detail():
    def print_in_style(text):
        if ENVIRONMENT.IS_REVIT_ENVIRONMENT:
            from pyrevit import script
            output = script.get_output()
            lines = text.split("\n")
            for line in lines:
                output.print_md(line)
            return
        print(text)

    output = ""
    data = DATA_FILE.get_data(TIMESHEET_DATA_FILE, is_local=False)
    if not data:
        data = DATA_FILE.get_data(TIMESHEET_DATA_FILE, is_local=True)

    for software in ["revit", "rhino", "terminal"]:
        output += "\n\n"
        output += "\n# Printing timesheet for {}".format(software.capitalize())
```

```
        log_data = data.get(software, {})
        for date, doc_data in sorted(log_data.items()):
            output += "\n## Date: {}".format(date)
            for doc_name, doc_info in doc_data.items():
                output += "\n### Doc Name: {}".format(doc_name)
                starting_time = doc_info.get("starting_time", None)
                end_time = doc_info.get("end_time", None)
                duration = end_time - starting_time if starting_time and
end_time else 0
                if duration < 2:
                    output += "\n    - Open Time:
{}".format(TIME.get_formatted_time(starting_time))
                else:
                    if starting_time and end_time:
```

UI.py

```
pass
```

UNIT_TEST.py

```
try:
    import imp
except:
    pass
import os
import traceback

import ENVIRONMENT
import NOTIFICATION
import OUTPUT
import TEXT


def print_boolean_in_color(bool):
    if not ENVIRONMENT.is_terminal_environment():
        return bool

    import TEXT

    if bool:
        return TEXT.colored_text("True", TEXT.TextColorEnum.Green)
    else:
        return TEXT.colored_text("False", TEXT.TextColorEnum.Red)


def print_text_in_highlight_color(text, ok=True):
    if not ENVIRONMENT.is_terminal_environment():
        return text

    return TEXT.colored_text(
        text, TEXT.TextColorEnum.Blue if ok else TEXT.TextColorEnum.Red
    )


IGNORE_LIST = ["__pycache__", "RHINO"]
def module_call_test(module_call):
    module = module_call.split(".")[0]
```

```
    eval("import {}".format(module))
    results = eval(repr(module_call))
    return results


def pretty_test(test_dict, filename):
    """Test function for a module with a dictionary of test cases.
    Only intended to run in terminal, using Python 3.
    Rhino and Revit in-envionment testing will be supported
    with future updates.
```

USER.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""Utilities for getting user information and permissions"""

import os
import traceback
import ENVIRONMENT
import SECRET
import UNIT_TEST


USER_NAME = os.environ["USERPROFILE"].split("\\")[-1]


# note: why has seperate system key and autodesk keys? becasue some
# developer might only be handling one software, not both.
EnneadTab_DEVELOPERS = SECRET.get_dev_dict()


def get_EA_email_address(user_name = USER_NAME):
    return "{}@ennead.com".format(user_name.replace(".EA",""))



def get_usernames_from_developers():
    """Get all usernames from a dictionary of developers.

    Args:
        username (str): The username or Autodesk ID to check.

    Returns:
        tuple: list: all system usernames, list: all Autodesk usernames
    """

    system_usernames = []
    autodesk_usernames = []
    for key in EnneadTab_DEVELOPERS:
        system_usernames += EnneadTab_DEVELOPERS[key]["system_id"]
        autodesk_usernames += EnneadTab_DEVELOPERS[key]["autodesk_id"]
    return system_usernames, autodesk_usernames


def is_EnneadTab_developer():
    """Checks if the current user is a developer of EnneadTab.

    Args:


    Returns:
```

VERSION_CONTROL.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-


import os

import EXE
import ENVIRONMENT
import NOTIFICATION




def update_EA_dist():
    EXE.try_open_app("EnneadTab_OS_Installer", safe_open=True)
    EXE.try_open_app("RegisterAutoStartup", safe_open=True)




def show_last_success_update_time():
    records = [file for file in os.listdir(ENVIRONMENT.ECO_SYS_FOLDER) if
file.endswith(".duck") and not "_ERROR" in file]
    if len(records) == 0:
        NOTIFICATION.messenger("Not successful update recently.\nYour life
sucks.")
        return
    records.sort()
    record = records[-1]
    with open(os.path.join(ENVIRONMENT.ECO_SYS_FOLDER, record)) as f:
        commit_line = f.readlines()[-1].replace("\n","")
    NOTIFICATION.messenger("Most recent update
at:{}\n{}".format(record.replace(".duck", ""),
                                                                    commit_line))

    pass

def unit_test():
    update_EA_dist()



if __name__ == "__main__":
    unit_test()
    show_last_success_update_time()
```

WEB.py

```
pass
```