

```
from datetime import datetime
import os
import pandas as pd

from extractors.direct_downloads import extract_with_request
from models.registry import Registry
from utils.helpers import clean_str, binary to decimal, date to iso, build address
```

```

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

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

        # assign constant values for final registry
        self.country_code = "CA"
        self.registry_filename = "Canada_registry.csv"
        self.registry_dir = "data/registries"
        self.registry_file = "Canada_registry.csv"

        # assign registry input data source and filename
        self.registry_dir = "data/registries"
        self.registry_file = "Canada_registry.csv"

        # initialize storage for registrant reference data
        self.registrant_refs = {}

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

        # assign registry input data reading method
        self.read_registry = self._read_registry

        # assign registry data wrangling method
        self.wrangle_registry = self._wrap_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
        self._extract_registry()

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

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

        # read data layout and definitions reference
        self._read_data_layout_and_definitions_reference()

        # read registry input data
        self._read_registry_input_data()

        # prep registry input data
        self._prep_registry_input_data()

        # assign registry input data
        self._assign_registry_input_data()

        # read registrant reference data
        self._read_registrant_reference_data()

        # prep registrant reference data
        self._prep_registrant_reference_data()

        # assign registrant reference data
        self._assign_registrant_reference_data()

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

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

        # loop through raw registrations
        for registration in self.registry:
            # initialize storage for wrangled representation of registration in focus
            wrangled_registration = {}

            # subset corresponding registrant references for registration in focus
            registrant_refs = self.registrant_refs[registration['registrant_id']]

            # update list of transformed values to populate final registry
            self._update_wrangled_registration(wrangled_registration, registration)

            # update list of transformed values to record registrant and accommodate multiple registrants
            self._update_registrant_references(wrangled_registration, registration)

    def _add_registrant_copies(self, owners, record):
        """
        Adds copies of a wrangled registration with updated registrant information to reflect all registrants

        :param (pandas.DataFrame) owners: registrant reference data corresponding to a particular registration
        :param (dict) record: wrangled representation of a registration indicated by 'owners' argument to duplicate
        :return: effect - modifies self.wrangled_records
        """

        # loop through registrants
        for owner in owners:
            # avoid modifying original wrangled representation of registration
            wrangled_registration = record.copy()

            # update registrant-related values
            wrangled_registration['registrant_id'] = owner['registrant_id']
            wrangled_registration['registrant_name'] = owner['registrant_name']
            wrangled_registration['registrant_address'] = owner['registrant_address']

            # update list of transformed values to populate final registry
            self._update_wrangled_registration(wrangled_registration, record)

    @staticmethod
    def _format_registry(registry, layout_refs):
        """
        Cleans registry input data and adds column names parsed from data layout and definitions reference

        :param (pandas.DataFrame) registry: registry input data to clean and add column names to
        :param (list) layout_refs: contents of data layout and definitions reference to extract column names from
        :return: cleaned copy of 'registry' argument with column names
        """

        # assign indices for column name extraction
        layout_refs = layout_refs[layout_refs['column_name'] != '']

        # extract and assign appropriate column names from data layout and definitions reference
        registry.columns = layout_refs['column_name']

        # remove invalid rows
        registry = registry[registry['column_name'] != '']

    @staticmethod
    def _format_registrant_refs(registrant_refs, registry, layout_refs):
        """
        Adds column names parsed from data layout and definitions reference to registrant reference input data

        :param (pandas.DataFrame) registrant_refs: aircraft registration registrant input data to add column names to
        :param (list) layout_refs: contents of data layout and definitions reference to extract column names from
        :return: copy of 'registrant_refs' argument with column names
        """

        # assign indices for column name extraction
        layout_refs = layout_refs[layout_refs['column_name'] != '']

        # extract and assign appropriate column names from data layout and definitions reference
        registrant_refs.columns = layout_refs['column_name']

    @staticmethod
    def _get_is_active(status):
        """
        Determines whether a registration is currently active

        :param (str) status: registration status to check
        :return: 'True' if 'status' argument represents an active registration
        """

        status = status.lower()

        if status in ['active']:
            return True
        elif status in ['inactive']:
            return False
        else:
            return False

    @staticmethod
    def _get_year(date):
        """
        Extracts the year from a date

        :param (str) date: registration date to get year from
        :return: year value from 'date' argument
        """

        date = date.split('-')[0]

        if date in ['']:
            return False
        else:
            return date

    @staticmethod
    def _get_is_company(category):
        """
        Determines whether a registrant represents a company based on data documentation

        :param (str) category: category label representing type of registrant
        :return: 'True' if 'category' argument represents a company
        """

        category = category.lower()

        if category in ['company']:
            return True
        elif category in ['individual']:
            return False
        elif category in ['other']:
            return False
        else:
            return False

        # handle when category has value "N/F"
        return False

    @staticmethod
    def _get_address(owner):
        """
        Builds human-readable address in common format

        :param (namedtuple) owner: raw observation of a registrant corresponding to a particular registration
        :return: cleaned and formatted address of registrant
        """

        # assign country-specific common address format
        if owner.country_code == "CA":
            address = owner.address_line_1 + ", " + owner.address_line_2 + ", " + owner.city + ", " + owner.province + ", " + owner.country_code
        else:
            address = owner.address_line_1 + ", " + owner.address_line_2 + ", " + owner.city + ", " + owner.country_code

        return address

```