

```
import camelot
import pandas as pd

from extractors.direct_downloads import extract_with_request
from models.registry import Registry
from utils.helpers import clean_str


class BLZ(Registry):
    def __init__(self, **kwargs):
        """
        Child container for extracting, transforming, and loading registry data for Belize

        :param kwargs: see models.registry for details
        """

        # assign constant values for final registry
        self.country_code = "Belize"
        self.country_name = "Belize"
        self.country_flag = "https://www.belize.gov.bz/images/belize-flag.png"
        self.airframe_model = "Boeing 737-800"
        self.airframe_manufacturer = "Boeing"

        # assign registry input data source and filename
        self.registry_input_dir = "https://www.belize.gov.bz/images/belize-airframe-model.png"
        self.registry_filename = "Belize-Airframe-Model.pdf"

        # assign registry input data extraction method
        self.extract_registry = extract_with_request

        # assign registry input data reading method
        self.read_registry = pd.read_csv

        # assign registry data wrangling method
        self.wrangle_registry = wrangle_registry

        # extract, transform, and load registry data
        self.extract_registry()

    def extract_registry(self):
        """
        Extracts registry input data from remote source

        :return: effect - creates [DATA_DIR|TEST_DIR]/[REGISTRY_INPUT_DIR]/[self.country_code]/[self.registry_filename] file
        """

        # extract registry input data
        url = "https://www.belize.gov.bz/images/belize-airframe-model.png"
        self.registry_filename = "Belize-Airframe-Model.pdf"

    def read_registry(self):
        """
        Reads registry input data

        :return: effect - modifies self.registry
        """

        # read registry input data
        self.registry = pd.read_csv(url)

        # prep registry input data
        self.registry.columns = ["id", "name", "manufacturer", "model"]

        # assign registry input data
        self.registry.columns = ["id", "name", "manufacturer", "model"]

    def wrangle_registry(self):
        """
        Transforms registry data

        :return: effect - modifies self.wrapped_records
        """

        # loop through raw registrations
        for i in range(len(self.registry)):
            # initialize storage for wrapped representation of registration in focus
            record = {}

            # assign airframe-related values
            record["id"] = self.registry.iloc[i]["id"]
            record["name"] = self.registry.iloc[i]["name"]
            record["manufacturer"] = self.registry.iloc[i]["manufacturer"]
            record["model"] = self.registry.iloc[i]["model"]

            # update list of transformed values to populate final registry
            self.wrapped_records.append(record)

    @staticmethod
    def __format_registry(self, registry):
        """
        Cleans and merges tables across multiple pdf pages

        :param (camelot.core.TableList) registry: tables of registrations from registry pdf to process
        :return: cleaned and merged transformation of 'registry' argument
        """

        # initialize storage for tables on multiple pages
        tables = []

        # initialize storage for column names
        headers = []

        # loop through pdf pages
        for i in range(len(registry)):
            # assign table on page in focus
            table = registry[i]

            # assign column names once using the table on the first page
            headers = self.__extract_colnames(table.headers)

            # drop header row from each table
            table.drop(table.headers, inplace=True)

        # merge tables across multiple pages together
        registry = pd.concat(tables)

        return registry

    @staticmethod
    def __extract_colnames(header):
        """
        Parses and validates table column names that are incorrectly captured by the camelot extraction process

        :param (list) header: table column names from camelot reading method to extract values from
        :return: formatted copy of 'header' argument to label table columns
        """

        # remove invalid column names
        header = [x for x in header if x != ""]

        # initialize storage for output
        colnames = []

        # parse column names erroneously concatenated to others
        for i in range(len(header)):
            colnames.append(header[i].split(" ")[0])

        return colnames

    @staticmethod
    def __format_colname(colname):
        """
        Formats column name for namedtuple compatibility

        :param (str) colname: column name from registry input data to format
        :return: formatted copy of 'colname' argument suitable for namedtuple indexing
        """

        colname = colname.replace(" ", "_").replace("-", "_")

        return colname

    @staticmethod
    def __get_manufacturer_and_model(man_mod):
        """
        Parses and formats airframe manufacturer and model

        :param (str) man_mod: registration airframe to extract values from
        :return: formatted airframe manufacturer;
                formatted airframe model
        """

        # isolate airframe components
        man, mod = man_mod.split("/")

        # format airframe manufacturer
        manufacturer = man.replace(" ", "_").replace("-", "_")

        # format airframe model
        model = mod.replace(" ", "_").replace("-", "_")

        # handle when no airframe model information exists
        if mod == "":
            model = None

        return manufacturer, model
```