# classes\interface_classes.py

```
"""
Classes that each exchange interface should inherit from, to ensure consistency

API classes:
    Interface: Base class for all exchange interfaces

Other general classes:
    DataStore: Stores data from API
    SymbolsManager: Provides utility functions for symbols
"""
# ================ Imports ================
import csv
import pandas as pd
import os

# ================ Classes ================
class Interface():
    """
    Defines public functions to interact with exchange API
    """
    def __init__(self, access_key, secret_key, host):
        pass

    def get_symbols(self):
        pass

    def get_kline_history(self, symbol, interval, limit):
        pass

    def subscribe_to_candlestick(self, symbol, interval, callback_func):
        pass

    def request_trades(self, symbol, callback_func):
        pass


class DataStore:
    """
    Stores data from API
    """
    def __init__(self, exchange: str, symbol: str, metric: str, csv_name: str = None,
id_buffer_size: int = 1000):
        self.exchange = exchange
        self.data = []
        self.symbol = symbol
        self.metric = metric
        self.csv_name = self._set_csv_name(csv_name)
        self._create_csv()
        #self._empty_csv()
        self.id_buffer_size = id_buffer_size
        self.timestamps = {}

    def _set_csv_name(self, csv_name: str):
        name = csv_name
        if csv_name is None:
```

```
            name = f"data/{self.exchange}/{self.metric}/{self.symbol}.csv"
        return name

    def _empty_csv(self):
        """Remove current contents of csv file"""
        with open(self.csv_name, 'w') as f:
            pass

    def _create_csv(self):
        """Create csv file if it doesn't exist.
        If this fails, it's likely a permissions error. Try creating the data folders
manually.
        """
        f = open(self.csv_name, 'w')
        f.close()

    def store_data(self, data):
        self.data.append(data)

    def get_data(self):
        return self.data

    def write_to_csv_string(self, data):
        data = self._clean_data(data)
        with open(self.csv_name, 'a') as f:
            writer = csv.writer(f)
            writer.writerow(data)

    def write_data_to_csv(self, data: list, id_index: int = 0, error_msg: str = "Duplicate
ID"):
        data = self._clean_data(data)
        # Check timestamp is unique
        if data[id_index] in self.timestamps:
            #raise Exception(error_msg)
            return # Return instead of raising exception to reduce log spam - can be
uncommented for debugging
        else:
            self.timestamps[data[0]] = True

        with open(self.csv_name, 'a') as f:
            writer = csv.writer(f)
            writer.writerow(data)

    def _clean_data(self, data: list):
        # Remove newlines, spaces, and commas from strings
        for i in range(len(data)):
            if isinstance(data[i], str):
                data[i] = data[i].replace("\n", "")
                data[i] = data[i].replace(" ", "")
                data[i] = data[i].replace(",", "")
        return data

    def _check_unique_id(self, id: str):
        if id in self.timestamps:
            return False
        else:
            self.timestamps[id] = True
            # Remove oldest id if buffer is full
            if len(self.timestamps) > self.id_buffer_size:
```

```
                self.timestamps.popitem(last=False)
            return True


class SymbolsManagerBase:
    """
    Defines functions to be implemented by exchange-specific SymbolsManager classes
    """
    def __init__(self, exchange: str):
        self.exchange = exchange

    def convert_to_dataframe(self, symbols: list):
        pass

    def filter_excluded(self, symbols: pd.DataFrame, excluded_coins: list = []):
        filtered_symbols = []
        for index, row in symbols.iterrows():
            if row["data"]["symbol"] not in excluded_coins:
                filtered_symbols.append(row["data"]["symbol"])
        return filtered_symbols

    def filter_offline(self, symbols: pd.DataFrame, key: str = "state", value: str =
"online"):
        filtered_symbols = []
        for index, row in symbols.iterrows():
            if row["data"][key] == value:
                filtered_symbols.append(row["data"]["symbol"])
        return filtered_symbols
```