pyUSPTO

Release 0.3.2.dev17+g089b26f0a

Andrew Piechocki

Dec 31, 2025

CONTENTS:

1 Installation **1**

2 Quick Start **3**
- 2.1 Configuration . . . 3
- 2.2 Examples . . . 3

3 API Reference **5**
- 3.1 Clients . . . 5
- 3.2 Models . . . 34
- 3.3 Configuration . . . 99
- 3.4 Exceptions . . . 101
- 3.5 Warnings . . . 105

4 Examples **107**
- 4.1 Bulk Data Examples . . . 107
- 4.2 Patent Data Examples . . . 109
- 4.3 Image File Wrapper Example . . . 116
- 4.4 Petition Decisions Example . . . 117
- 4.5 PTAB Appeals Example . . . 123
- 4.6 PTAB Interferences Example . . . 129
- 4.7 PTAB Trials Example . . . 134

5 Development **139**
- 5.1 Contributing . . . 139
- 5.2 Testing . . . 139

6 Indices and tables **141**

Python Module Index **143**

Index **145**

CHAPTER

ONE

INSTALLATION

```
pip install pyUSPTO
```

For development installation:

```
git clone https://github.com/DunlapCoddingPC/pyUSPTO.git
cd pyUSPTO
pip install -e .
```

CHAPTER

TWO

QUICK START

This guide will help you get started with the pyUSPTO library.

2.1 Configuration

```python
from pyUSPTO import BulkDataClient, PatentDataClient
from pyUSPTO.config import USPTOConfig
import os

# Method 1: Direct API key initialization
client1 = BulkDataClient(api_key="your_api_key_here")

# Method 2: Using USPTOConfig
config = USPTOConfig(api_key="your_api_key_here")
client2 = BulkDataClient(config=config)

# Method 3: Using environment variables
os.environ["USPTO_API_KEY"] = "your_api_key_here"
config_from_env = USPTOConfig.from_env()
client3 = BulkDataClient(config=config_from_env)
```

2.2 Examples

Searching for patents:

```python
from pyUSPTO import PatentDataClient

client = PatentDataClient(api_key="your_api_key_here")

# Search for patents by inventor name
inventor_search = client.search_patents(inventor_name="Smith")
print(f"Found {inventor_search.count} patents with 'Smith' as inventor")
```

CHAPTER

THREE

API REFERENCE

3.1 Clients

clients.bulk_data - Client for USPTO bulk data API.

This module provides a client for interacting with the USPTO Open Data Portal (ODP) Bulk Data API. It allows you to search for and download bulk data products.

class pyUSPTO.clients.bulk_data.**BulkDataClient**(*api_key=None*, *base_url=None*, *config=None*)

Bases: BaseUSPTOClient[*BulkDataResponse*]

Client for interacting with the USPTO bulk data API.

ENDPOINTS = {'download_file': 'api/v1/datasets/products/files/{file_download_uri}', 'product_by_id': 'api/v1/datasets/products/{product_id}', 'products_search': 'api/v1/datasets/products/search'}

__init__(*api_key=None*, *base_url=None*, *config=None*)

Initialize the BulkDataClient.

Parameters

- **api_key** (Optional[str]) – Optional API key for authentication
- **base_url** (Optional[str]) – The base URL of the API, defaults to config.bulk_data_base_url or "https://api.uspto.gov/api/v1/datasets"
- **config** (Optional[*USPTOConfig*]) – Optional USPTOConfig instance

download_file(*file_data*, *destination*)

Download a file from the API.

Parameters

- **file_data** (*FileData*) – FileData object containing file information
- **destination** (str) – Directory where the file should be saved

Return type
str

Returns
Path to the downloaded file

get_product_by_id(*product_id*, *file_data_from_date=None*, *file_data_to_date=None*, *offset=None*, *limit=None*, *include_files=None*, *latest=None*)

Get a specific bulk data product by ID.

Parameters

- **product_id** (str) – The product identifier
- **file_data_from_date** (Optional[str]) – Filter files by data from date (YYYY-MM-DD)
- **file_data_to_date** (Optional[str]) – Filter files by data to date (YYYY-MM-DD)
- **offset** (Optional[int]) – Number of product file records to skip
- **limit** (Optional[int]) – Number of product file records to collect
- **include_files** (Optional[bool]) – Whether to include product files in the response
- **latest** (Optional[bool]) – Whether to return only the latest product file

Return type
BulkDataProduct

Returns
BulkDataProduct object containing the product data

get_products(*params=None*)

Get a list of bulk data products.

This method is deprecated. Use search_products instead.

Parameters
params (Optional[dict[str, Any]]) – Optional query parameters

Return type
BulkDataResponse

Returns
BulkDataResponse object containing the API response

paginate_products(*post_body=None*, ***kwargs*)

Paginate through all products matching the search criteria.

Supports both GET and POST requests.

Parameters

- **post_body** (Optional[dict[str, Any]]) – Optional POST body for complex search queries
- ****kwargs** (Any) – Keyword arguments for GET-based pagination

Yields
BulkDataProduct objects

Return type
Iterator[*BulkDataProduct*]

search_products(*query=None*, *product_title=None*, *product_description=None*, *product_short_name=None*, *from_date=None*, *to_date=None*, *categories=None*, *labels=None*, *datasets=None*, *file_types=None*, *offset=None*, *limit=None*, *include_files=None*, *latest=None*, *facets=None*)

Search for products with various filters.

Parameters

- **query** (Optional[str]) – Search text
- **product_title** (Optional[str]) – Filter by product title
- **product_description** (Optional[str]) – Filter by product description

- **product_short_name** (Optional[str]) – Filter by product identifier (short name)
- **from_date** (Optional[str]) – Filter products with data from this date (YYYY-MM-DD)
- **to_date** (Optional[str]) – Filter products with data until this date (YYYY-MM-DD)
- **categories** (Optional[list[str]]) – Filter by dataset categories
- **labels** (Optional[list[str]]) – Filter by product labels
- **datasets** (Optional[list[str]]) – Filter by datasets
- **file_types** (Optional[list[str]]) – Filter by file types
- **offset** (Optional[int]) – Number of product records to skip
- **limit** (Optional[int]) – Number of product records to collect
- **include_files** (Optional[bool]) – Whether to include product files in the response
- **latest** (Optional[bool]) – Whether to return only the latest product file for each product
- **facets** (Optional[bool]) – Whether to enable facets in the response

Return type
BulkDataResponse

Returns
BulkDataResponse object containing matching products

clients.patent_data - Client for USPTO patent data API.

This module provides a client for interacting with the USPTO Patent Data API. It allows you to search for and retrieve patent application data.

class pyUSPTO.clients.patent_data.**PatentDataClient**(*api_key=None*, *base_url=None*, *config=None*)

Bases: BaseUSPTOClient[*PatentDataResponse*]

Client for interacting with the USPTO Patent Data API.

```
ENDPOINTS = {'download_application_document':
'api/v1/download/applications/{application_number}/{document_id}',
'get_application_adjustment':
'api/v1/patent/applications/{application_number}/adjustment',
'get_application_assignment':
'api/v1/patent/applications/{application_number}/assignment',
'get_application_associated_documents':
'api/v1/patent/applications/{application_number}/associated-documents',
'get_application_attorney':
'api/v1/patent/applications/{application_number}/attorney',
'get_application_by_number':  'api/v1/patent/applications/{application_number}',
'get_application_continuity':
'api/v1/patent/applications/{application_number}/continuity',
'get_application_documents':
'api/v1/patent/applications/{application_number}/documents',
'get_application_foreign_priority':
'api/v1/patent/applications/{application_number}/foreign-priority',
'get_application_metadata':
'api/v1/patent/applications/{application_number}/meta-data',
'get_application_transactions':
'api/v1/patent/applications/{application_number}/transactions',
'get_search_results':  'api/v1/patent/applications/search/download',
'search_applications':  'api/v1/patent/applications/search', 'status_codes':
'api/v1/patent/status-codes'}
```

__init__(*api_key=None*, *base_url=None*, *config=None*)

Initialize the PatentDataClient.

Parameters

- **api_key** (Optional[str]) – USPTO API key. If not provided, uses key from config or environment.
- **base_url** (Optional[str]) – Base URL for the USPTO Patent Data API. Defaults to https://api.uspto.gov.
- **config** (Optional[*USPTOConfig*]) – USPTOConfig instance. If not provided, creates one with the given api_key.

download_archive(*printed_metadata*, *destination=None*, *file_name=None*, *overwrite=False*)

Download Printed Metadata (XML data).

These are XML files of the patent as printed. Auto-extracts if the server sends a TAR/ZIP archive.

> **Note**
>
> See also *download_publication()* for a clearer method name with identical functionality.

Parameters

- **printed_metadata** (*PrintedMetaData*) – ArchiveMetaData object containing download URL and metadata
- **destination** (Optional[str]) – Optional directory path to save the file
- **file_name** (Optional[str]) – Optional filename. If not provided, uses Content-Disposition header

- **overwrite** (bool) – Whether to overwrite existing files. Default False

Returns
Path to the downloaded file (extracted if was in archive)

Return type
str

Raises

- **ValueError** – If printed_metadata has no download URL
- **FileExistsError** – If file exists and overwrite=False

download_document(*document*, *format=DocumentMimeType.PDF*, *destination=None*, *file_name=None*, *overwrite=False*)

Download document in specified format.

Automatically extracts if USPTO sends TAR/ZIP.

Parameters

- **document** (*Document*) – Document with document_formats list
- **format** (str | *DocumentMimeType*) – Which format (PDF, XML, MS_WORD). Can be string or DocumentMimeType enum. Defaults to PDF.
- **destination** (Optional[str]) – Directory to save to (default: current directory)
- **file_name** (Optional[str]) – Override filename (default: from Content-Disposition)
- **overwrite** (bool) – Overwrite existing file

Return type
str

Returns
Path to downloaded file (extracted if was in archive)

Raises
FormatNotAvailableError – If format not available for this document. The exception includes *requested_format*, *available_formats*, and *document* attributes for programmatic error handling.

Example

```
>>> docs = client.get_application_documents("19312841", document_codes=["CTNF
↪"])
>>> path = client.download_document(docs[0], format="XML")
>>> # Or using enum:
>>> path = client.download_document(docs[0], format=DocumentMimeType.XML)
```

download_publication(*printed_metadata*, *destination=None*, *file_name=None*, *overwrite=False*)

Download a publication XML file (grant or pre-grant publication).

This method downloads publication XML files from PrintedMetaData objects, such as grant documents or pre-grant publications (pgpub). Auto-extracts if the server sends a TAR/ZIP archive.

Parameters

- **printed_metadata** (*PrintedMetaData*) – PrintedMetaData object containing the publication download URL and filename information. Typically obtained from *get_application_associated_documents()* or from PatentFileWrapper's *grant_document_meta_data* or *pg_publication_document_meta_data*.
- **destination** (Optional[str]) – Optional directory path where the file should be saved. If not provided, saves to the current directory. The directory will be created if it doesn't exist.
- **file_name** (Optional[str]) – Optional custom filename. If not provided, uses the *xml_file_name* from the metadata (e.g., "18915708_12307527.xml").
- **overwrite** (bool) – Whether to overwrite an existing file at the destination. Default is False, which raises FileExistsError if file exists.

Returns

Absolute path to the downloaded publication file (extracted if was in archive).

Return type

str

Raises

- **ValueError** – If printed_metadata has no file_location_uri (download URL).
- **FileExistsError** – If the file already exists and overwrite=False.

Examples

Download grant XML to a specific directory (auto-filename):

```
>>> response = client.get_application_by_number("18/915,708")
>>> ifw = response
>>> grant_metadata = ifw.grant_document_meta_data
>>> path = client.download_publication(grant_metadata, destination="./
↪downloads")
>>> print(path)
'./downloads/18915708_12307527.xml'
```

Download pgpub XML with custom filename:

```
>>> pgpub_metadata = ifw.pg_publication_document_meta_data
>>> path = client.download_publication(
...     pgpub_metadata,
...     file_name="my_publication.xml",
...     destination="./downloads"
... )
>>> print(path)
'./downloads/my_publication.xml'
```

Download to current directory:

```
>>> path = client.download_publication(grant_metadata)
>>> print(path)
'./18915708_12307527.xml'
```

get_IFW_metadata(*application_number=None*, *publication_number=None*, *patent_number=None*, *PCT_app_number=None*, *PCT_pub_number=None*)

Retrieve complete patent file wrapper data using common identifiers.

This utility fetches the *PatentFileWrapper*, which contains comprehensive IFW metadata, application details, and more. Provide only one identifier if possible. If multiple are given, they are processed in the order listed in the arguments, and the first successful match is returned.

Parameters

- **application_number** (Optional[str], optional) – USPTO application number (e.g., "16123456"). Checked first (direct lookup).
- **patent_number** (Optional[str], optional) – USPTO patent number (e.g., "11000000"). Checked second (uses search).
- **publication_number** (Optional[str], optional) – USPTO pre-grant publication number (e.g., "20230123456"). Checked third (uses search).
- **PCT_app_number** (Optional[str], optional) – PCT application number. Checked fourth (direct lookup, treated as USPTO app#).
- **PCT_pub_number** (Optional[str], optional) – PCT publication number (e.g., "2023012345"). Checked fifth (uses search).

Returns

A *PatentFileWrapper* object with
comprehensive data if found using one of the identifiers, otherwise None.

Return type
Optional[*PatentFileWrapper*]

get_application_adjustment(*application_number*)

Retrieve patent term adjustment (PTA) data for a specific application.

This method fetches the *PatentTermAdjustmentData* component from the full patent file wrapper. This data includes details on various delay quantities (e.g., A, B, C delays, applicant delays), the total calculated adjustment, and a history of PTA events that influenced the term.

Parameters
application_number (str) – The USPTO application number for which PTA data is being requested (e.g., "16123456").

Returns

A *PatentTermAdjustmentData*
object containing the PTA details if the application is found and has such data. Returns None if the application cannot be found or if PTA data is not available in the response.

Return type
Optional[*PatentTermAdjustmentData*]

get_application_assignment(*application_number*)

Retrieve a list of patent assignments for a specific application.

This method fetches the *assignment_bag* from the patent file wrapper, which contains a list of *Assignment* objects. Each *Assignment* object details an assignment including information such as reel and frame numbers, recording dates, conveyance text, and details about the assignors and assignees.

Parameters
application_number (str) – The USPTO application number for which assignment data is being requested (e.g., "16123456").

Returns

A list of *Assignment* objects, each
representing a recorded assignment for the application. Returns None if the application cannot be found, or if no assignment data is available in the response. An empty list may be returned if the application is found but has no recorded assignments.

Return type
Optional[List[*Assignment*]]

`get_application_associated_documents`(*application_number*)

Retrieve metadata for Pre-Grant Publication and Grant documents.

This method fetches metadata specifically for published documents associated with the patent application, such as Pre-Grant Publications (PGPUBs) and granted patent documents. It does not retrieve the prosecution history documents (see *get_application_documents* for that). The result is a *PrintedPublication* object, which holds *PrintedMetaData* including file URIs and names. Download with download_archive.

Parameters
application_number (str) – The USPTO application number for which associated PGPUB/Grant document metadata is being requested (e.g., "16123456").

Returns

A *PrintedPublication* object
containing *PrintedMetaData* for the Pre-Grant Publication and/or the Grant document, if available. Returns None if the application cannot be found or if no such associated document metadata is available. The fields within the returned object (*pgpub_document_meta_data*, *grant_document_meta_data*) may themselves be None if a particular type of document (e.g., PGPUB) does not exist for the application.

Return type
Optional[*PrintedPublication*]

`get_application_attorney`(*application_number*)

Retrieve data for the attorney(s) of record for a specific application.

This method fetches the *RecordAttorney* object associated with the patent application. This object contains details about the attorney(s) of record, including customer number correspondence data, power of attorney information, and a list of listed attorneys.

Parameters
application_number (str) – The USPTO application number for which attorney data is being requested (e.g., "16123456").

Returns

A *RecordAttorney* object with details
about the attorney(s) of record if the application is found and such data exists. Returns None if the application cannot be found or if no attorney data is available in the response.

Return type
Optional[*RecordAttorney*]

`get_application_by_number`(*application_number*)

Retrieve the full details for a specific patent application by its number.

This method fetches comprehensive information for a single patent application identified by its unique application number.

Parameters
application_number (str) – The USPTO application number for the patent application

(e.g., "16123456" or "18/915,708"). The application number will be automatically sanitized to remove commas and spaces.

Returns

A *PatentFileWrapper* object representing
the complete file wrapper for the application if found. This object contains all data sections related to the application, such as metadata, addresses, assignments, attorney/agent data, continuity data, PTA/PTE data, transactions, and associated documents. Returns None if the application cannot be found or if the response does not contain the expected data.

Return type
Optional[*PatentFileWrapper*]

`get_application_continuity`(*application_number*)

Retrieve continuity data (parent/child applications) for a specific application.

This method fetches the lineage of the specified application, returning an *ApplicationContinuityData* object. This object consolidates lists of *ParentContinuity* (applications to which the current one claims priority) and *ChildContinuity* (applications claiming priority to the current one) objects, each detailing the related application's key identifiers and status.

Parameters
application_number (str) – The USPTO application number for which continuity data is being requested (e.g., "16123456").

Returns

An *ApplicationContinuityData*
object containing lists of parent and child continuity relationships. Returns None if the application cannot be found or if the underlying data to construct continuity is not available. The lists within the returned object may be empty if no parent or child continuity links exist.

Return type
Optional[*ApplicationContinuityData*]

`get_application_documents`(*application_number*, *document_codes=None*, *official_date_from=None*, *official_date_to=None*)

Retrieve metadata for documents associated with a specific application.

This method fetches a collection of document metadata related to the given patent application. The result is a *DocumentBag* object, which is an iterable collection of *Document* instances. Each *Document* object contains metadata such as its identifier, official date, document code and description, direction (incoming/outgoing), and available download formats.

Parameters

- **application_number** (str) – The USPTO application number for which document metadata is being requested (e.g., "16123456").
- **document_codes** (Optional[List[str]]) – Filter by specific document type codes. If provided, only documents with these codes will be returned. Examples: ['ABST', 'CLM', 'SPEC', 'DRWD'].
- **official_date_from** (Optional[str]) – Filter documents from this date (inclusive). Date format: YYYY-MM-DD (e.g., "2020-01-15").
- **official_date_to** (Optional[str]) – Filter documents to this date (inclusive). Date format: YYYY-MM-DD (e.g., "2023-12-31").

Returns

A *DocumentBag* object containing metadata for all
publicly available documents associated with the application that match the provided filters. The bag will be empty if no documents are found or if the API response indicates no documents. It does not return None for "not found" cases; an empty collection is returned instead.

Return type
DocumentBag

`get_application_foreign_priority`(*application_number*)

Retrieve a list of foreign priority claims for a specific application.

This method fetches the *foreign_priority_bag* from the patent file wrapper. This bag contains a list of *ForeignPriority* objects, each representing a claim to a foreign patent application's priority date. Details include the IP office name, filing date, and application number of the foreign priority application.

Parameters
application_number (str) – The USPTO application number for which foreign priority data is being requested (e.g., "16123456").

Returns

A list of *ForeignPriority* objects,
each detailing a claimed foreign priority. Returns None if the application cannot be found or if no foreign priority data is available. An empty list may be returned if the application is found but has no foreign priority claims.

Return type
Optional[List[*ForeignPriority*]]

`get_application_metadata`(*application_number*)

Retrieve key metadata for a specific patent application.

This method fetches the *ApplicationMetaData* component from the full patent file wrapper. The metadata includes a wide range of information such as application status, important dates (filing, grant, publication), applicant and inventor details, classification data, and other core identifying information for the application.

Parameters
application_number (str) – The USPTO application number for which metadata is being requested (e.g., "16123456" or "18/915,708"). The application number will be automatically sanitized.

Returns

An *ApplicationMetaData* object
containing the core details of the patent application if found. Returns None if the application cannot be found or if metadata is not available in the response.

Return type
Optional[*ApplicationMetaData*]

`get_application_transactions`(*application_number*)

Retrieve the transaction history (events) for a specific application.

This method fetches the *event_data_bag* from the patent file wrapper. This bag contains a list of *EventData* objects, each representing a single recorded event in the prosecution history of the patent application. Events include details like an event code, a textual description, and the date the event was recorded.

Parameters
application_number (str) – The USPTO application number for which transaction history is being requested (e.g., "16123456").

Returns

A list of *EventData* objects, each
detailing a transaction or event in the application's history. Returns None if the application cannot be found or if no transaction data is available. An empty list may be returned if the application is found but has no recorded transaction events.

Return type
Optional[List[*EventData*]]

get_search_results(*query=None*, *sort=None*, *offset=0*, *limit=25*, *fields_param=None*, *filters_param=None*, *range_filters_param=None*, *post_body=None*, *application_number_q=None*, *patent_number_q=None*, *inventor_name_q=None*, *applicant_name_q=None*, *assignee_name_q=None*, *filing_date_from_q=None*, *filing_date_to_q=None*, *grant_date_from_q=None*, *grant_date_to_q=None*, *classification_q=None*, *additional_query_params=None*)

Fetch a dataset of patent applications based on search criteria, always requesting JSON format.

For GET, parameters align with OpenAPI for /api/v1/patent/applications/search/download. For POST, post_body should conform to PatentDownloadRequest schema.

Return type
list[*ApplicationMetaData*]

get_status_codes(*params=None*)

Retrieve USPTO patent application status codes and their descriptions.

This method fetches a list of defined USPTO patent application status codes (e.g., codes for "Pending," "Abandoned," "Issued") using a GET request. The request can be customized with query parameters to filter or paginate the results if supported by the API endpoint.

Parameters
params (Optional[Dict[str, Any]], optional) – A dictionary of query parameters to be sent with the GET request. These parameters can be used to filter or control the output of the status codes list. Defaults to None, which typically retrieves all available status codes or the API's default set.

Returns

An object containing a count of matching
status codes, a *StatusCodeCollection* of the *StatusCode* objects (code and description), and a request identifier.

Return type
StatusCodeSearchResponse

paginate_applications(*post_body=None*, ***kwargs*)

Provide an iterator to easily paginate through patent application search results.

This method simplifies the process of fetching all patent applications that match a given search query by automatically handling pagination. Supports both GET and POST requests.

For GET requests, provide search parameters as keyword arguments. For POST requests, provide the search criteria in *post_body*.

The *offset* and *limit* parameters are managed by the pagination logic; setting them directly in *kwargs* or *post_body* might lead to unexpected behavior.

Parameters

- **post_body** (Optional[dict[str, Any]]) – Optional POST body for complex search queries. If provided, performs POST-based pagination.

- ****kwargs** (Any) – Keyword arguments for GET-based pagination or additional query parameters for POST requests.

Returns

An iterator that yields *PatentFileWrapper*
objects, allowing iteration over all matching patent applications across multiple pages of results.

Return type
Iterator[*PatentFileWrapper*]

Examples

GET-based pagination for wrapper in client.paginate_applications(

query="applicationNumberText:16*", limit=50

):

print(wrapper.application_number_text)

POST-based pagination for wrapper in client.paginate_applications(

post_body={
"q": "applicationNumberText:16*", "facets": "true", "fields": "applicationNumberText,applicationMetaData"

}

):

print(wrapper.application_number_text)

sanitize_application_number(*input_number*)

Sanitize and validate a USPTO application number.

Application numbers are either: - 8 digits (e.g., "16123456") - Series code format: 2 digits + "/" + 6 digits (e.g., "08/123456") - PCT format: "PCT/US2024/012345" → "PCTUS2412345"

This method removes common separators (commas, spaces) while preserving the "/" in series code format.

Parameters
input_number (str) – Raw application number input. May include commas, spaces, or other formatting.

Returns
Sanitized application number (either "NNNNNNNN" or "NN/NNNNNN").

Return type
str

Raises
ValueError – If the format is invalid.

Examples

```
>>> client.sanitize_application_number("16123456")
"16123456"
>>> client.sanitize_application_number("16,123,456")
```

```
"16123456"
>>> client.sanitize_application_number("08/123456")
"08/123456"
>>> client.sanitize_application_number("08/123,456")
"08/123456"
```

search_applications(*query=None*, *sort=None*, *offset=0*, *limit=25*, *facets=None*, *fields=None*, *filters=None*, *range_filters=None*, *post_body=None*, *application_number_q=None*, *patent_number_q=None*, *inventor_name_q=None*, *applicant_name_q=None*, *assignee_name_q=None*, *filing_date_from_q=None*, *filing_date_to_q=None*, *grant_date_from_q=None*, *grant_date_to_q=None*, *classification_q=None*, *earliestPublicationNumber_q=None*, *pctPublicationNumber_q=None*, *additional_query_params=None*)

Search for patent applications.

Can perform a GET request based on OpenAPI query parameters or a POST request if post_body is specified.

Return type
PatentDataResponse

search_status_codes(*search_request*)

Search USPTO patent application status codes using POST criteria.

Performs targeted searches for USPTO patent application status codes (e.g., for "Pending," "Abandoned," "Issued") by sending a POST request with a JSON body containing the *search_request* criteria. This method is suited for more complex queries than the GET-based *get_status_codes*.

Parameters
search_request (Dict[str, Any]) – A dictionary with search criteria, sent as the JSON POST body. The structure must conform to USPTO API requirements for this endpoint (e.g., for searching by code or description keywords).

Returns

An object containing a count of matching
status codes, a *StatusCodeCollection* of the *StatusCode* objects (code and description), and a request identifier.

Return type
StatusCodeSearchResponse

clients.petition_decisions - Client for USPTO Final Petition Decisions API.

This module provides a client for interacting with the USPTO Final Petition Decisions API. It allows you to search for and retrieve final agency petition decisions in publicly available patent applications and patents filed in 2001 or later.

class pyUSPTO.clients.petition_decisions.**FinalPetitionDecisionsClient**(*api_key=None*, *base_url=None*, *config=None*)

Bases: BaseUSPTOClient[PetitionDecisionResponse]

Client for interacting with the USPTO Final Petition Decisions API.

This client provides methods to search for petition decisions, retrieve specific decisions by ID, download decision data, and download associated documents.

Final petition decisions data are incrementally added to the USPTO Open Data Portal on a monthly basis starting with data from 2022 and later.

```
ENDPOINTS = {'download_decisions':  'api/v1/petition/decisions/search/download',
'get_decision_by_id':
'api/v1/petition/decisions/{petitionDecisionRecordIdentifier}', 'search_decisions':
'api/v1/petition/decisions/search'}
```

__init__(*api_key=None*, *base_url=None*, *config=None*)

Initialize the FinalPetitionDecisionsClient.

Parameters

- **api_key** (Optional[str]) – Optional API key for authentication.
- **base_url** (Optional[str]) – Optional base URL override for the API.
- **config** (Optional[*USPTOConfig*]) – Optional USPTOConfig instance for configuration.

download_decisions(*format='json'*, *query=None*, *sort=None*, *offset=None*, *limit=None*, *fields=None*, *filters=None*, *range_filters=None*, *application_number_q=None*, *patent_number_q=None*, *inventor_name_q=None*, *applicant_name_q=None*, *decision_date_from_q=None*, *decision_date_to_q=None*, *additional_query_params=None*, *file_name=None*, *destination=None*, *overwrite=False*)

Download petition decisions data in the specified format.

This endpoint is designed for bulk downloads of petition decisions data. It supports JSON and CSV formats.

Parameters

- **format** (str) – Download format, either "json" or "csv". Defaults to "json".
- **query** (Optional[str]) – Direct query string in USPTO search syntax.
- **sort** (Optional[str]) – Sort order for results.
- **offset** (Optional[int]) – Number of records to skip (pagination).
- **limit** (Optional[int]) – Maximum number of records to return.
- **fields** (Optional[str]) – Specific fields to return.
- **filters** (Optional[str]) – Filter configuration string.
- **range_filters** (Optional[str]) – Range filter configuration string.
- **application_number_q** (Optional[str]) – Filter by application number.
- **patent_number_q** (Optional[str]) – Filter by patent number.
- **inventor_name_q** (Optional[str]) – Filter by inventor name.
- **applicant_name_q** (Optional[str]) – Filter by applicant name.
- **decision_date_from_q** (Optional[str]) – Filter decisions from this date (YYYY-MM-DD).
- **decision_date_to_q** (Optional[str]) – Filter decisions to this date (YYYY-MM-DD).
- **additional_query_params** (Optional[dict[str, Any]]) – Additional custom query parameters.
- **file_name** (Optional[str]) – Optional filename for CSV downloads. Defaults to "petition_decisions.csv".
- **destination** (Optional[str]) – Optional directory path to save CSV file. If None, returns Response.

- **overwrite** (bool) – Whether to overwrite existing files. Default False.

Returns

- If format="json": Returns PetitionDecisionDownloadResponse
- If format="csv" and destination is None: Returns streaming Response
- If format="csv" and destination is set: Returns str path to saved file

Return type
Union[PetitionDecisionDownloadResponse, requests.Response, str]

Raises
FileExistsError – If CSV file exists and overwrite=False

Examples

Download as JSON >>> download = client.download_decisions(… format="json", … technology_center_q="1700", … limit=1000 …) >>> for decision in download.petition_decision_data: … print(decision.application_number_text)

Download CSV and save to file >>> file_path = client.download_decisions(… format="csv", … decision_date_from_q="2023-01-01", … destination="./downloads" …) >>> print(f"Saved to: {file_path}")

Download CSV as streaming response (advanced usage) >>> response = client.download_decisions(format="csv") >>> with open("decisions.csv", "wb") as f: … for chunk in response.iter_content(chunk_size=8192): … f.write(chunk)

download_petition_document(*download_option*, *destination=None*, *file_name=None*, *overwrite=False*)

Download petition document (auto-extracts if in archive).

Parameters

- **download_option** (DocumentDownloadOption) – DocumentDownloadOption object containing the download URL and metadata.
- **destination** (Optional[str]) – Optional directory path where the file should be saved. If not provided, saves to the current directory.
- **file_name** (Optional[str]) – Optional filename for the downloaded file. If not provided, it will be extracted from Content-Disposition header or URL.
- **overwrite** (bool) – Whether to overwrite an existing file. Defaults to False.

Returns
The absolute path to the downloaded file (extracted if was in archive).

Return type
str

Raises

- **ValueError** – If download_option has no download URL.
- **FileExistsError** – If the file exists and overwrite=False.

Examples

Download first document from a decision >>> decision = client.get_decision_by_id(… "34044333-4b40-515f-a684-2515325c57c5", … include_documents=True …) >>> if decision.document_bag: … doc = decision.document_bag[0] … if doc.download_option_bag: … # Download PDF version … pdf_option = next(… opt for opt in doc.download_option_bag … if opt.mime_type_identifier == "PDF" …) … path = client.download_petition_document(… pdf_option, … destination="./downloads" …) … print(f"Downloaded to: {path}")

`get_decision_by_id`(*petition_decision_record_identifier*, *include_documents=None*)

Retrieve a specific petition decision by its record identifier.

Parameters

- **petition_decision_record_identifier** (str) – The unique identifier for the petition decision record (UUID format).
- **include_documents** (Optional[bool]) – Whether to include associated documents in the response. If True, adds includeDocuments=true query parameter.

Returns

The petition decision if found, None otherwise.

Return type

Optional[PetitionDecision]

Examples

Get decision without documents >>> decision = client.get_decision_by_id(… "9f1a4a2b-eee1-58ec-a3aa-167c4075aed4" …)

Get decision with documents >>> decision = client.get_decision_by_id(… "34044333-4b40-515f-a684-2515325c57c5", … include_documents=True …)

`paginate_decisions`(*post_body=None*, ***kwargs*)

Provide an iterator to paginate through petition decision search results.

This method simplifies fetching all petition decisions matching a search query by automatically handling pagination. Supports both GET and POST requests.

The offset and limit parameters are managed by the pagination logic; setting them directly in kwargs or post_body might lead to unexpected behavior.

Parameters

- **post_body** (Optional[dict[str, Any]]) – Optional POST body for complex search queries
- ****kwargs** (Any) – Keyword arguments for GET-based pagination

Returns

An iterator yielding PetitionDecision objects,
allowing iteration over all matching petition decisions across multiple pages of results.

Return type

Iterator[PetitionDecision]

Examples

GET pagination through all decisions for a technology center >>> for decision in client.paginate_decisions(technology_center_q="1700"): ... print(f"{decision.application_number_text}: {decision.decision_type_code}")

POST pagination with date range >>> for decision in client.paginate_decisions(... post_body={ ... "decision_date_from_q": "2023-01-01", ... "decision_date_to_q": "2023-12-31" ... } ...): ... process_decision(decision)

search_decisions(*query=None*, *sort=None*, *offset=0*, *limit=25*, *facets=None*, *fields=None*, *filters=None*, *range_filters=None*, *post_body=None*, *application_number_q=None*, *patent_number_q=None*, *inventor_name_q=None*, *applicant_name_q=None*, *invention_title_q=None*, *decision_type_code_q=None*, *decision_date_from_q=None*, *decision_date_to_q=None*, *petition_mail_date_from_q=None*, *petition_mail_date_to_q=None*, *technology_center_q=None*, *final_deciding_office_name_q=None*, *additional_query_params=None*)

Return final petition decisions matching the given criteria.

This method can perform either a GET request using query parameters or a POST request if post_body is specified. When using GET, you can provide either a direct query string or use convenience parameters that will be automatically combined into a query.

Parameters

- **query** (Optional[str]) – Direct query string in USPTO search syntax.
- **sort** (Optional[str]) – Sort order for results.
- **offset** (int | None) – Number of records to skip (pagination).
- **limit** (int | None) – Maximum number of records to return.
- **facets** (Optional[str]) – Facet configuration string.
- **fields** (Optional[str]) – Specific fields to return.
- **filters** (Optional[str]) – Filter configuration string.
- **range_filters** (Optional[str]) – Range filter configuration string.
- **post_body** (Optional[dict[str, Any]]) – Optional POST body for complex queries.
- **application_number_q** (Optional[str]) – Filter by application number.
- **patent_number_q** (Optional[str]) – Filter by patent number.
- **inventor_name_q** (Optional[str]) – Filter by inventor name.
- **applicant_name_q** (Optional[str]) – Filter by applicant name.
- **invention_title_q** (Optional[str]) – Filter by invention title.
- **decision_type_code_q** (Optional[str]) – Filter by decision type code.
- **decision_date_from_q** (Optional[str]) – Filter decisions from this date (YYYY-MM-DD).
- **decision_date_to_q** (Optional[str]) – Filter decisions to this date (YYYY-MM-DD).
- **petition_mail_date_from_q** (Optional[str]) – Filter petition mail dates from (YYYY-MM-DD).

- **petition_mail_date_to_q** (Optional[str]) – Filter petition mail dates to (YYYY-MM-DD).
- **technology_center_q** (Optional[str]) – Filter by technology center.
- **final_deciding_office_name_q** (Optional[str]) – Filter by deciding office name.
- **additional_query_params** (Optional[dict[str, Any]]) – Additional custom query parameters.

Returns
Response containing matching petition decisions.

Return type
PetitionDecisionResponse

Examples

Search with direct query >>> response = client.search_decisions(query="applicationNumberText:17765301")

Search with convenience parameters >>> response = client.search_decisions(… applicant_name_q="ACME Corp", … decision_date_from_q="2022-01-01", … limit=50 …)

Search with POST body >>> response = client.search_decisions(… post_body={"q": "technologyCenter:1700", "limit": 100} …)

clients.ptab_appeals - Client for USPTO PTAB Appeals API.

This module provides a client for interacting with the USPTO PTAB (Patent Trial and Appeal Board) Appeals API. It allows you to search for ex parte appeal decisions.

class pyUSPTO.clients.ptab_appeals.**PTABAppealsClient**(*api_key=None*, *base_url=None*, *config=None*)

Bases: BaseUSPTOClient[*PTABAppealResponse*]

Client for interacting with the USPTO PTAB Appeals API.

This client provides methods to search for ex parte appeal decisions from the Patent Trial and Appeal Board.

Appeals data includes decisions on patent application appeals from the examiner to the PTAB.

ENDPOINTS = {'search_decisions': 'api/v1/patent/appeals/decisions/search'}

__init__(*api_key=None*, *base_url=None*, *config=None*)

Initialize the PTABAppealsClient.

Parameters

- **api_key** (Optional[str]) – Optional API key for authentication.
- **base_url** (Optional[str]) – Optional base URL override for the API.
- **config** (Optional[*USPTOConfig*]) – Optional USPTOConfig instance for configuration.

download_appeal_archive(*appeal_meta_data*, *destination=None*, *file_name=None*, *overwrite=False*)

Download appeal archive (ZIP/TAR) without extraction.

Parameters

- **appeal_meta_data** (*AppealMetaData*) – AppealMetaData with file_download_uri
- **destination** (Optional[str]) – Directory to save to
- **file_name** (Optional[str]) – Override filename

- **overwrite** (bool) – Overwrite existing file

Return type
str

Returns
Path to downloaded archive file

Raises
ValueError – If appeal_meta_data has no file_download_uri

download_appeal_document(*document_data*, *destination=None*, *file_name=None*, *overwrite=False*)

Download individual appeal document (auto-extracts if needed).

Parameters

- **document_data** (*AppealDocumentData*) – AppealDocumentData with file_download_uri
- **destination** (Optional[str]) – Directory to save to
- **file_name** (Optional[str]) – Override filename
- **overwrite** (bool) – Overwrite existing file

Return type
str

Returns
Path to downloaded file

Raises
ValueError – If document_data has no file_download_uri

download_appeal_documents(*appeal_meta_data*, *destination=None*, *overwrite=False*)

Download and extract all appeal documents.

Parameters

- **appeal_meta_data** (*AppealMetaData*) – AppealMetaData with file_download_uri
- **destination** (Optional[str]) – Directory to save/extract to
- **overwrite** (bool) – Overwrite existing files

Return type
str

Returns
Path to extraction directory

Raises
ValueError – If appeal_meta_data has no file_download_uri

paginate_decisions(*post_body=None*, ***kwargs*)

Provide an iterator to paginate through appeal decision search results.

This method simplifies fetching all appeal decisions matching a search query by automatically handling pagination. It internally calls the search_decisions method, batching results and yielding them one by one.

Supports both GET and POST requests. For POST requests, provide the search criteria in *post_body*. For GET requests, use keyword arguments.

The offset parameter is managed by the pagination logic and should not be provided by the user. The limit parameter can be customized.

Parameters

- **post_body** (Optional[dict[str, Any]]) – Optional POST body for complex search queries.
- ****kwargs** (Any) – Keyword arguments passed to search_decisions for constructing the search query (for GET-based pagination).

Returns

An iterator yielding PTABAppealDecision objects,
allowing iteration over all matching decisions across multiple pages of results.

Return type
Iterator[*PTABAppealDecision*]

Examples

GET-based pagination with convenience parameters >>> for decision in client.paginate_decisions(technology_center_number_q="3600"): ... print(f"{decision.appeal_meta_data.appeal_number}: " ... f"{decision.decision_data.decision_type_category}")

GET-based pagination with date range and custom limit >>> for decision in client.paginate_decisions(... decision_date_from_q="2023-01-01", ... decision_date_to_q="2023-12-31", ... limit=50 ...): ... process_decision(decision)

POST-based pagination >>> for decision in client.paginate_decisions(... post_body={"q": "decisionTypeCategory:Affirmed", "limit": 100} ...): ... process_decision(decision)

search_decisions(*query=None*, *sort=None*, *offset=0*, *limit=25*, *facets=None*, *fields=None*, *filters=None*, *range_filters=None*, *post_body=None*, *appeal_number_q=None*, *application_number_text_q=None*, *appellant_name_q=None*, *requestor_name_q=None*, *decision_type_category_q=None*, *decision_date_from_q=None*, *decision_date_to_q=None*, *technology_center_number_q=None*, *additional_query_params=None*)

Search for PTAB appeal decisions.

This method can perform either a GET request using query parameters or a POST request if post_body is specified. When using GET, you can provide either a direct query string or use convenience parameters that will be automatically combined into a query.

Parameters

- **query** (Optional[str]) – Direct query string in USPTO search syntax.
- **sort** (Optional[str]) – Sort order for results.
- **offset** (int | None) – Number of records to skip (pagination).
- **limit** (int | None) – Maximum number of records to return.
- **facets** (Optional[str]) – Facet configuration string.
- **fields** (Optional[str]) – Specific fields to return.
- **filters** (Optional[str]) – Filter configuration string.
- **range_filters** (Optional[str]) – Range filter configuration string.
- **post_body** (Optional[dict[str, Any]]) – Optional POST body for complex queries.

- **appeal_number_q** (Optional[str]) – Filter by appeal number.
- **application_number_text_q** (Optional[str]) – Filter by application number.
- **appellant_name_q** (Optional[str]) – Filter by appellant name.
- **requestor_name_q** (Optional[str]) – Filter by requestor name.
- **decision_type_category_q** (Optional[str]) – Filter by decision type category.
- **decision_date_from_q** (Optional[str]) – Filter decisions from this date (YYYY-MM-DD).
- **decision_date_to_q** (Optional[str]) – Filter decisions to this date (YYYY-MM-DD).
- **technology_center_number_q** (Optional[str]) – Filter by technology center number.
- **additional_query_params** (Optional[dict[str, Any]]) – Additional custom query parameters.

Returns
Response containing matching appeal decisions.

Return type
PTABAppealResponse

Examples

Search with direct query >>> response = client.search_decisions(query="appealNumber:2023-001234")

Search with convenience parameters >>> response = client.search_decisions(… technology_center_number_q="3600", … decision_date_from_q="2023-01-01", … limit=50 …)

Search with POST body >>> response = client.search_decisions(… post_body={"q": "decisionTypeCategory:Affirmed", "limit": 100} …)

clients.ptab_interferences - Client for USPTO PTAB Interferences API.

This module provides a client for interacting with the USPTO PTAB (Patent Trial and Appeal Board) Interferences API. It allows you to search for patent interference decisions.

class pyUSPTO.clients.ptab_interferences.**PTABInterferencesClient**(*api_key=None*, *base_url=None*, *config=None*)

Bases: BaseUSPTOClient[*PTABInterferenceResponse*]

Client for interacting with the USPTO PTAB Interferences API.

This client provides methods to search for patent interference decisions from the Patent Trial and Appeal Board.

Interference proceedings are used to determine priority of invention when two or more parties claim the same patentable invention.

ENDPOINTS = {'search_decisions': 'api/v1/patent/interferences/decisions/search'}

__init__(*api_key=None*, *base_url=None*, *config=None*)

Initialize the PTABInterferencesClient.

Parameters

- **api_key** (Optional[str]) – Optional API key for authentication.
- **base_url** (Optional[str]) – Optional base URL override for the API.

- **config** (Optional[*USPTOConfig*]) – Optional USPTOConfig instance for configuration.

download_interference_archive(*interference_meta_data*, *destination=None*, *file_name=None*, *overwrite=False*)

Download interference archive (ZIP/TAR) without extraction.

Parameters

- **interference_meta_data** (*InterferenceMetaData*) – InterferenceMetaData with file_download_uri
- **destination** (Optional[str]) – Directory to save to
- **file_name** (Optional[str]) – Override filename
- **overwrite** (bool) – Overwrite existing file

Return type
str

Returns
Path to downloaded archive file

Raises
ValueError – If interference_meta_data has no file_download_uri

download_interference_document(*document_data*, *destination=None*, *file_name=None*, *overwrite=False*)

Download individual interference document (auto-extracts if needed).

Parameters

- **document_data** (*InterferenceDocumentData*) – InterferenceDocumentData with file_download_uri
- **destination** (Optional[str]) – Directory to save to
- **file_name** (Optional[str]) – Override filename
- **overwrite** (bool) – Overwrite existing file

Return type
str

Returns
Path to downloaded file

Raises
ValueError – If document_data has no file_download_uri

download_interference_documents(*interference_meta_data*, *destination=None*, *overwrite=False*)

Download and extract all interference documents.

Parameters

- **interference_meta_data** (*InterferenceMetaData*) – InterferenceMetaData with file_download_uri
- **destination** (Optional[str]) – Directory to save/extract to
- **overwrite** (bool) – Overwrite existing files

Return type
str

Returns
Path to extraction directory

Raises
`ValueError` – If interference_meta_data has no file_download_uri

paginate_decisions(*post_body=None*, ***kwargs*)

Provide an iterator to paginate through interference decision search results.

This method simplifies fetching all interference decisions matching a search query by automatically handling pagination. It internally calls the search_decisions method, batching results and yielding them one by one.

Supports both GET and POST requests. For POST requests, provide the search criteria in *post_body*. For GET requests, use keyword arguments.

The offset parameter is managed by the pagination logic and should not be provided by the user. The limit parameter can be customized.

Parameters

- **post_body** (`Optional[dict[str, Any]]`) – Optional POST body for complex search queries.
- ****kwargs** (`Any`) – Keyword arguments passed to search_decisions for constructing the search query (for GET-based pagination).

Returns

An iterator yielding PTABInterferenceDecision
objects, allowing iteration over all matching decisions across multiple pages of results.

Return type
Iterator[*PTABInterferenceDecision*]

Examples

GET-based pagination through all decisions >>> for decision in client.paginate_decisions(): ... print(f"{decision.interference_meta_data.interference_number}: " ... f"{decision.document_data.interference_outcome_category}")

GET-based pagination with date range and custom limit >>> for decision in client.paginate_decisions(... decision_date_from_q="2020-01-01", ... decision_date_to_q="2023-12-31", ... limit=50 ...): ... process_decision(decision)

POST-based pagination >>> for decision in client.paginate_decisions(... post_body={"q": "interferenceOutcomeCategory:Priority to Senior Party"} ...): ... process_decision(decision)

search_decisions(*query=None*, *sort=None*, *offset=0*, *limit=25*, *facets=None*, *fields=None*, *filters=None*, *range_filters=None*, *post_body=None*, *interference_number_q=None*, *senior_party_application_number_q=None*, *junior_party_application_number_q=None*, *senior_party_name_q=None*, *junior_party_name_q=None*, *real_party_in_interest_q=None*, *interference_outcome_category_q=None*, *decision_type_category_q=None*, *decision_date_from_q=None*, *decision_date_to_q=None*, *additional_query_params=None*)

Search for PTAB interference decisions.

This method can perform either a GET request using query parameters or a POST request if post_body is specified. When using GET, you can provide either a direct query string or use convenience parameters that will be automatically combined into a query.

Parameters

- **query** (Optional[str]) – Direct query string in USPTO search syntax.
- **sort** (Optional[str]) – Sort order for results.
- **offset** (int | None) – Number of records to skip (pagination).
- **limit** (int | None) – Maximum number of records to return.
- **facets** (Optional[str]) – Facet configuration string.
- **fields** (Optional[str]) – Specific fields to return.
- **filters** (Optional[str]) – Filter configuration string.
- **range_filters** (Optional[str]) – Range filter configuration string.
- **post_body** (Optional[dict[str, Any]]) – Optional POST body for complex queries.
- **interference_number_q** (Optional[str]) – Filter by interference number.
- **senior_party_application_number_q** (Optional[str]) – Filter by senior party application number.
- **junior_party_application_number_q** (Optional[str]) – Filter by junior party application number.
- **senior_party_name_q** (Optional[str]) – Filter by senior party name.
- **junior_party_name_q** (Optional[str]) – Filter by junior party name.
- **real_party_in_interest_q** (Optional[str]) – Filter by Real Party in Interest.
- **interference_outcome_category_q** (Optional[str]) – Filter by interference outcome category.
- **decision_type_category_q** (Optional[str]) – Filter by decision type category.
- **decision_date_from_q** (Optional[str]) – Filter decisions from this date (YYYY-MM-DD).
- **decision_date_to_q** (Optional[str]) – Filter decisions to this date (YYYY-MM-DD).
- **additional_query_params** (Optional[dict[str, Any]]) – Additional custom query parameters.

Returns

Response containing matching interference decisions.

Return type

PTABInterferenceResponse

Examples

Search with direct query >>> response = client.search_decisions(query="interferenceNumber:106123")

Search with convenience parameters >>> response = client.search_decisions(… interference_outcome_category_q="Priority to Senior Party", … decision_date_from_q="2020-01-01", … limit=50 …)

```
# Search with POST body >>> response = client.search_decisions( ... post_body={"q": "decision-
TypeCategory:Final Decision", "limit": 100} ... )
```

clients.ptab_trials - Client for USPTO PTAB Trials API.

This module provides a client for interacting with the USPTO PTAB (Patent Trial and Appeal Board) Trials API. It allows you to search for trial proceedings, documents, and decisions.

class pyUSPTO.clients.ptab_trials.**PTABTrialsClient**(*api_key=None*, *base_url=None*, *config=None*)

Bases: BaseUSPTOClient[*PTABTrialProceedingResponse* | *PTABTrialDocumentResponse*]

Client for interacting with the USPTO PTAB Trials API.

This client provides methods to search for trial proceedings, trial documents, and trial decisions from the Patent Trial and Appeal Board.

Trial proceedings data includes IPR (Inter Partes Review), PGR (Post-Grant Review), CBM (Covered Business Method), and DER (Derivation) proceedings.

ENDPOINTS = {'search_decisions': 'api/v1/patent/trials/decisions/search', 'search_documents': 'api/v1/patent/trials/documents/search', 'search_proceedings': 'api/v1/patent/trials/proceedings/search'}

__init__(*api_key=None*, *base_url=None*, *config=None*)

Initialize the PTABTrialsClient.

Parameters

- **api_key** (Optional[str]) – Optional API key for authentication.
- **base_url** (Optional[str]) – Optional base URL override for the API.
- **config** (Optional[*USPTOConfig*]) – Optional USPTOConfig instance for configuration.

download_trial_archive(*trial_meta_data*, *destination=None*, *file_name=None*, *overwrite=False*)

Download trial archive (ZIP/TAR) without extraction.

Parameters

- **trial_meta_data** (*TrialMetaData*) – TrialMetaData with file_download_uri
- **destination** (Optional[str]) – Directory to save to
- **file_name** (Optional[str]) – Override filename
- **overwrite** (bool) – Overwrite existing file

Return type
str

Returns
Path to downloaded archive file

Raises
ValueError – If trial_meta_data has no file_download_uri

download_trial_document(*document_data*, *destination=None*, *file_name=None*, *overwrite=False*)

Download individual trial document (auto-extracts if needed).

Parameters

- **document_data** (*TrialDocumentData*) – TrialDocumentData with file_download_uri

- **destination** (Optional[str]) – Directory to save to
- **file_name** (Optional[str]) – Override filename
- **overwrite** (bool) – Overwrite existing file

Return type
str

Returns
Path to downloaded file

Raises
ValueError – If document_data has no file_download_uri

download_trial_documents(*trial_meta_data*, *destination=None*, *overwrite=False*)

Download and extract all trial documents.

Parameters

- **trial_meta_data** (*TrialMetaData*) – TrialMetaData with file_download_uri
- **destination** (Optional[str]) – Directory to save/extract to
- **overwrite** (bool) – Overwrite existing files

Return type
str

Returns
Path to extraction directory

Raises
ValueError – If trial_meta_data has no file_download_uri

paginate_proceedings(*post_body=None*, ***kwargs*)

Provide an iterator to paginate through trial proceeding search results.

Supports both GET and POST requests.

Parameters

- **post_body** (Optional[dict[str, Any]]) – Optional POST body for complex search queries
- ****kwargs** (Any) – Keyword arguments for GET-based pagination

Yields
PTABTrialProceeding objects

Return type
Iterator[*PTABTrialProceeding*]

search_decisions(*query=None*, *sort=None*, *offset=0*, *limit=25*, *facets=None*, *fields=None*, *filters=None*, *range_filters=None*, *post_body=None*, *trial_number_q=None*, *decision_type_category_q=None*, *document_type_description_q=None*, *decision_date_from_q=None*, *decision_date_to_q=None*, *trial_type_code_q=None*, *patent_number_q=None*, *application_number_q=None*, *patent_owner_name_q=None*, *trial_status_category_q=None*, *real_party_in_interest_name_q=None*, *document_category_q=None*, *additional_query_params=None*)

Search for PTAB trial decisions.

This method can perform either a GET request using query parameters or a POST request if post_body is specified. When using GET, you can provide either a direct query string or use convenience parameters that will be automatically combined into a query.

Parameters

- **query** (Optional[str]) – Direct query string in USPTO search syntax.
- **sort** (Optional[str]) – Sort order for results.
- **offset** (int | None) – Number of records to skip (pagination).
- **limit** (int | None) – Maximum number of records to return.
- **facets** (Optional[str]) – Facet configuration string.
- **fields** (Optional[str]) – Specific fields to return.
- **filters** (Optional[str]) – Filter configuration string.
- **range_filters** (Optional[str]) – Range filter configuration string.
- **post_body** (Optional[dict[str, Any]]) – Optional POST body for complex queries.
- **trial_number_q** (Optional[str]) – Filter by trial number.
- **decision_type_category_q** (Optional[str]) – Filter by decision type category.
- **document_type_description_q** (Optional[str]) – Filter by "*[description]*".
- **decision_date_from_q** (Optional[str]) – Filter decisions from this date (YYYY-MM-DD).
- **decision_date_to_q** (Optional[str]) – Filter decisions to this date (YYYY-MM-DD).
- **trial_type_code_q** (Optional[str]) – Filter by trial type code (e.g., "IPR", "PGR", "CBM", "DER").
- **patent_number_q** (Optional[str]) – Filter by patent number.
- **application_number_q** (Optional[str]) – Filter by application number.
- **patent_owner_name_q** (Optional[str]) – Filter by patent owner name.
- **trial_status_category_q** (Optional[str]) – Filter by trial status category.
- **real_party_in_interest_name_q** (Optional[str]) – Filter by real party in interest name.
- **document_category_q** (Optional[str]) – Filter by document category.
- **additional_query_params** (Optional[dict[str, Any]]) – Additional custom query parameters.

Returns

Response containing matching trial decisions.

Return type

PTABTrialDocumentResponse

Examples

Search with direct query >>> response = client.search_decisions(query="trialNumber:IPR2023-00001")

```
# Search with convenience parameters >>> response = client.search_decisions( ... decision_type_category_q="Final Written Decision", ... decision_date_from_q="2023-01-01", ... limit=50 ... )
```

search_documents(*query=None*, *sort=None*, *offset=0*, *limit=25*, *facets=None*, *fields=None*, *filters=None*, *range_filters=None*, *post_body=None*, *trial_number_q=None*, *document_category_q=None*, *document_type_name_q=None*, *filing_date_from_q=None*, *filing_date_to_q=None*, *petitioner_real_party_in_interest_name_q=None*, *inventor_name_q=None*, *real_party_in_interest_name_q=None*, *patent_number_q=None*, *patent_owner_name_q=None*, *additional_query_params=None*)

Search for PTAB trial documents.

This method can perform either a GET request using query parameters or a POST request if post_body is specified. When using GET, you can provide either a direct query string or use convenience parameters that will be automatically combined into a query.

Parameters

- **query** (Optional[str]) – Direct query string in USPTO search syntax.
- **sort** (Optional[str]) – Sort order for results.
- **offset** (int | None) – Number of records to skip (pagination).
- **limit** (int | None) – Maximum number of records to return.
- **facets** (Optional[str]) – Facet configuration string.
- **fields** (Optional[str]) – Specific fields to return.
- **filters** (Optional[str]) – Filter configuration string.
- **range_filters** (Optional[str]) – Range filter configuration string.
- **post_body** (Optional[dict[str, Any]]) – Optional POST body for complex queries.
- **trial_number_q** (Optional[str]) – Filter by trial number.
- **document_category_q** (Optional[str]) – Filter by document category (e.g., "Petition") DOCUMENTED BUT NOT IN API.
- **document_type_name_q** (Optional[str]) – Filter by document type name (description).
- **filing_date_from_q** (Optional[str]) – Filter documents from this date (YYYY-MM-DD).
- **filing_date_to_q** (Optional[str]) – Filter documents to this date (YYYY-MM-DD).
- **petitioner_real_party_in_interest_name_q** (Optional[str]) – Filter by petitioner real party in interest.
- **inventor_name_q** (Optional[str]) – Filter by inventor name.
- **real_party_in_interest_name_q** (Optional[str]) – Filter by real party in interest (generic).
- **patent_number_q** (Optional[str]) – Filter by patent number.
- **patent_owner_name_q** (Optional[str]) – Filter by patent owner name.
- **additional_query_params** (Optional[dict[str, Any]]) – Additional custom query parameters.

Returns
Response containing matching trial documents.

Return type
PTABTrialDocumentResponse

Examples

Search with direct query >>> response = client.search_documents(query="trialNumber:IPR2023-00001")

Search with convenience parameters >>> response = client.search_documents(… document_category_q="Paper", … filing_date_from_q="2023-01-01", … limit=50 …)

search_proceedings(*query=None*, *sort=None*, *offset=0*, *limit=25*, *facets=None*, *fields=None*, *filters=None*, *range_filters=None*, *post_body=None*, *trial_number_q=None*, *patent_owner_name_q=None*, *petitioner_real_party_in_interest_name_q=None*, *respondent_name_q=None*, *trial_type_code_q=None*, *trial_status_category_q=None*, *petition_filing_date_from_q=None*, *petition_filing_date_to_q=None*, *additional_query_params=None*)

Search for PTAB trial proceedings.

This method can perform either a GET request using query parameters or a POST request if post_body is specified. When using GET, you can provide either a direct query string or use convenience parameters that will be automatically combined into a query.

Parameters

- **query** (Optional[str]) – Direct query string in USPTO search syntax.
- **sort** (Optional[str]) – Sort order for results.
- **offset** (int | None) – Number of records to skip (pagination).
- **limit** (int | None) – Maximum number of records to return.
- **facets** (Optional[str]) – Facet configuration string.
- **fields** (Optional[str]) – Specific fields to return.
- **filters** (Optional[str]) – Filter configuration string.
- **range_filters** (Optional[str]) – Range filter configuration string.
- **post_body** (Optional[dict[str, Any]]) – Optional POST body for complex queries.
- **trial_number_q** (Optional[str]) – Filter by trial number (e.g., "IPR2023-00001").
- **patent_owner_name_q** (Optional[str]) – Filter by patent owner name.
- **petitioner_real_party_in_interest_name_q** (Optional[str]) – Filter by petitioner real party in interest.
- **respondent_name_q** (Optional[str]) – Filter by respondent name.
- **trial_type_code_q** (Optional[str]) – Filter by trial type code (e.g., "IPR", "PGR", "CBM", "DER").
- **trial_status_category_q** (Optional[str]) – Filter by trial status category.
- **petition_filing_date_from_q** (Optional[str]) – Filter proceedings from this date (YYYY-MM-DD).

- **petition_filing_date_to_q** (Optional[str]) – Filter proceedings to this date (YYYY-MM-DD).
- **additional_query_params** (Optional[dict[str, Any]]) – Additional custom query parameters.

Returns
Response containing matching trial proceedings.

Return type
PTABTrialProceedingResponse

Examples

Search with direct query >>> response = client.search_proceedings(query="trialNumber:IPR2023-00001")

Search with convenience parameters >>> response = client.search_proceedings(… trial_type_code_q="IPR", … petition_filing_date_from_q="2023-01-01", … limit=50 …)

3.2 Models

models.bulk_data - Data models for USPTO bulk data API.

This module provides data models for the USPTO Open Data Portal (ODP) Bulk Data API.

class pyUSPTO.models.bulk_data.**BulkDataProduct**(*product_identifier*, *product_description_text*, *product_title_text*, *product_frequency_text*, *product_label_array_text*, *product_dataset_array_text*, *product_dataset_category_array_text*, *product_from_date*, *product_to_date*, *product_total_file_size*, *product_file_total_quantity*, *last_modified_date_time*, *mime_type_identifier_array_text*, *product_file_bag=None*, *days_of_week_text=None*)

Bases: object

Represent a product in the bulk data API.

days_of_week_text: str | None = None

classmethod from_dict(*data*)

Create a BulkDataProduct object from a dictionary.

Return type
BulkDataProduct

last_modified_date_time: str

mime_type_identifier_array_text: list[str]

product_dataset_array_text: list[str]

product_dataset_category_array_text: list[str]

product_description_text: str

product_file_bag: *ProductFileBag* | None = None

product_file_total_quantity: int

product_frequency_text: str

product_from_date: str

product_identifier: str

product_label_array_text: list[str]

product_title_text: str

product_to_date: str

product_total_file_size: int

class pyUSPTO.models.bulk_data.**BulkDataResponse**(*count*, *bulk_data_product_bag*, *raw_data=None*)

Bases: object

Top-level response from the bulk data API.

count

The number of bulk data products in the response.

bulk_data_product_bag

List of bulk data products.

raw_data

Optional raw JSON data from the API response (for debugging).

bulk_data_product_bag: list[*BulkDataProduct*]

count: int

classmethod **from_dict**(*data*, *include_raw_data=False*)

Create a BulkDataResponse object from a dictionary.

Parameters

- **data** (dict[str, Any]) – Dictionary containing API response data.
- **include_raw_data** (bool) – If True, store the raw JSON for debugging.

Returns

An instance of BulkDataResponse.

Return type

BulkDataResponse

raw_data: str | None = None

to_dict()

Convert the BulkDataResponse object to a dictionary.

Return type

dict[str, Any]

class pyUSPTO.models.bulk_data.**FileData**(*file_name*, *file_size*, *file_data_from_date*, *file_data_to_date*, *file_type_text*, *file_release_date*, *file_download_uri=None*, *file_date=None*, *file_last_modified_date_time=None*)

Bases: object

Represent a file in the bulk data API.

file_data_from_date: str

file_data_to_date: str

file_date: str | None = None

file_download_uri: str | None = None

file_last_modified_date_time: str | None = None

file_name: str

file_release_date: str

file_size: int

file_type_text: str

classmethod **from_dict**(*data*)

Create a FileData object from a dictionary.

Return type
FileData

class pyUSPTO.models.bulk_data.**ProductFileBag**(*count*, *file_data_bag*)

Bases: object

Container for file data elements.

count: int

file_data_bag: list[*FileData*]

classmethod **from_dict**(*data*)

Create a ProductFileBag object from a dictionary.

Return type
ProductFileBag

models.patent_data - Data models for USPTO patent data API.

This module provides Pydantic-style data models, primarily using frozen dataclasses, for representing responses from the USPTO Patent Data API. It aims to offer more Pythonic representations (e.g., Enums, native date/datetime objects) of the API's JSON data. Models cover aspects like application metadata, party information (applicants, inventors, attorneys), document details, continuity, assignments, and more.

class pyUSPTO.models.patent_data.**ActiveIndicator**(*value*)

Bases: Enum

Represent an active or inactive status, often used for practitioners or entities.

This Enum is designed to flexibly parse common string representations of active/inactive or true/false states (e.g., "Y", "N", "true", "false", "Active") into standardized Enum members.

ACTIVE = 'Active'

FALSE = 'false'

NO = 'N'

TRUE = 'true'

YES = 'Y'

class pyUSPTO.models.patent_data.**Address**(*name_line_one_text=None*, *name_line_two_text=None*, *address_line_one_text=None*, *address_line_two_text=None*, *address_line_three_text=None*, *address_line_four_text=None*, *geographic_region_name=None*, *geographic_region_code=None*, *postal_code=None*, *city_name=None*, *country_code=None*, *country_name=None*, *postal_address_category=None*, *correspondent_name_text=None*, *country_or_state_code=None*, *ict_state_code=None*, *ict_country_code=None*)

Bases: object

Represent a postal address with fields for street, city, region, country, and postal code.

It can be used for various entities like applicants, inventors, or correspondence.

name_line_one_text

First line of the name (e.g., company name).

name_line_two_text

Second line of the name.

address_line_one_text

First line of the street address.

address_line_two_text

Second line of the street address.

address_line_three_text

Third line of the street address.

address_line_four_text

Fourth line of the street address.

geographic_region_name

Name of the geographic region (e.g., state, province).

geographic_region_code

Code for the geographic region.

postal_code

Postal or ZIP code.

city_name

Name of the city.

country_code

Two-letter country code (e.g., "US").

country_name
Full name of the country (e.g., "United States").

postal_address_category
Category of the address (e.g., "MAILING_ADDRESS").

correspondent_name_text
Name of the correspondent at this address.

country_or_state_code
Country or state code.

ict_state_code
International code for the state/region (USPTO format).

ict_country_code
International code for the country (USPTO format).

address_line_four_text: `str | None = None`

address_line_one_text: `str | None = None`

address_line_three_text: `str | None = None`

address_line_two_text: `str | None = None`

city_name: `str | None = None`

correspondent_name_text: `str | None = None`

country_code: `str | None = None`

country_name: `str | None = None`

country_or_state_code: `str | None = None`

classmethod from_dict(*data*)
Create an *Address* instance from a dictionary representation.

Maps camelCase keys from API data to class attributes.

Parameters
data (`Dict[str, Any]`) – Dictionary containing address data.

Returns
An instance of *Address*.

Return type
Address

geographic_region_code: `str | None = None`

geographic_region_name: `str | None = None`

ict_country_code: `str | None = None`

ict_state_code: `str | None = None`

name_line_one_text: `str | None = None`

name_line_two_text: `str | None = None`

postal_address_category: str | None = None

postal_code: str | None = None

to_dict()

Convert the *Address* instance to a dictionary with camelCase keys.

Returns
A dictionary representation of the address.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**Applicant**(*first_name=None*, *middle_name=None*, *last_name=None*, *name_prefix=None*, *name_suffix=None*, *preferred_name=None*, *country_code=None*, *applicant_name_text=None*, *correspondence_address_bag=<factory>*)

Bases: *Person*

Represent an applicant for a patent, inheriting from Person.

Includes applicant-specific name text and a list of correspondence addresses.

applicant_name_text

The full name of the applicant as a single string.

correspondence_address_bag

A list of *Address* objects for the applicant.

applicant_name_text: str | None = None

correspondence_address_bag: list[*Address*]

classmethod from_dict(*data*)

Create an *Applicant* instance from a dictionary.

Inherits person fields and adds applicant-specific fields.

Parameters
data (Dict[str, Any]) – Dictionary with applicant data.

Returns
An instance of *Applicant*.

Return type
Applicant

to_dict()

Convert the *Applicant* instance to a dictionary.

Includes inherited person fields and applicant-specific fields, using camelCase keys and omitting None values or empty lists.

Returns
Dictionary representation of the applicant.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**ApplicationContinuityData**(*parent_continuity_bag=<factory>*, *child_continuity_bag=<factory>*)

Bases: object

Holds parent and child continuity application data for a specific patent application.

This class consolidates lists of ParentContinuity and ChildContinuity objects, representing the lineage of an application.

parent_continuity_bag

List of *ParentContinuity* objects.

child_continuity_bag

List of *ChildContinuity* objects.

child_continuity_bag: list[*ChildContinuity*]

classmethod **from_wrapper**(*wrapper*)

Create an *ApplicationContinuityData* instance from a *PatentFileWrapper*.

Extracts parent and child continuity bags from the wrapper.

Parameters
wrapper (*PatentFileWrapper*) – The patent file wrapper containing continuity data.

Returns
An instance of *ApplicationContinuityData*.

Return type
ApplicationContinuityData

parent_continuity_bag: list[*ParentContinuity*]

to_dict()

Convert the *ApplicationContinuityData* instance to a dictionary.

Returns

Dictionary representation with "parentContinuityBag"
and "childContinuityBag" keys.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**ApplicationMetaData**(*national_stage_indicator=None*, *entity_status_data=None*, *publication_date_bag=<factory>*, *publication_sequence_number_bag=<factory>*, *publication_category_bag=<factory>*, *docket_number=None*, *first_inventor_to_file_indicator=None*, *first_applicant_name=None*, *first_inventor_name=None*, *application_confirmation_number=None*, *application_status_date=None*, *application_status_description_text=None*, *filing_date=None*, *effective_filing_date=None*, *grant_date=None*, *group_art_unit_number=None*, *application_type_code=None*, *application_type_label_name=None*, *application_type_category=None*, *invention_title=None*, *patent_number=None*, *application_status_code=None*, *earliest_publication_number=None*, *earliest_publication_date=None*, *pct_publication_number=None*, *pct_publication_date=None*, *international_registration_publication_date=None*, *international_registration_number=None*, *examiner_name_text=None*, *class_field=None*, *subclass=None*, *uspc_symbol_text=None*, *customer_number=None*, *cpc_classification_bag=<factory>*, *applicant_bag=<factory>*, *inventor_bag=<factory>*, *raw_data=None*)

Bases: object

Represents the metadata associated with a patent application.

This class holds a wide range of information including application status, dates (filing, grant, publication), applicant and inventor details, classification data, and other identifying information.

national_stage_indicator

Indicates if the application is a national stage entry.

entity_status_data

EntityStatus object detailing applicant's entity status.

publication_date_bag

List of publication dates.

publication_sequence_number_bag

List of publication sequence numbers.

publication_category_bag

List of publication categories.

docket_number

Applicant's or attorney's docket number.

`first_inventor_to_file_indicator`
: Boolean indicating if under First-Inventor-to-File.

`first_applicant_name`
: Name of the first listed applicant.

`first_inventor_name`
: Name of the first listed inventor.

`application_confirmation_number`
: USPTO confirmation number for the application.

`application_status_date`
: Date the current application status was set.

`application_status_description_text`
: Textual description of the current application status.

`filing_date`
: Official filing date of the application.

`effective_filing_date`
: Effective filing date, considering priority claims.

`grant_date`
: Date the patent was granted, if applicable.

`group_art_unit_number`
: USPTO Group Art Unit number.

`application_type_code`
: Code for the application type.

`application_type_label_name`
: Label for the application type (e.g., "Utility").

`application_type_category`
: Category of the application type.

`invention_title`
: Title of the invention.

`patent_number`
: USPTO patent number, if granted.

`application_status_code`
: Numeric code for the application status.

`earliest_publication_number`
: Number of the earliest pre-grant publication.

`earliest_publication_date`
: Date of the earliest pre-grant publication.

`pct_publication_number`
: PCT publication number, if applicable.

pct_publication_date
PCT publication date, if applicable.

international_registration_publication_date
Date of international registration publication.

international_registration_number
International registration number.

examiner_name_text
Name of the patent examiner.

class_field
USPC main classification. (Named *class_field* to avoid keyword clash).

subclass
USPC subclass.

uspc_symbol_text
Full USPC classification symbol.

customer_number
USPTO customer number associated with the application.

cpc_classification_bag
List of CPC classification symbols.

applicant_bag
List of *Applicant* objects.

inventor_bag
List of *Inventor* objects.

raw_data
Raw JSON string of the data used to create this instance (for debugging).

```
applicant_bag: list[Applicant]
```

```
application_confirmation_number: int | None = None
```

```
application_status_code: int | None = None
```

```
application_status_date: date | None = None
```

```
application_status_description_text: str | None = None
```

```
application_type_category: str | None = None
```

```
application_type_code: str | None = None
```

```
application_type_label_name: str | None = None
```

```
class_field: str | None = None
```

```
cpc_classification_bag: list[str]
```

```
customer_number: int | None = None
```

```
docket_number: str | None = None
```

earliest_publication_date: date | None = None

earliest_publication_number: str | None = None

effective_filing_date: date | None = None

entity_status_data: *EntityStatus* | None = None

examiner_name_text: str | None = None

filing_date: date | None = None

first_applicant_name: str | None = None

first_inventor_name: str | None = None

first_inventor_to_file_indicator: bool | None = None

classmethod from_dict(*data*, *include_raw_data=False*)

Create an *ApplicationMetaData* instance from a dictionary.

Parameters

- **data** (Dict[str, Any]) – Dictionary with application metadata.
- **include_raw_data** (bool) – If True, store the raw JSON for debugging.

Returns

An instance of *ApplicationMetaData*.

Return type

ApplicationMetaData

grant_date: date | None = None

group_art_unit_number: str | None = None

international_registration_number: str | None = None

international_registration_publication_date: date | None = None

invention_title: str | None = None

inventor_bag: list[*Inventor*]

property is_aia: bool | None

Returns True if the application is AIA, False if pre-AIA, None if unknown.

property is_pre_aia: bool | None

Returns True if the application is pre-AIA, False if AIA, None if unknown.

national_stage_indicator: bool | None = None

patent_number: str | None = None

pct_publication_date: date | None = None

pct_publication_number: str | None = None

publication_category_bag: list[str]

publication_date_bag: list[date]

publication_sequence_number_bag: list[str]

raw_data: str | None = None

subclass: str | None = None

to_dict()

Convert the *ApplicationMetaData* instance to a dictionary.

Serializes attributes to camelCase keys suitable for API interaction or storage. Omits keys with None values or empty lists. Handles date and boolean serialization.

Returns
Dictionary representation of the application metadata.

Return type
Dict[str, Any]

uspc_symbol_text: str | None = None

class pyUSPTO.models.patent_data.**Assignee**(*assignee_name_text=None*, *assignee_address=None*)

Bases: object

Represent an assignee in a patent assignment.

assignee_name_text

The name of the party receiving the assignment.

assignee_address

The *Address* of the assignee.

assignee_address: *Address* | None = None

assignee_name_text: str | None = None

classmethod from_dict(*data*)

Create an *Assignee* instance from a dictionary.

Parameters
data (Dict[str, Any]) – Dictionary with assignee data.

Returns
An instance of *Assignee*.

Return type
Assignee

to_dict()

Convert the *Assignee* instance to a dictionary.

Omits keys with None values.

Returns
Dictionary representation.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**Assignment**(*reel_number=None*, *frame_number=None*, *reel_and_frame_number=None*, *page_total_quantity=None*, *assignment_document_location_uri=None*, *assignment_received_date=None*, *assignment_recorded_date=None*, *assignment_mailed_date=None*, *conveyance_text=None*, *image_available_status_code=None*, *attorney_docket_number=None*, *assignor_bag=<factory>*, *assignee_bag=<factory>*, *correspondence_address=None*, *domestic_representative=None*)

Bases: object

Represent a patent assignment, detailing the transfer of rights.

Includes information about the reel and frame, document location, dates, conveyance text, and bags of assignors, assignees, correspondence address, and domestic representative.

reel_number

Reel number for the assignment record.

frame_number

Frame number for the assignment record.

reel_and_frame_number

Combined reel and frame number.

page_total_quantity

Total number of pages in the assignment document.

assignment_document_location_uri

URI for the assignment document.

assignment_received_date

Date the assignment was received by USPTO.

assignment_recorded_date

Date the assignment was recorded by USPTO.

assignment_mailed_date

Date the assignment notification was mailed.

conveyance_text

Text describing the nature of the conveyance.

image_available_status_code

Code to indicate the availability of the image.

attorney_docket_number

Attorney docket number for the assignment.

assignor_bag

List of *Assignor* objects.

assignee_bag

List of *Assignee* objects.

correspondence_address

Address object for correspondence (single object).

domestic_representative

Address object for the domestic representative.

assignee_bag: list[*Assignee*]

assignment_document_location_uri: str | None = None

assignment_mailed_date: date | None = None

assignment_received_date: date | None = None

assignment_recorded_date: date | None = None

assignor_bag: list[*Assignor*]

attorney_docket_number: str | None = None

conveyance_text: str | None = None

correspondence_address: *Address* | None = None

domestic_representative: *Address* | None = None

frame_number: int | None = None

classmethod from_dict(*data*)

Create an *Assignment* instance from a dictionary.

Parameters
data (Dict[str, Any]) – Dictionary with assignment data.

Returns
An instance of *Assignment*.

Return type
Assignment

image_available_status_code: bool | None = None

page_total_quantity: int | None = None

reel_and_frame_number: str | None = None

reel_number: int | None = None

to_dict()

Convert the *Assignment* instance to a dictionary.

Returns
Dictionary representation with camelCase keys.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**Assignor**(*assignor_name=None*, *execution_date=None*)

Bases: object

Represent an assignor in a patent assignment.

assignor_name
The name of the assigning party.

execution_date
The date the assignment was executed.

assignor_name: `str | None = None`

execution_date: `date | None = None`

classmethod from_dict(*data*)
Create an *Assignor* instance from a dictionary.

Parameters
data (`Dict[str, Any]`) – Dictionary with assignor data.

Returns
An instance of *Assignor*.

Return type
Assignor

to_dict()
Convert the *Assignor* instance to a dictionary.

Returns
Dictionary representation with camelCase keys.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**Attorney**(*first_name=None*, *middle_name=None*, *last_name=None*, *name_prefix=None*, *name_suffix=None*, *preferred_name=None*, *country_code=None*, *registration_number=None*, *active_indicator=None*, *registered_practitioner_category=None*, *attorney_address_bag=<factory>*, *telecommunication_address_bag=<factory>*)

Bases: *Person*

Represent an attorney or agent associated with a patent application, inheriting from Person.

Includes registration number, active status, practitioner category, addresses, and telecommunication details.

registration_number
The attorney's USPTO registration number.

active_indicator
Indicates if the attorney is currently active (e.g., "Y", "N").

registered_practitioner_category
Category of the practitioner (e.g., "ATTORNEY", "AGENT").

attorney_address_bag
List of *Address* objects for the attorney.

telecommunication_address_bag
List of *Telecommunication* objects for the attorney.

active_indicator: `str | None = None`

attorney_address_bag: list[*Address*]

classmethod from_dict(*data*)

Create an *Attorney* instance from a dictionary.

Inherits person fields and adds attorney-specific details.

Parameters
data (Dict[str, Any]) – Dictionary with attorney data.

Returns
An instance of *Attorney*.

Return type
Attorney

registered_practitioner_category: str | None = None

registration_number: str | None = None

telecommunication_address_bag: list[*Telecommunication*]

to_dict()

Convert the *Attorney* instance to a dictionary.

Includes inherited person fields and attorney-specific fields, using camelCase keys and omitting None values or empty lists.

Returns
Dictionary representation of the attorney.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**ChildContinuity**(*first_inventor_to_file_indicator=None*, *application_number_text=None*, *filing_date=None*, *status_code=None*, *status_description_text=None*, *patent_number=None*, *claim_parentage_type_code=None*, *claim_parentage_type_code_description_text=None*, *child_application_status_code=None*, *parent_application_number_text=None*, *child_application_number_text=None*, *child_application_status_description_text=None*, *child_application_filing_date=None*, *child_patent_number=None*)

Bases: *Continuity*

Represent a child application in a patent application's continuity chain.

Inherits from Continuity and adds specific fields for child application details.

child_application_status_code

Status code of the child application.

parent_application_number_text

Application number of the parent (current) application.

child_application_number_text

Application number of the child application.

child_application_status_description_text

Status description of the child application.

child_application_filing_date

Filing date of the child application.

child_patent_number

Patent number of the child application, if granted.

child_application_filing_date: date | None = None

child_application_number_text: str | None = None

child_application_status_code: int | None = None

child_application_status_description_text: str | None = None

child_patent_number: str | None = None

classmethod from_dict(*data*)

Create a *ChildContinuity* instance from a dictionary.

Parameters
data (Dict[str, Any]) – Dictionary with child continuity data.

Returns
An instance of *ChildContinuity*.

Return type
ChildContinuity

parent_application_number_text: str | None = None

to_dict()

Convert the *ChildContinuity* instance to a dictionary.

Maps attributes to specific camelCase keys expected by the API for child continuity. Filters out None values to match the API response structure.

Returns
Dictionary representation.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**Continuity**(*first_inventor_to_file_indicator=None*, *application_number_text=None*, *filing_date=None*, *status_code=None*, *status_description_text=None*, *patent_number=None*, *claim_parentage_type_code=None*, *claim_parentage_type_code_description_text=None*)

Bases: object

Base class representing continuity data for a patent application.

This includes details about the application's relationship to other applications (parent/child), its filing status under AIA (America Invents Act), and key identifiers.

first_inventor_to_file_indicator

Boolean indicating if the application is under First-Inventor-to-File provisions.

application_number_text
The application number of the related (parent or child) application.

filing_date
The filing date of the related application.

status_code
The status code of the related application.

status_description_text
The status description of the related application.

patent_number
The patent number if the related application is granted.

claim_parentage_type_code
Code indicating the type of continuity claim (e.g., "CON", "DIV").

claim_parentage_type_code_description_text
Description of the continuity claim type.

application_number_text: str | None = None

claim_parentage_type_code: str | None = None

claim_parentage_type_code_description_text: str | None = None

filing_date: date | None = None

first_inventor_to_file_indicator: bool | None = None

property is_aia: bool | None
Returns True if the application is AIA, False if pre-AIA, None if unknown.

property is_pre_aia: bool | None
Returns True if the application is pre-AIA, False if AIA, None if unknown.

patent_number: str | None = None

status_code: int | None = None

status_description_text: str | None = None

to_dict()
Convert the *Continuity* instance to a dictionary.

Omits attributes that are None and property-derived fields. Keys are converted to camelCase.

Returns
A dictionary representation of the continuity data.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**CustomerNumberCorrespondence**(*patron_identifier=None*, *organization_standard_name=None*, *power_of_attorney_address_bag=<factory>*, *telecommunication_address_bag=<factory>*)

Bases: object

Represents correspondence data associated with a USPTO customer number.

Includes patron identifier, organization name, power of attorney addresses, and telecommunication details.

patron_identifier

The USPTO customer number.

organization_standard_name

The name of the organization associated with the customer number.

power_of_attorney_address_bag

List of *Address* objects for power of attorney.

telecommunication_address_bag

List of *Telecommunication* objects.

classmethod from_dict(*data*)

Create a *CustomerNumberCorrespondence* instance from a dictionary.

Parameters
data (Dict[str, Any]) – Dictionary with customer number correspondence data.

Returns
An instance of *CustomerNumberCorrespondence*.

Return type
CustomerNumberCorrespondence

organization_standard_name: str | None = None

patron_identifier: int | None = None

power_of_attorney_address_bag: list[Address]

telecommunication_address_bag: list[Telecommunication]

to_dict()

Convert the *CustomerNumberCorrespondence* instance to a dictionary.

Omits keys with None values or empty lists.

Returns
Dictionary representation.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**DirectionCategory**(*value*)

Bases: Enum

Represents the direction of a document relative to the USPTO (e.g., INCOMING, OUTGOING).

INCOMING = 'INCOMING'

OUTGOING = 'OUTGOING'

class pyUSPTO.models.patent_data.**Document**(*application_number_text=None*, *official_date=None*, *document_identifier=None*, *document_code=None*, *document_code_description_text=None*, *direction_category=None*, *document_formats=<factory>*)

Bases: object

Represent a single document associated with a patent application.

This includes metadata such as its identifier, official date, code, description, direction (incoming/outgoing), and available download formats.

application_number_text

The application number this document belongs to.

official_date

The official date of the document.

document_identifier

A unique identifier for this document.

document_code

A code representing the type of document.

document_code_description_text

A textual description of the document code.

direction_category

The direction of the document (e.g., INCOMING, OUTGOING).

document_formats

A list of available download formats for this document.

__repr__()

Return a developer-friendly string representation of the Document.

Returns
A string showing the document ID, code, and date.

Return type
str

__str__()

Return a human-readable string representation of the Document.

Returns
A description including document ID, code, description, and date.

Return type
str

application_number_text: str | None = None

direction_category: *DirectionCategory* | None = None

document_code: str | None = None

document_code_description_text: str | None = None

document_formats: list[*DocumentFormat*]

document_identifier: str | None = None

classmethod from_dict(*data*)

Create a *Document* instance from a dictionary representation.

Maps API JSON keys (camelCase) to class attributes, parsing nested objects like *DocumentFormat* and *DirectionCategory*.

Parameters
data (Dict[str, Any]) – A dictionary containing document data, typically from an API response.

Returns
An instance of *Document*.

Return type
Document

get_format(*format*)

Get the DocumentFormat object for a specific format.

Parameters
format (str | *DocumentMimeType*) – The format to retrieve. Can be a string (e.g., "XML", "PDF") or a DocumentMimeType enum value.

Return type
DocumentFormat | None

Returns
The DocumentFormat object if found, None otherwise.

Example

```
>>> xml_format = doc.get_format("XML")
>>> if xml_format:
>>>     print(f"XML has {xml_format.page_total_quantity} pages")
```

has_format(*format*)

Check if this document has a specific format available.

Parameters
format (str | *DocumentMimeType*) – The format to check for. Can be a string (e.g., "XML", "PDF") or a DocumentMimeType enum value.

Return type
bool

Returns
True if the document has the specified format, False otherwise.

Example

```
>>> if doc.has_format("XML"):
>>>     client.download_document(doc, format="XML")
```

official_date: datetime | None = None

to_dict()

Convert the *Document* instance to a dictionary for API compatibility.

Serializes attributes to camelCase keys and handles nested objects. Omits keys with None values or empty lists.

Returns
A dictionary representation of the *Document*.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**DocumentBag**(*documents*)

Bases: object

A collection of Document objects associated with a patent application.

Provides iterable access and standard collection methods like *len* and *getitem*. This class is immutable by convention after initialization.

documents

An immutable tuple of *Document* objects.

Type
tuple[Document, ...]

__getitem__(*index*)

Return the document at the specified index.

Parameters
index (int) – The index of the document to retrieve.

Returns
The document at the specified index.

Return type
Document

__init__(*documents*)

Initialize a DocumentBag with a list of documents.

Parameters
documents (List[Document]) – A list of *Document* instances.

__iter__()

Return an iterator over the documents in the collection.

Returns
An iterator of Document instances.

Return type
Iterator[*Document*]

__len__()

Return the number of documents in the collection.

Returns
The count of documents.

Return type
int

__repr__()

Return a detailed string representation for debugging.

Returns
Detailed representation of the DocumentBag.

Return type
str

__str__()

Return a string representation showing document count and summary.

Returns
Human-readable summary of the DocumentBag.

Return type
str

property documents: tuple[*Document*, ...]

Provide access to the tuple of documents.

filter_by_format(*format*)

Filter documents to only include those with a specific format.

Parameters
format (str | *DocumentMimeType*) – The format to filter by. Can be a string (e.g., "XML", "PDF") or a DocumentMimeType enum value.

Return type
DocumentBag

Returns
A new DocumentBag containing only documents that have the specified format.

Example

```
>>> all_docs = client.get_application_documents(app_no)
>>> xml_docs = all_docs.filter_by_format("XML")
>>> for doc in xml_docs:
>>>     client.download_document(doc, format="XML")
```

classmethod from_dict(*data*)

Create a *DocumentBag* instance from a dictionary representation.

Expects a dictionary with a "documentBag" key containing a list of document data dictionaries.

Parameters
data (Dict[str, Any]) – A dictionary, typically from an API response, containing the document bag.

Returns
An instance of *DocumentBag*.

Return type
DocumentBag

`to_dict()`

Convert the *DocumentBag* instance to a dictionary.

Serializes the collection into a dictionary with a "documentBag" key, containing a list of *Document* dictionaries.

Returns
A dictionary representation of the *DocumentBag*.

Return type
Dict[str, Any]

`class pyUSPTO.models.patent_data.DocumentFormat(mime_type_identifier=None, download_url=None, page_total_quantity=None)`

Bases: `object`

Represent an available download format for a specific document.

`mime_type_identifier`

The MIME type of the downloadable file (e.g., "PDF").

`download_url`

The URL from which the document format can be downloaded.

`page_total_quantity`

The total number of pages in this document format.

`__repr__()`

Return a developer-friendly string representation of the DocumentFormat.

Returns
A string showing the mime type and page count.

Return type
str

`__str__()`

Return a human-readable string representation of the DocumentFormat.

Returns
A description of the format type and page count.

Return type
str

`download_url: str | None = None`

`classmethod from_dict(data)`

Create a *DocumentFormat* instance from a dictionary representation.

This factory method is typically used to construct *DocumentFormat* objects from data parsed from an API JSON response. It maps dictionary keys (expected in camelCase) to the class attributes.

Parameters
data (`Dict[str, Any]`) – A dictionary containing the data for a *DocumentFormat*. Expected keys from the API are "mimeTypeIdentifier", "downloadUrl", and "pageTotalQuantity".

Returns

An instance of *DocumentFormat* initialized with
data from the input dictionary.

Return type
DocumentFormat

mime_type_identifier: str | None = None

page_total_quantity: int | None = None

to_dict()

Convert the *DocumentFormat* instance to a dictionary.

This method serializes the *DocumentFormat* object into a dictionary, mapping the instance's attributes to camelCase keys. This is typically useful for generating JSON representations compatible with API expectations.

Returns

A dictionary representation of the *DocumentFormat*
instance with keys "mimeTypeIdentifier", "downloadUrl", and "pageTotalQuantity".

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**DocumentMimeType**(*value*)

Bases: str, Enum

MIME types for document downloads from USPTO.

MS_WORD = 'MS_WORD'

PDF = 'PDF'

XML = 'XML'

__format__(*format_spec*)

Returns format using actual value type unless __str__ has been overridden.

class pyUSPTO.models.patent_data.**EntityStatus**(*small_entity_status_indicator=None*, *business_entity_status_category=None*)

Bases: object

Represents the entity status of an applicant (e.g., small entity status).

small_entity_status_indicator

Boolean indicating if the applicant qualifies for small entity status.

business_entity_status_category

String category of the business entity status (e.g., "Undiscounted").

business_entity_status_category: str | None = None

classmethod from_dict(*data*)

Create an *EntityStatus* instance from a dictionary.

Parameters
data (Dict[str, Any]) – Dictionary with entity status data.

Returns
An instance of *EntityStatus*.

Return type
EntityStatus

small_entity_status_indicator: bool | None = None

to_dict()

Convert the *EntityStatus* instance to a dictionary.

Returns
Dictionary representation with camelCase keys.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**EventData**(*event_code=None*, *event_description_text=None*, *event_date=None*)

Bases: object

Represent a single event in the transaction history of a patent application.

event_code

A code identifying the type of event.

event_description_text

A textual description of the event.

event_date

The date the event was recorded.

event_code: str | None = None

event_date: date | None = None

event_description_text: str | None = None

classmethod from_dict(*data*)

Create an *EventData* instance from a dictionary.

Parameters
data (Dict[str, Any]) – Dictionary with event data.

Returns
An instance of *EventData*.

Return type
EventData

to_dict()

Convert the *EventData* instance to a dictionary.

Omits keys with None values and converts field names to camelCase.

Returns
Dictionary representation.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**ForeignPriority**(*ip_office_name=None*, *filing_date=None*, *application_number_text=None*)

Bases: object

Represent a foreign priority claim for a patent application.

ip_office_name
The name of the intellectual property office of the priority application.

filing_date
The filing date of the priority application.

application_number_text
The application number of the priority application.

application_number_text: `str | None = None`

filing_date: `date | None = None`

classmethod from_dict(*data*)
Create a *ForeignPriority* instance from a dictionary.

Parameters
data (`Dict[str, Any]`) – Dictionary with foreign priority data.

Returns
An instance of *ForeignPriority*.

Return type
ForeignPriority

ip_office_name: `str | None = None`

to_dict()
Convert the *ForeignPriority* instance to a dictionary.

Returns
Dictionary representation with camelCase keys.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**Inventor**(*first_name=None*, *middle_name=None*, *last_name=None*, *name_prefix=None*, *name_suffix=None*, *preferred_name=None*, *country_code=None*, *inventor_name_text=None*, *correspondence_address_bag=<factory>*)

Bases: *Person*

Represent an inventor for a patent application, inheriting from Person.

Includes inventor-specific name text and a list of correspondence addresses.

inventor_name_text
The full name of the inventor as a single string.

correspondence_address_bag
A list of *Address* objects for the inventor.

correspondence_address_bag: `list[Address]`

classmethod from_dict(*data*)
Create an *Inventor* instance from a dictionary.

Inherits person fields and adds inventor-specific fields.

Parameters
data (`Dict[str, Any]`) – Dictionary with inventor data.

Returns
An instance of *Inventor*.

Return type
Inventor

inventor_name_text: str | None = None

to_dict()

Convert the *Inventor* instance to a dictionary.

Includes inherited person fields and inventor-specific fields, using camelCase keys and omitting None values or empty lists.

Returns
Dictionary representation of the inventor.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**ParentContinuity**(*first_inventor_to_file_indicator=None*, *application_number_text=None*, *filing_date=None*, *status_code=None*, *status_description_text=None*, *patent_number=None*, *claim_parentage_type_code=None*, *claim_parentage_type_code_description_text=None*, *parent_application_status_code=None*, *parent_patent_number=None*, *parent_application_status_description_text=None*, *parent_application_filing_date=None*, *parent_application_number_text=None*, *child_application_number_text=None*)

Bases: *Continuity*

Represent a parent application in a patent application's continuity chain.

Inherits from Continuity and adds specific fields for parent application details.

parent_application_status_code
Status code of the parent application.

parent_patent_number
Patent number of the parent application, if granted.

parent_application_status_description_text
Status description of the parent application.

parent_application_filing_date
Filing date of the parent application.

parent_application_number_text
Application number of the parent application.

child_application_number_text
Application number of the child (current) application.

child_application_number_text: str | None = None

classmethod from_dict(*data*)

Create a *ParentContinuity* instance from a dictionary.

Parameters
data (Dict[str, Any]) – Dictionary with parent continuity data.

Returns
An instance of *ParentContinuity*.

Return type
ParentContinuity

parent_application_filing_date: date | None = None

parent_application_number_text: str | None = None

parent_application_status_code: int | None = None

parent_application_status_description_text: str | None = None

parent_patent_number: str | None = None

to_dict()

Convert the *ParentContinuity* instance to a dictionary.

Maps attributes to specific camelCase keys expected by the API for parent continuity. Filters out None values to match the API response structure.

Returns
Dictionary representation.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**PatentDataResponse**(*count*, *patent_file_wrapper_data_bag=<factory>*, *request_identifier=None*, *raw_data=None*)

Bases: object

Represents the overall response from a patent data API request.

It typically includes a count of the results and a list of PatentFileWrapper objects, each containing detailed data for a patent application.

count

The total number of patent applications found matching the query.

patent_file_wrapper_data_bag

A list of *PatentFileWrapper* objects.

request_identifier

An identifier for the API request, if provided.

raw_data

Optional raw JSON data from the API response (for debugging).

count: int

classmethod from_dict(*data*, *include_raw_data=False*)

Create a *PatentDataResponse* instance from a dictionary.

Parameters

- **data** (Dict[str, Any]) – Dictionary with API response data.
- **include_raw_data** (bool) – If True, store the raw JSON for debugging.

Returns
An instance of *PatentDataResponse*.

Return type
PatentDataResponse

patent_file_wrapper_data_bag: list[*PatentFileWrapper*]

raw_data: str | None = None

request_identifier: str | None = None

to_csv()

Convert the patent data in this response to a CSV formatted string.

The CSV will contain key metadata fields for each application, such as invention title, application number, filing date, status, etc.

Returns
A string containing the data in CSV format.

Return type
str

to_dict()

Convert the *PatentDataResponse* instance to a dictionary.

Returns
Dictionary representation.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**PatentFileWrapper**(*application_number_text*, *application_meta_data=None*, *correspondence_address_bag=<factory>*, *assignment_bag=<factory>*, *record_attorney=None*, *foreign_priority_bag=<factory>*, *parent_continuity_bag=<factory>*, *child_continuity_bag=<factory>*, *patent_term_adjustment_data=None*, *event_data_bag=<factory>*, *pgpub_document_meta_data=None*, *grant_document_meta_data=None*, *last_ingestion_date_time=None*)

Bases: object

Represents the complete file wrapper for a single patent application.

This is a top-level object containing all data sections related to an application, such as metadata, addresses, assignments, attorney information, continuity data, PTA data, transaction events, and associated document metadata.

application_number_text
: The primary application number.

application_meta_data
: Comprehensive *ApplicationMetaData*.

correspondence_address_bag
: List of *Address* objects for correspondence.

assignment_bag
: List of *Assignment* records.

record_attorney
: Information about the *RecordAttorney*.

foreign_priority_bag
: List of *ForeignPriority* claims.

parent_continuity_bag
: List of *ParentContinuity* records.

child_continuity_bag
: List of *ChildContinuity* records.

patent_term_adjustment_data
: *PatentTermAdjustmentData* details.

event_data_bag
: List of *EventData* (transaction history).

pgpub_document_meta_data
: *PrintedMetaData* for Pre-Grant Publication.

grant_document_meta_data
: *PrintedMetaData* for the granted patent.

last_ingestion_date_time
: Timestamp of when this data was last ingested by the API (UTC).

application_meta_data: *ApplicationMetaData* | None = None

application_number_text: str

assignment_bag: list[*Assignment*]

child_continuity_bag: list[*ChildContinuity*]

correspondence_address_bag: list[*Address*]

event_data_bag: list[*EventData*]

foreign_priority_bag: list[*ForeignPriority*]

classmethod from_dict(*data*, *include_raw_data=False*)
: Create a *PatentFileWrapper* instance from a dictionary.

  Parameters

  - **data** (`Dict[str, Any]`) – Dictionary with patent file wrapper data.
  - **include_raw_data** (`bool`) – If True, store the raw JSON for debugging.

Returns
An instance of *PatentFileWrapper*.

Return type
PatentFileWrapper

grant_document_meta_data: *PrintedMetaData* | None = None

last_ingestion_date_time: datetime | None = None

parent_continuity_bag: list[*ParentContinuity*]

patent_term_adjustment_data: *PatentTermAdjustmentData* | None = None

pgpub_document_meta_data: *PrintedMetaData* | None = None

record_attorney: *RecordAttorney* | None = None

to_dict()

Convert the *PatentFileWrapper* instance to a dictionary.

Omits keys with None values or empty lists. Serializes nested objects.

Returns
Dictionary representation.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**PatentTermAdjustmentData**(*a_delay_quantity=None*, *adjustment_total_quantity=None*, *applicant_day_delay_quantity=None*, *b_delay_quantity=None*, *c_delay_quantity=None*, *non_overlapping_day_quantity=None*, *overlapping_day_quantity=None*, *non_overlapping_day_delay_quantity=None*, *ip_office_adjustment_delay_quantity=None*, *patent_term_adjustment_history_data_bag=<factory>*)

Bases: object

Represents the overall patent term adjustment (PTA) data for an application.

Includes various delay quantities (A, B, C, applicant, IP office), total adjustment, and a history of PTA events.

a_delay_quantity

Number of days of 'A' delay.

adjustment_total_quantity

Total calculated PTA in days.

applicant_day_delay_quantity

Total days of delay attributable to the applicant.

b_delay_quantity

Number of days of 'B' delay.

c_delay_quantity

Number of days of 'C' delay.

non_overlapping_day_quantity
Number of non-overlapping delay days.

overlapping_day_quantity
Number of overlapping delay days.

non_overlapping_day_delay_quantity
Number of non-overlapping delay days specifically for delay calculation.

ip_office_adjustment_delay_quantity
Days of IP office delay used in adjustment calculation.

patent_term_adjustment_history_data_bag
List of *PatentTermAdjustmentHistoryData* events.

a_delay_quantity: float | None = None

adjustment_total_quantity: float | None = None

applicant_day_delay_quantity: float | None = None

b_delay_quantity: float | None = None

c_delay_quantity: float | None = None

classmethod from_dict(*data*)
Create a *PatentTermAdjustmentData* instance from a dictionary.

Parameters
data (Dict[str, Any]) – Dictionary with PTA data.

Returns
An instance of *PatentTermAdjustmentData*.

Return type
PatentTermAdjustmentData

ip_office_adjustment_delay_quantity: float | None = None

non_overlapping_day_delay_quantity: float | None = None

non_overlapping_day_quantity: float | None = None

overlapping_day_quantity: float | None = None

patent_term_adjustment_history_data_bag: list[*PatentTermAdjustmentHistoryData*]

to_dict()
Convert the *PatentTermAdjustmentData* instance to a dictionary.

Omits keys with None values or empty lists, and converts field names to camelCase.

Returns
Dictionary representation.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**PatentTermAdjustmentHistoryData**(*event_date=None*, *applicant_day_delay_quantity=None*, *event_description_text=None*, *event_sequence_number=None*, *originating_event_sequence_number=None*, *pta_pte_code=None*, *ip_office_day_delay_quantity=None*)

Bases: object

Represent a single entry in the patent term adjustment (PTA) history for an application.

Details specific events, dates, and day quantities affecting the patent term.

event_date
: Date of the PTA event.

applicant_day_delay_quantity
: Number of days of delay attributable to the applicant for this event.

event_description_text
: Textual description of the PTA event.

event_sequence_number
: Sequence number of this event in the PTA history.

originating_event_sequence_number
: Sequence number of an event that originated this event.

pta_pte_code
: Code indicating if the event relates to PTA or Patent Term Extension (PTE).

ip_office_day_delay_quantity
: Number of days of IP office delay used in adjustment calculation for this event.

applicant_day_delay_quantity: float | None = None

event_date: date | None = None

event_description_text: str | None = None

event_sequence_number: float | None = None

classmethod **from_dict**(*data*)

Create a *PatentTermAdjustmentHistoryData* instance from a dictionary.

Parameters
data (Dict[str, Any]) – Dictionary with PTA history event data.

Returns
An instance of *PatentTermAdjustmentHistoryData*.

Return type
PatentTermAdjustmentHistoryData

ip_office_day_delay_quantity: float | None = None

originating_event_sequence_number: float | None = None

pta_pte_code: str | None = None

to_dict()

Convert the *PatentTermAdjustmentHistoryData* instance to a dictionary.

Omits keys with None values.

Returns
Dictionary representation with camelCase keys.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**Person**(*first_name=None*, *middle_name=None*, *last_name=None*, *name_prefix=None*, *name_suffix=None*, *preferred_name=None*, *country_code=None*)

Bases: object

A base data class representing a person with common name and country attributes.

This class is typically inherited by more specific types like Applicant, Inventor, or Attorney.

first_name

The first name of the person.

middle_name

The middle name or initial of the person.

last_name

The last name or surname of the person.

name_prefix

A prefix for the name (e.g., "Dr.", "Mr.").

name_suffix

A suffix for the name (e.g., "Jr.", "PhD").

preferred_name

The person's preferred name, if different.

country_code

The country code associated with the person (e.g., citizenship).

country_code: str | None = None

first_name: str | None = None

last_name: str | None = None

middle_name: str | None = None

name_prefix: str | None = None

name_suffix: str | None = None

preferred_name: str | None = None

to_dict()

Convert the *Person* instance to a dictionary with camelCase keys.

Omits attributes that are None.

Returns
A dictionary representation of the person.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**PrintedMetaData**(*zip_file_name=None*, *product_identifier=None*, *file_location_uri=None*, *file_create_date_time=None*, *xml_file_name=None*)

Bases: object

Represents metadata for a specific archive file, such as a PGPUB or Grant XML file.

zip_file_name

The name of the ZIP archive.

product_identifier

An identifier for the data product (e.g., "APPXML", "PTGRXML").

file_location_uri

The URI where the document file can be accessed.

file_create_date_time

The creation timestamp of the document file (UTC).

xml_file_name

The name of the XML file within the ZIP archive.

file_create_date_time: datetime | None = None

file_location_uri: str | None = None

classmethod from_dict(*data*)

Create a *PrintedMetaData* instance from a dictionary.

Parameters
data (Dict[str, Any]) – Dictionary with printed metadata.

Returns
An instance of *PrintedMetaData*.

Return type
PrintedMetaData

product_identifier: str | None = None

to_dict()

Convert the *PrintedMetaData* instance to a dictionary.

Omits keys with None values. Serializes datetime to ISO format with 'Z'.

Returns
Dictionary representation with camelCase keys.

Return type
Dict[str, Any]

xml_file_name: *str | None = None*

zip_file_name: *str | None = None*

class pyUSPTO.models.patent_data.**PrintedPublication**(*pgpub_document_meta_data=None*, *grant_document_meta_data=None*)

Bases: object

Represent metadata for associated documents such as PGPUB and Grant publications.

> **Note**
>
> PGPUB refers to a Pre-Grant Publication.

pgpub_document_meta_data

PrintedMetaData for the Pre-Grant Publication, if any.

grant_document_meta_data

PrintedMetaData for the Grant document, if any.

classmethod from_wrapper(*wrapper*)

Create a *PrintedPublication* instance from a *PatentFileWrapper*.

Extracts PGPUB and Grant document metadata from the wrapper.

Parameters
wrapper (*PatentFileWrapper*) – The patent file wrapper.

Returns
An instance of *PrintedPublication*.

Return type
PrintedPublication

grant_document_meta_data: *PrintedMetaData | None = None*

pgpub_document_meta_data: *PrintedMetaData | None = None*

to_dict()

Convert the *PrintedPublication* instance to a dictionary.

Omits keys if their corresponding metadata is None.

Returns
Dictionary representation.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**RecordAttorney**(*customer_number_correspondence_data=None*, *power_of_attorney_bag=<factory>*, *attorney_bag=<factory>*)

Bases: object

Represents information about the attorney(s) of record for a patent application.

Contains customer number correspondence data, power of attorney information, and listed attorneys.

customer_number_correspondence_data

CustomerNumberCorrespondence object with customer number details.

power_of_attorney_bag

List of *Attorney* objects named in a power of attorney.

attorney_bag

List of *Attorney* objects listed as attorneys of record.

attorney_bag: list[*Attorney*]

customer_number_correspondence_data: *CustomerNumberCorrespondence* | None = None

classmethod from_dict(*data*)

Create a *RecordAttorney* instance from a dictionary.

Parameters
data (Dict[str, Any]) – Dictionary with record attorney data.

Returns
An instance of *RecordAttorney*.

Return type
RecordAttorney

power_of_attorney_bag: list[*Attorney*]

to_dict()

Convert the *RecordAttorney* instance to a dictionary.

Omits keys with None values. Includes empty lists to match API behavior.

Returns
Dictionary representation.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**StatusCode**(*code=None*, *description=None*)

Bases: object

Represent a USPTO application status code and its textual description.

code

The numeric status code.

description

The textual description of the status code.

__str__()

Return a user-friendly string representation of the status code.

Return type
str

code: int | None = None

description: str | None = None

classmethod from_dict(*data*)

Create a *StatusCode* instance from a dictionary.

Handles two possible key sets from the API for status information.

Parameters
data (Dict[str, Any]) – Dictionary with status code data.

Returns
An instance of *StatusCode*.

Return type
StatusCode

to_dict()

Convert the *StatusCode* instance to a dictionary.

Uses keys "applicationStatusCode" and "applicationStatusDescriptionText" for consistency with some API response parts.

Returns
Dictionary representation.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**StatusCodeCollection**(*status_codes*)

Bases: object

A collection of StatusCode objects.

Provides iterable access and helper methods to find or filter status codes. This class is immutable by convention after initialization.

status_codes

An immutable tuple of *StatusCode* objects.

Type
tuple[StatusCode, ...]

__getitem__(*index*)

Return the status code at the specified index.

Parameters
index (int) – The index of the status code to retrieve.

Returns
The status code at the specified index.

Return type
StatusCode

__init__(*status_codes*)

Initialize a StatusCodeCollection with a list of status codes.

Parameters
status_codes (List[StatusCode]) – A list of *StatusCode* instances.

__iter__()

Return an iterator over the status codes in the collection.

Returns
An iterator of StatusCode instances.

Return type
Iterator[*StatusCode*]

`__len__`()

Return the number of status codes in the collection.

Returns
The count of status codes.

Return type
int

`__repr__`()

Return a developer-friendly string representation of the StatusCodeCollection.

Returns
A string showing the collection size and sample status codes.

Return type
str

`__str__`()

Return a human-readable string representation of the StatusCodeCollection.

Returns
A description of the collection size.

Return type
str

`find_by_code`(*code_to_find*)

Find a status code by its numeric code.

Parameters
code_to_find (int) – The numeric status code to search for.

Returns
The *StatusCode* object if found, otherwise None.

Return type
Optional[*StatusCode*]

`search_by_description`(*text*)

Search for status codes by a case-insensitive text match in their description.

Parameters
text (str) – The text to search for within status code descriptions.

Returns
A new collection containing matching status codes.

Return type
StatusCodeCollection

`to_dict`()

Convert the collection of status codes to a list of dictionaries.

Returns

A list where each item is the dictionary
representation of a *StatusCode*.

Return type
List[Dict[str, Any]]

class pyUSPTO.models.patent_data.**StatusCodeSearchResponse**(*count*, *status_code_bag*, *request_identifier=None*)

Bases: object

Represents the response from a search query for patent application status codes.

count

The total number of status codes found matching the query.

status_code_bag

A *StatusCodeCollection* of the *StatusCode* objects returned.

request_identifier

An identifier for the API request, if provided.

count: int

classmethod from_dict(*data*)

Create a *StatusCodeSearchResponse* instance from a dictionary.

Parameters
data (Dict[str, Any]) – Dictionary with API response data for status codes.

Returns
An instance of *StatusCodeSearchResponse*.

Return type
StatusCodeSearchResponse

request_identifier: str | None = None

status_code_bag: *StatusCodeCollection*

to_dict()

Convert the *StatusCodeSearchResponse* instance to a dictionary.

Omits keys with None values or empty lists.

Returns
Dictionary representation.

Return type
Dict[str, Any]

class pyUSPTO.models.patent_data.**Telecommunication**(*telecommunication_number=None*, *extension_number=None*, *telecom_type_code=None*)

Bases: object

Represent telecommunication details, such as phone or fax numbers.

telecommunication_number

The main number (e.g., phone number).

extension_number

Any extension associated with the number.

telecom_type_code

A code indicating the type of telecommunication (e.g., "TEL", "FAX").

extension_number: str | None = None

classmethod from_dict(*data*)

Create a *Telecommunication* instance from a dictionary.

Parameters
data (Dict[str, Any]) – Dictionary with telecommunication data.

Returns
An instance of *Telecommunication*.

Return type
Telecommunication

telecom_type_code: str | None = None

telecommunication_number: str | None = None

to_dict()

Convert the *Telecommunication* instance to a dictionary.

Returns
Dictionary representation with camelCase keys.

Return type
Dict[str, Any]

models.ptab - Data models for USPTO PTAB (Patent Trial and Appeal Board) APIs.

This module provides data models, primarily using frozen dataclasses, for representing responses from the USPTO PTAB APIs. These models cover: - Patent trial proceedings (IPR, PGR, CBM, DER) - Trial documents and decisions - Appeal decisions - Interference decisions

class pyUSPTO.models.ptab.**AdditionalPartyData**(*application_number_text=None*, *inventor_name=None*, *patent_number=None*, *additional_party_name=None*)

Bases: object

Additional party information in an interference.

application_number_text

Application number.

inventor_name

Name of inventor.

patent_number

Patent number.

additional_party_name

Name of additional party.

additional_party_name: str | None = None

application_number_text: str | None = None

classmethod from_dict(*data*, *include_raw_data=False*)

Create an AdditionalPartyData instance from a dictionary.

Parameters

- **data** (dict[str, Any]) – Dictionary containing additional party data from API response.
- **include_raw_data** (bool) – Ignored for this model.

Returns
An instance of AdditionalPartyData.

Return type
AdditionalPartyData

inventor_name: str | None = None

patent_number: str | None = None

to_dict()

Convert the AdditionalPartyData instance to a dictionary.

Returns
Dictionary with camelCase keys and None values filtered.

Return type
Dict[str, Any]

class pyUSPTO.models.ptab.**AppealDocumentData**(*document_filing_date=None*, *document_identifier=None*, *document_name=None*, *document_size_quantity=None*, *document_ocr_text=None*, *document_type_description_text=None*, *file_download_uri=None*)

Bases: object

Appeal document metadata.

document_filing_date

Date the document was filed.

document_identifier

Unique identifier for the document.

document_name

Name of the document.

document_size_quantity

Size of the document in bytes.

document_ocr_text

Full OCR text of the document.

document_type_description_text

Description of the document type.

file_download_uri

URI to download the document.

document_filing_date: date | None = None

document_identifier: str | None = None

document_name: str | None = None

document_ocr_text: str | None = None

document_size_quantity: int | None = None

document_type_description_text: str | None = None

file_download_uri: str | None = None

classmethod from_dict(*data*, *include_raw_data=False*)

Create an AppealDocumentData instance from a dictionary.

Parameters

- **data** (dict[str, Any]) – Dictionary containing document data from API response.
- **include_raw_data** (bool) – Ignored for this model.

Returns

An instance of AppealDocumentData.

Return type

AppealDocumentData

to_dict()

Convert the AppealDocumentData instance to a dictionary.

Returns

Dictionary with camelCase keys and None values filtered.

Return type

Dict[str, Any]

class pyUSPTO.models.ptab.**AppealMetaData**(*appeal_filing_date=None*, *appeal_last_modified_date=None*, *appeal_last_modified_date_time=None*, *application_type_category=None*, *docket_notice_mailed_date=None*, *file_download_uri=None*)

Bases: object

Appeal metadata.

appeal_filing_date

Date the appeal was filed.

appeal_last_modified_date

Last modification date.

appeal_last_modified_date_time

Last modification timestamp.

application_type_category

Type of application.

docket_notice_mailed_date

Date the docket notice was mailed.

file_download_uri

URI to download ZIP of appeal documents.

appeal_filing_date: date | None = None

appeal_last_modified_date: date | None = None

appeal_last_modified_date_time: datetime | None = None

application_type_category: str | None = None

docket_notice_mailed_date: date | None = None

file_download_uri: str | None = None

classmethod from_dict(*data*, *include_raw_data=False*)

Create an AppealMetaData instance from a dictionary.

Parameters

- **data** (dict[str, Any]) – Dictionary containing appeal metadata from API response.
- **include_raw_data** (bool) – Ignored for this model.

Returns

An instance of AppealMetaData.

Return type

AppealMetaData

to_dict()

Convert the AppealMetaData instance to a dictionary.

Returns

Dictionary with camelCase keys and None values filtered.

Return type

Dict[str, Any]

class pyUSPTO.models.ptab.**AppellantData**(*application_number_text=None*, *counsel_name=None*, *grant_date=None*, *group_art_unit_number=None*, *inventor_name=None*, *real_party_in_interest_name=None*, *patent_number=None*, *patent_owner_name=None*, *technology_center_number=None*, *publication_date=None*, *publication_number=None*)

Bases: PartyData

Appellant party data in PTAB appeals.

Inherits all attributes from PartyData. Used in appeal proceedings to represent the party appealing an examiner decision.

class pyUSPTO.models.ptab.**DecisionData**(*appeal_outcome_category=None*, *statute_and_rule_bag=<factory>*, *decision_issue_date=None*, *decision_type_category=None*, *issue_type_bag=<factory>*)

Bases: object

Appeal decision information.

appeal_outcome_category

Outcome of the appeal.

statute_and_rule_bag

List of applicable statutes and rules.

decision_issue_date

Date the decision was issued.

decision_type_category

Type of decision.

issue_type_bag

List of issue types.

appeal_outcome_category: `str | None = None`

decision_issue_date: `date | None = None`

decision_type_category: `str | None = None`

classmethod from_dict(*data*, *include_raw_data=False*)

Create a DecisionData instance from a dictionary.

Parameters

- **data** (`dict[str, Any]`) – Dictionary containing decision data from API response.
- **include_raw_data** (`bool`) – Ignored for this model.

Returns

An instance of DecisionData.

Return type

DecisionData

issue_type_bag: `list[str]`

statute_and_rule_bag: `list[str]`

to_dict()

Convert the DecisionData instance to a dictionary.

Returns

Dictionary with camelCase keys and None values filtered.

Return type

Dict[str, Any]

class pyUSPTO.models.ptab.**DerivationPetitionerData**(*application_number_text=None*, *counsel_name=None*, *grant_date=None*, *group_art_unit_number=None*, *inventor_name=None*, *real_party_in_interest_name=None*, *patent_number=None*, *patent_owner_name=None*, *technology_center_number=None*, *publication_date=None*, *publication_number=None*)

Bases: `PartyData`

Derivation petitioner data in derivation proceedings.

Inherits all attributes from PartyData. Used in DER proceedings to represent the petitioning party claiming derivation.

class pyUSPTO.models.ptab.**InterferenceDocumentData**(*document_identifier=None*, *document_name=None*, *document_size_quantity=None*, *document_ocr_text=None*, *document_title_text=None*, *interference_outcome_category=None*, *document_filing_date=None*, *decision_issue_date=None*, *decision_type_category=None*, *file_download_uri=None*, *statute_and_rule_bag=<factory>*, *issue_type_bag=<factory>*)

Bases: object

Interference document metadata.

document_identifier

Unique identifier for the document.

document_name

Name of the document.

document_size_quantity

Size of the document in bytes.

document_ocr_text

Full OCR text of the document.

document_title_text

Title of the document.

interference_outcome_category

Outcome of the interference.

document_filing_date

Date the document was filed.

decision_issue_date

Date the decision was issued.

decision_type_category

Type of decision.

file_download_uri

URI to download the document.

statute_and_rule_bag

List of applicable statutes and rules.

issue_type_bag

List of issues addressed.

decision_issue_date: date | None = None

decision_type_category: str | None = None

document_filing_date: date | None = None

document_identifier: str | None = None

document_name: str | None = None

document_ocr_text: str | None = None

document_size_quantity: int | None = None

document_title_text: str | None = None

file_download_uri: str | None = None

classmethod from_dict(*data*, *include_raw_data=False*)

Create an InterferenceDocumentData instance from a dictionary.

Parameters

- **data** (dict[str, Any]) – Dictionary containing document data from API response.
- **include_raw_data** (bool) – Ignored for this model.

Returns

An instance of InterferenceDocumentData.

Return type

InterferenceDocumentData

interference_outcome_category: str | None = None

issue_type_bag: list[str]

statute_and_rule_bag: list[str]

to_dict()

Convert the InterferenceDocumentData instance to a dictionary.

Returns

Dictionary with camelCase keys and None values filtered.

Return type

Dict[str, Any]

class pyUSPTO.models.ptab.**InterferenceMetaData**(*interference_style_name=None*, *interference_last_modified_date=None*, *interference_last_modified_date_time=None*, *declaration_date=None*, *file_download_uri=None*)

Bases: object

Interference metadata.

interference_style_name

Style name of the interference.

interference_last_modified_date

Last modification date.

interference_last_modified_date_time

Last modification datetime.

declaration_date

Declaration date.

file_download_uri

URI to download ZIP of interference documents.

declaration_date: date | None = None

file_download_uri: str | None = None

classmethod from_dict(*data*, *include_raw_data=False*)

Create an InterferenceMetaData instance from a dictionary.

Parameters

- **data** (dict[str, Any]) – Dictionary containing interference metadata from API response.
- **include_raw_data** (bool) – Ignored for this model.

Returns

An instance of InterferenceMetaData.

Return type

InterferenceMetaData

interference_last_modified_date: date | None = None

interference_last_modified_date_time: datetime | None = None

interference_style_name: str | None = None

to_dict()

Convert the InterferenceMetaData instance to a dictionary.

Returns

Dictionary with camelCase keys and None values filtered.

Return type

Dict[str, Any]

class pyUSPTO.models.ptab.**JuniorPartyData**(*application_number_text=None*, *counsel_name=None*, *grant_date=None*, *group_art_unit_number=None*, *inventor_name=None*, *real_party_in_interest_name=None*, *patent_number=None*, *patent_owner_name=None*, *technology_center_number=None*, *publication_date=None*, *publication_number=None*)

Bases: PartyData

Junior party information in PTAB interference proceedings.

Inherits all attributes from PartyData. Represents the party with the later effective filing date in an interference.

class pyUSPTO.models.ptab.**PTABAppealDecision**(*appeal_number=None*, *last_modified_date_time=None*, *appeal_document_category=None*, *appeal_meta_data=None*, *appellant_data=None*, *requestor_data=None*, *document_data=None*, *decision_data=None*, *raw_data=None*)

Bases: object

Individual PTAB appeal decision record.

appeal_number

Appeal number.

last_modified_date_time
Last modification timestamp.

appeal_document_category
Document category.

appeal_meta_data
Appeal metadata.

appellant_data
Appellant information.

requestor_data
Third party requestor information.

document_data
Document metadata.

decision_data
Decision information.

raw_data
Raw JSON response data (if include_raw_data=True).

appeal_document_category: `str | None = None`

appeal_meta_data: *AppealMetaData* `| None = None`

appeal_number: `str | None = None`

appellant_data: *AppellantData* `| None = None`

decision_data: *DecisionData* `| None = None`

document_data: *AppealDocumentData* `| None = None`

classmethod from_dict(*data*, *include_raw_data=False*)
Create a PTABAppealDecision instance from a dictionary.

Parameters

- **data** (`dict[str, Any]`) – Dictionary containing appeal decision data from API response.
- **include_raw_data** (`bool`) – Whether to include raw JSON data in the instance.

Returns
An instance of PTABAppealDecision.

Return type
PTABAppealDecision

last_modified_date_time: `datetime | None = None`

raw_data: `dict[str, Any] | None = None`

requestor_data: *RequestorData* `| None = None`

to_dict()

Convert the PTABAppealDecision instance to a dictionary.

Returns
Dictionary with camelCase keys and None values filtered.

Return type
Dict[str, Any]

class pyUSPTO.models.ptab.**PTABAppealResponse**(*count=0*, *request_identifier=None*, *patent_appeal_data_bag=<factory>*, *raw_data=None*)

Bases: object

Response container for PTAB appeals search.

count

Total number of matching results.

request_identifier

UUID for the API request.

patent_appeal_data_bag

List of appeal decisions.

raw_data

Raw JSON response data (if include_raw_data=True).

count: int = 0

classmethod **from_dict**(*data*, *include_raw_data=False*)

Create a PTABAppealResponse instance from a dictionary.

Parameters

- **data** (dict[str, Any]) – Dictionary containing response data from API.
- **include_raw_data** (bool) – Whether to include raw JSON data in the instance.

Returns
An instance of PTABAppealResponse.

Return type
PTABAppealResponse

patent_appeal_data_bag: list[*PTABAppealDecision*]

raw_data: dict[str, Any] | None = None

request_identifier: str | None = None

to_dict()

Convert the PTABAppealResponse instance to a dictionary.

Returns
Dictionary with camelCase keys and None values filtered.

Return type
Dict[str, Any]

class pyUSPTO.models.ptab.**PTABInterferenceDecision**(*interference_number=None*, *last_modified_date_time=None*, *interference_meta_data=None*, *senior_party_data=None*, *junior_party_data=None*, *additional_party_data_bag=<factory>*, *document_data=None*, *raw_data=None*)

Bases: object

Individual PTAB interference decision record.

interference_number
Interference number.

last_modified_date_time
Last modification timestamp.

interference_meta_data
Interference metadata.

senior_party_data
Senior party information.

junior_party_data
Junior party information.

additional_party_data_bag
List of additional parties.

document_data
Document metadata.

raw_data
Raw JSON response data (if include_raw_data=True).

additional_party_data_bag: list[*AdditionalPartyData*]

document_data: *InterferenceDocumentData* | None = None

classmethod **from_dict**(*data*, *include_raw_data=False*)
Create a PTABInterferenceDecision instance from a dictionary.

Parameters

- **data** (dict[str, Any]) – Dictionary containing interference decision data from API response.
- **include_raw_data** (bool) – Whether to include raw JSON data in the instance.

Returns
An instance of PTABInterferenceDecision.

Return type
PTABInterferenceDecision

interference_meta_data: *InterferenceMetaData* | None = None

interference_number: str | None = None

junior_party_data: *JuniorPartyData* | None = None

last_modified_date_time: *datetime* | None = None

raw_data: dict[str, Any] | None = None

senior_party_data: *SeniorPartyData* | None = None

to_dict()

Convert the PTABInterferenceDecision instance to a dictionary.

Returns
Dictionary with camelCase keys and None values filtered.

Return type
Dict[str, Any]

class pyUSPTO.models.ptab.**PTABInterferenceResponse**(*count=0*, *request_identifier=None*, *patent_interference_data_bag=<factory>*, *raw_data=None*)

Bases: object

Response container for PTAB interferences search.

count

Total number of matching results.

request_identifier

UUID for the API request.

patent_interference_data_bag

List of interference decisions.

raw_data

Raw JSON response data (if include_raw_data=True).

count: int = 0

classmethod from_dict(*data*, *include_raw_data=False*)

Create a PTABInterferenceResponse instance from a dictionary.

Parameters

- **data** (dict[str, Any]) – Dictionary containing response data from API.
- **include_raw_data** (bool) – Whether to include raw JSON data in the instance.

Returns
An instance of PTABInterferenceResponse.

Return type
PTABInterferenceResponse

patent_interference_data_bag: list[*PTABInterferenceDecision*]

raw_data: dict[str, Any] | None = None

request_identifier: str | None = None

to_dict()

Convert the PTABInterferenceResponse instance to a dictionary.

Returns
Dictionary with camelCase keys and None values filtered.

Return type
Dict[str, Any]

class pyUSPTO.models.ptab.**PTABTrialDocument**(*trial_document_category=None*, *last_modified_date_time=None*, *trial_number=None*, *trial_type_code=None*, *trial_meta_data=None*, *patent_owner_data=None*, *regular_petitioner_data=None*, *respondent_data=None*, *derivation_petitioner_data=None*, *document_data=None*, *decision_data=None*, *raw_data=None*)

Bases: object

Individual trial document or decision record from PTAB document/decision search APIs.

Used by search_documents() and search_decisions() endpoints. Contains document-specific metadata (documentData) or decision information (decisionData), plus trial context. Differs from PTABTrialProceeding which represents the entire proceeding rather than individual documents within it.

trial_document_category
Category (Document or Decision).

last_modified_date_time
Last modification timestamp.

trial_number
Trial number (e.g., "IPR2023-00123").

trial_type_code
Type of trial (IPR, PGR, CBM, DER).

trial_meta_data
Trial metadata.

patent_owner_data
Patent owner information.

regular_petitioner_data
Regular petitioner information.

respondent_data
Respondent information.

derivation_petitioner_data
Derivation petitioner information.

document_data
Document metadata (if document).

decision_data
Decision information (if decision).

raw_data
Raw JSON response data (if include_raw_data=True).

decision_data: *TrialDecisionData* | None = None

derivation_petitioner_data: *DerivationPetitionerData* | None = None

document_data: *TrialDocumentData* | None = None

classmethod from_dict(*data*, *include_raw_data=False*)

Create a PTABTrialDocument instance from a dictionary.

Parameters

- **data** (dict[str, Any]) – Dictionary with API response data containing PTAB trial document information.
- **include_raw_data** (bool) – If True, includes the raw API response data in the instance.

Returns

A new instance populated with data from the dictionary.

Return type

PTABTrialDocument

last_modified_date_time: datetime | None = None

patent_owner_data: *PatentOwnerData* | None = None

raw_data: dict[str, Any] | None = None

regular_petitioner_data: *RegularPetitionerData* | None = None

respondent_data: *RespondentData* | None = None

to_dict()

Convert the PTABTrialDocument instance to a dictionary.

Returns

Dictionary with camelCase keys and None values filtered.

Return type

Dict[str, Any]

trial_document_category: str | None = None

trial_meta_data: *TrialMetaData* | None = None

trial_number: str | None = None

trial_type_code: str | None = None

class pyUSPTO.models.ptab.**PTABTrialDocumentResponse**(*count=0*, *request_identifier=None*, *patent_trial_document_data_bag=<factory>*, *raw_data=None*)

Bases: object

Response container for PTAB trial documents/decisions search.

count: int = 0

classmethod from_dict(*data*, *include_raw_data=False*)

Create a PTABTrialDocumentResponse instance from a dictionary.

Parameters

- **data** (dict[str, Any]) – Dictionary with API response data containing PTAB trial document response information.
- **include_raw_data** (bool) – If True, includes the raw API response data in the instance.

Returns
A new instance populated with data from the dictionary.

Return type
PTABTrialDocumentResponse

patent_trial_document_data_bag: list[*PTABTrialDocument*]

raw_data: dict[str, Any] | None = None

request_identifier: str | None = None

to_dict()

Convert the PTABTrialDocumentResponse instance to a dictionary.

Returns
Dictionary with camelCase keys and None values filtered.

Return type
Dict[str, Any]

class pyUSPTO.models.ptab.**PTABTrialProceeding**(*trial_number=None*, *last_modified_date_time=None*, *trial_meta_data=None*, *patent_owner_data=None*, *regular_petitioner_data=None*, *respondent_data=None*, *derivation_petitioner_data=None*, *raw_data=None*)

Bases: object

Individual PTAB trial proceeding record.

trial_number
Trial number (e.g., "IPR2023-00123").

trial_record_identifier
UUID identifier for the trial record.

last_modified_date_time
Last modification timestamp.

trial_meta_data
Trial metadata.

patent_owner_data
Patent owner information.

regular_petitioner_data
Regular petitioner information.

respondent_data
Respondent information.

derivation_petitioner_data
Derivation petitioner information.

raw_data
Raw JSON response data (if include_raw_data=True).

derivation_petitioner_data: *DerivationPetitionerData* | None = None

classmethod from_dict(*data*, *include_raw_data=False*)

Create a PTABTrialProceeding instance from a dictionary.

Parameters

- **data** (dict[str, Any]) – Dictionary containing trial proceeding data from API response.
- **include_raw_data** (bool) – Whether to include raw JSON data in the instance.

Returns
An instance of PTABTrialProceeding.

Return type
PTABTrialProceeding

last_modified_date_time: datetime | None = None

patent_owner_data: *PatentOwnerData* | None = None

raw_data: dict[str, Any] | None = None

regular_petitioner_data: *RegularPetitionerData* | None = None

respondent_data: *RespondentData* | None = None

to_dict()

Convert the PTABTrialProceeding instance to a dictionary.

Returns
Dictionary with camelCase keys and None values filtered.

Return type
Dict[str, Any]

trial_meta_data: *TrialMetaData* | None = None

trial_number: str | None = None

class pyUSPTO.models.ptab.**PTABTrialProceedingResponse**(*count=0*, *request_identifier=None*, *patent_trial_proceeding_data_bag=<factory>*, *raw_data=None*)

Bases: object

Response container for PTAB trial proceedings search.

count

Total number of matching results.

request_identifier

UUID for the API request.

patent_trial_proceeding_data_bag

List of trial proceedings.

raw_data

Raw JSON response data (if include_raw_data=True).

count: int = 0

classmethod from_dict(*data*, *include_raw_data=False*)

Create a PTABTrialProceedingResponse instance from a dictionary.

Parameters

- **data** (dict[str, Any]) – Dictionary containing response data from API.
- **include_raw_data** (bool) – Whether to include raw JSON data in the instance.

Returns
An instance of PTABTrialProceedingResponse.

Return type
PTABTrialProceedingResponse

patent_trial_proceeding_data_bag: list[*PTABTrialProceeding*]

raw_data: dict[str, Any] | None = None

request_identifier: str | None = None

to_dict()

Convert the PTABTrialProceedingResponse instance to a dictionary.

Returns
Dictionary with camelCase keys and None values filtered.

Return type
Dict[str, Any]

class pyUSPTO.models.ptab.**PatentOwnerData**(*application_number_text=None*, *counsel_name=None*, *grant_date=None*, *group_art_unit_number=None*, *inventor_name=None*, *real_party_in_interest_name=None*, *patent_number=None*, *patent_owner_name=None*, *technology_center_number=None*, *publication_date=None*, *publication_number=None*)

Bases: PartyData

Party data for a patent owner in PTAB trial proceedings.

Inherits all attributes from PartyData. Used in IPR, PGR, CBM, and DER proceedings to represent the patent holder.

class pyUSPTO.models.ptab.**RegularPetitionerData**(*counsel_name=None*, *real_party_in_interest_name=None*)

Bases: object

Regular petitioner information.

counsel_name

Name of counsel.

real_party_in_interest_name

Real party in interest name.

counsel_name: str | None = None

classmethod from_dict(*data*, *include_raw_data=False*)

Create a RegularPetitionerData instance from a dictionary.

Parameters

- **data** (dict[str, Any]) – Dictionary containing petitioner data from API response.
- **include_raw_data** (bool) – Ignored for this model.

Returns
An instance of RegularPetitionerData.

Return type
RegularPetitionerData

real_party_in_interest_name: str | None = None

to_dict()
Convert the RegularPetitionerData instance to a dictionary.

Returns
Dictionary with camelCase keys and None values filtered.

Return type
Dict[str, Any]

class pyUSPTO.models.ptab.**RequestorData**(*third_party_name=None*)
Bases: object

Third party requestor information.

third_party_name
Name of the third party.

classmethod from_dict(*data*, *include_raw_data=False*)
Create a RequestorData instance from a dictionary.

Parameters
- **data** (dict[str, Any]) – Dictionary containing requestor data from API response.
- **include_raw_data** (bool) – Ignored for this model.

Returns
An instance of RequestorData.

Return type
RequestorData

third_party_name: str | None = None

to_dict()
Convert the RequestorData instance to a dictionary.

Returns
Dictionary with camelCase keys and None values filtered.

Return type
Dict[str, Any]

class pyUSPTO.models.ptab.**RespondentData**(*application_number_text=None*, *counsel_name=None*, *grant_date=None*, *group_art_unit_number=None*, *inventor_name=None*, *real_party_in_interest_name=None*, *patent_number=None*, *patent_owner_name=None*, *technology_center_number=None*, *publication_date=None*, *publication_number=None*)

Bases: PartyData

Respondent party data in derivation proceedings.

Inherits all attributes from PartyData. Used in DER proceedings to represent the responding party.

class pyUSPTO.models.ptab.**SeniorPartyData**(*application_number_text=None*, *counsel_name=None*, *grant_date=None*, *group_art_unit_number=None*, *inventor_name=None*, *real_party_in_interest_name=None*, *patent_number=None*, *patent_owner_name=None*, *technology_center_number=None*, *publication_date=None*, *publication_number=None*)

Bases: PartyData

Senior party information in PTAB interference proceedings.

Inherits all attributes from PartyData. Represents the party with the earlier effective filing date in an interference.

class pyUSPTO.models.ptab.**TrialDecisionData**(*statute_and_rule_bag=<factory>*, *decision_issue_date=None*, *decision_type_category=None*, *issue_type_bag=<factory>*, *trial_outcome_category=None*)

Bases: object

Metadata for a decision in a PTAB trial.

statute_and_rule_bag

List of applicable statutes and rules.

decision_issue_date

Date issued.

decision_type_category

Type of decision (e.g. "Final Written Decision").

issue_type_bag

List of issues addressed.

trial_outcome_category

Outcome (e.g., "Denied").

decision_issue_date: date | None = None

decision_type_category: str | None = None

classmethod from_dict(*data*)

Create a TrialDecisionData instance from a dictionary.

Parameters
data (dict[str, Any]) – Dictionary with API response data containing trial decision information.

Returns
A new instance populated with data from the dictionary.

Return type
TrialDecisionData

issue_type_bag: list[str]

statute_and_rule_bag: list[str]

to_dict()

Convert the TrialDecisionData instance to a dictionary.

Returns
Dictionary with camelCase keys and None values filtered.

Return type
Dict[str, Any]

trial_outcome_category: str | None = None

class pyUSPTO.models.ptab.**TrialDocumentData**(*document_category=None*, *document_filing_date=None*, *document_identifier=None*, *document_name=None*, *document_number=None*, *document_size_quantity=None*, *document_ocr_text=None*, *document_title_text=None*, *document_type_description_text=None*, *file_download_uri=None*, *filing_party_category=None*)

Bases: object

Metadata for a document in a PTAB trial.

document_category
Category of the document.

document_filing_date
Filing date.

document_identifier
Unique ID.

document_name
Filename.

document_number
Document number in the proceeding.

document_size_quantity
Size in bytes.

document_ocr_text
OCR text content.

document_title_text
Title of the document.

document_type_description_text
Description of document type.

file_download_uri
URL to download the file.

filing_party_category
Who filed (e.g., "Petitioner").

mime_type_identifier
MIME type (e.g., "application/pdf").

document_status
Public status.

document_category: str | None = None

document_filing_date: date | None = None

document_identifier: str | None = None

document_name: str | None = None

document_number: str | None = None

document_ocr_text: str | None = None

document_size_quantity: int | None = None

document_title_text: str | None = None

document_type_description_text: str | None = None

file_download_uri: str | None = None

filing_party_category: str | None = None

classmethod from_dict(*data*, *include_raw_data=False*)
Create a TrialDocumentData instance from a dictionary.

Parameters
- **data** (dict[str, Any]) – Dictionary containing document data from API response.
- **include_raw_data** (bool) – Ignored for this model.

Returns
An instance of TrialDocumentData.

Return type
TrialDocumentData

to_dict()
Convert the TrialDocumentData instance to a dictionary.

Returns
Dictionary with camelCase keys and None values filtered.

Return type
Dict[str, Any]

class pyUSPTO.models.ptab.**TrialMetaData**(*petition_filing_date=None*, *accorded_filing_date=None*, *trial_last_modified_date_time=None*, *trial_last_modified_date=None*, *trial_status_category=None*, *trial_type_code=None*, *file_download_uri=None*, *termination_date=None*, *latest_decision_date=None*, *institution_decision_date=None*)

Bases: object

Trial metadata including status, dates, and download URI.

petition_filing_date
Date the petition was filed.

accorded_filing_date
The filing date accorded to the petition.

trial_last_modified_date_time
Last modification timestamp.

trial_last_modified_date
Last modification date.

trial_status_category
Status of the trial (e.g., "Institution Denied", "Instituted").

trial_type_code
Type of trial (IPR, PGR, CBM, DER).

file_download_uri
URI to download ZIP of all trial documents.

termination_date
Date the trial was terminated.

latest_decision_date
Date of the most recent decision.

institution_decision_date
Date of the institution decision.

accorded_filing_date: `date | None = None`

file_download_uri: `str | None = None`

classmethod from_dict(*data*, *include_raw_data=False*)
Create a TrialMetaData instance from a dictionary.

Parameters

- **data** (`dict[str, Any]`) – Dictionary containing trial metadata from API response.
- **include_raw_data** (`bool`) – Ignored for this model (no raw_data field).

Returns
An instance of TrialMetaData.

Return type
TrialMetaData

institution_decision_date: `date | None = None`

latest_decision_date: `date | None = None`

petition_filing_date: `date | None = None`

termination_date: `date | None = None`

to_dict()
Convert the TrialMetaData instance to a dictionary.

Returns
Dictionary with camelCase keys and None values filtered.

Return type
Dict[str, Any]

trial_last_modified_date: date | None = None

trial_last_modified_date_time: datetime | None = None

trial_status_category: str | None = None

trial_type_code: str | None = None

models.utils - Utility functions for USPTO data models.

This module provides utility functions for parsing, serializing, and converting data used across USPTO API data models. These utilities handle date/datetime conversions, boolean string representations, and string transformations.

pyUSPTO.models.utils.**parse_to_date**(*date_str*, *fmt='%Y-%m-%d'*)

Parse a string representation of a date into a date object.

Parameters

- **date_str** (Optional[str]) – The string to parse as a date.
- **fmt** (str, optional) – The expected strptime format string for parsing the date. Defaults to "%Y-%m-%d".

Returns

A date object if parsing is successful and *date_str*
is not None. Returns None if *date_str* is None or if parsing fails.

Return type
Optional[date]

Warns
USPTODateParseWarning – If the date string cannot be parsed.

pyUSPTO.models.utils.**parse_to_datetime_utc**(*datetime_str*)

Parse a string representation of a datetime into a UTC datetime object.

Attempts to parse ISO format strings. If the input string contains timezone information, it's used. If the string is a naive datetime (no timezone), it's assumed to be in the *ASSUMED_NAIVE_TIMEZONE_STR* (e.g., "America/New_York") and then converted to UTC.

Parameters
datetime_str (Optional[str]) – The string to parse as a datetime. Supports ISO 8601 format, including those ending with "Z".

Returns

A timezone-aware datetime object in UTC if parsing
is successful and *datetime_str* is not None. Returns None if *datetime_str* is None or if parsing/conversion fails.

Return type
Optional[datetime]

Warns

- **USPTODateParseWarning** – If the datetime string cannot be parsed.
- **USPTOTimezoneWarning** – If timezone localization fails.

pyUSPTO.models.utils.**parse_yn_to_bool**(*value*)

Convert a 'Y'/'N' (case-insensitive) string to a boolean.

Parameters
value (Optional[str]) – The string value to convert. Expected to be 'Y', 'y', 'N', or 'n'.

Returns

True if *value* is 'Y' or 'y', False if *value* is
'N' or 'n'. Returns None if *value* is None or any other string.

Return type
Optional[bool]

Warns
USPTOBooleanParseWarning – If the value is not 'Y' or 'N'.

pyUSPTO.models.utils.**serialize_bool_to_yn**(*value*)

Convert a boolean value to its 'Y'/'N' string representation.

Parameters
value (Optional[bool]) – The boolean value to convert.

Returns

"Y" if *value* is True, "N" if *value* is False.
Returns None if *value* is None.

Return type
Optional[str]

pyUSPTO.models.utils.**serialize_date**(*d*)

Serialize a date object into an ISO 8601 string (YYYY-MM-DD).

Parameters
d (Optional[date]) – The date object to serialize.

Returns

The date as an ISO 8601 formatted string, or None
if the input is None.

Return type
Optional[str]

pyUSPTO.models.utils.**serialize_datetime_as_iso**(*dt*)

Serialize a datetime object to a local-timezone ISO 8601 string.

If the input datetime object is timezone-aware, it is converted to the assumed local timezone defined by *ASSUMED_NAIVE_TIMEZONE*. If it is naive (lacks timezone information), it is first assigned that assumed local timezone.

The resulting datetime is formatted as:
YYYY-MM-DDTHH:MM:SS.000±HHMM

(e.g., "2024-12-10T00:00:00.000-0500")

Parameters
dt (Optional[datetime]) – The datetime object to serialize. Can be naive or timezone-aware.

Returns

The datetime formatted in the assumed local timezone,
or None if the input *dt* is None.

Return type
Optional[str]

pyUSPTO.models.utils.**serialize_datetime_as_naive**(*dt*)

Serialize a datetime object to ISO format as a naive datetime.

Converts aware datetimes to the assumed timezone and strips timezone information before serializing to ISO format.

Parameters
dt (datetime) – The datetime object to serialize.

Returns
ISO formatted datetime string without timezone information.

Return type
str

pyUSPTO.models.utils.**to_camel_case**(*snake_str*)

Convert a snake_case string to lowerCamelCase.

For example, "example_snake_string" becomes "exampleSnakeString".

Parameters
snake_str (str) – The input string in snake_case.

Returns
The converted string in lowerCamelCase.

Return type
str

3.3 Configuration

config - Configuration management for USPTO API clients.

This module provides configuration management for USPTO API clients, including API keys, base URLs, and HTTP transport settings.

class pyUSPTO.config.**USPTOConfig**(*api_key=None*, *bulk_data_base_url='https://api.uspto.gov'*, *patent_data_base_url='https://api.uspto.gov'*, *petition_decisions_base_url='https://api.uspto.gov'*, *ptab_base_url='https://api.uspto.gov'*, *http_config=None*, *include_raw_data=False*)

Bases: object

Configuration for USPTO API clients.

Manages API-level configuration (keys, URLs) and optionally accepts HTTP transport configuration via HTTPConfig.

__init__(*api_key=None*, *bulk_data_base_url='https://api.uspto.gov'*, *patent_data_base_url='https://api.uspto.gov'*, *petition_decisions_base_url='https://api.uspto.gov'*, *ptab_base_url='https://api.uspto.gov'*, *http_config=None*, *include_raw_data=False*)

Initialize the USPTOConfig.

Parameters

- **api_key** (Optional[str]) – API key for authentication, defaults to USPTO_API_KEY environment variable
- **bulk_data_base_url** (str) – Base URL for the Bulk Data API
- **patent_data_base_url** (str) – Base URL for the Patent Data API

- **petition_decisions_base_url** (str) – Base URL for the Final Petition Decisions API
- **ptab_base_url** (str) – Base URL for the PTAB (Patent Trial and Appeal Board) API
- **http_config** (Optional[*HTTPConfig*]) – Optional HTTPConfig for request handling (uses defaults if None)
- **include_raw_data** (bool) – If True, store raw JSON in response objects for debugging (default: False)

classmethod from_env()

Create a USPTOConfig from environment variables.

Return type
USPTOConfig

Returns
USPTOConfig instance with values from environment

http_config - HTTP client configuration for USPTO API requests.

This module provides configuration for HTTP transport-level settings including timeouts, retries, connection pooling, and custom headers.

class pyUSPTO.http_config.**HTTPConfig**(*timeout=30.0*, *connect_timeout=10.0*, *max_retries=3*, *backoff_factor=2.0*, *retry_status_codes=<factory>*, *pool_connections=10*, *pool_maxsize=10*, *download_chunk_size=8192*, *custom_headers=None*)

Bases: object

HTTP client configuration for request handling.

This class separates transport-level HTTP concerns from API-level configuration, allowing fine-grained control over request behavior.

timeout
Read timeout in seconds for requests (default: 30.0)

connect_timeout
Connection establishment timeout in seconds (default: 10.0)

max_retries
Maximum number of retry attempts (default: 3)

backoff_factor
Exponential backoff multiplier for retries (default: 1.0)

retry_status_codes
HTTP status codes that trigger retries

pool_connections
Number of connection pools to cache (default: 10)

pool_maxsize
Maximum number of connections per pool (default: 10)

download_chunk_size
Chunk size in bytes for streaming file downloads (default: 8192)

custom_headers
Additional headers to include in all requests

__post_init__()

Validate configuration after initialization.

Return type
None

backoff_factor: float = 2.0

connect_timeout: float | None = 10.0

custom_headers: dict[str, str] | None = None

download_chunk_size: int = 8192

classmethod from_env()

Create HTTPConfig from environment variables.

Environment variables:
USPTO_REQUEST_TIMEOUT: Request timeout in seconds USPTO_CONNECT_TIMEOUT: Connection timeout in seconds USPTO_MAX_RETRIES: Maximum retry attempts USPTO_BACKOFF_FACTOR: Retry backoff factor USPTO_POOL_CONNECTIONS: Connection pool size USPTO_POOL_MAXSIZE: Max connections per pool USPTO_DOWNLOAD_CHUNK_SIZE: Chunk size for streaming downloads (bytes)

Return type
HTTPConfig

Returns
HTTPConfig instance with values from environment or defaults

get_timeout_tuple()

Get timeout as tuple for requests library.

Return type
tuple[float | None, float | None]

Returns
Tuple of (connect_timeout, read_timeout) for requests

max_retries: int = 3

pool_connections: int = 10

pool_maxsize: int = 10

retry_status_codes: list[int]

timeout: float | None = 30.0

3.4 Exceptions

exceptions - Exception classes for USPTO API clients.

This module provides exception classes for USPTO API errors that correspond to the various response types from the USPTO API. It also includes helper structures and functions for creating these exceptions.

class pyUSPTO.exceptions.**APIErrorArgs**(*message*, *status_code=None*, *api_short_error=None*, *error_details=None*, *request_identifier=None*)

Bases: object

Data structure to hold arguments for API exception constructors.

api_short_error: str | None = None

error_details: str | dict | None = None

classmethod **from_http_error**(*http_error*, *client_operation_message*)

Create an APIErrorArgs instance by parsing a requests.exceptions.HTTPError.

Parameters

- **http_error** (HTTPError) – The HTTPError object from the requests library.
- **client_operation_message** (str) – A message describing the client operation that failed.

Return type
APIErrorArgs

Returns
An instance of APIErrorArgs populated with details from the HTTPError.

classmethod **from_request_exception**(*request_exception*, *client_operation_message=None*)

Create an APIErrorArgs instance.

Create an APIErrorArgs instance from a generic requests.exceptions.RequestException. (e.g., ConnectionError, Timeout) that is not an HTTPError.

Return type
APIErrorArgs

message: str

request_identifier: str | None = None

status_code: int | None = None

exception pyUSPTO.exceptions.**FormatNotAvailableError**(*requested_format*, *available_formats*, *document=None*)

Bases: ValueError

Raised when a requested document format is not available.

This exception is raised when attempting to download a document in a format that is not available for that specific document. It provides programmatic access to the requested format and available alternatives.

requested_format

The format that was requested (e.g., "XML", "PDF")

available_formats

List of available format identifiers

document

Optional Document object for additional context

__init__(*requested_format*, *available_formats*, *document=None*)

Initialize FormatNotAvailableError.

Parameters

- **requested_format** (str) – The format that was requested
- **available_formats** (list[str]) – List of available format identifiers
- **document** (Optional[*Document*]) – Optional Document object for additional context

exception pyUSPTO.exceptions.**USPTOApiAuthError**(*message*, *status_code=None*, *api_short_error=None*, *error_details=None*, *request_identifier=None*)

Bases: *USPTOApiError*

Authentication/Authorization error (HTTP 401/403).

exception pyUSPTO.exceptions.**USPTOApiBadRequestError**(*message*, *status_code=None*, *api_short_error=None*, *error_details=None*, *request_identifier=None*)

Bases: *USPTOApiError*

Bad Request error (HTTP 400).

exception pyUSPTO.exceptions.**USPTOApiError**(*message*, *status_code=None*, *api_short_error=None*, *error_details=None*, *request_identifier=None*)

Bases: Exception

Base exception for USPTO API errors.

This is the parent class for all USPTO API-specific exceptions. It includes information about the status code, API's short error message, detailed error information, and request identifier from the API response.

DEFAULT_UNKNOWN_MESSAGE = 'UNK USPTO API ERROR'

__init__(*message*, *status_code=None*, *api_short_error=None*, *error_details=None*, *request_identifier=None*)

Initialize a USPTOApiError.

Parameters

- **message** (str) – The primary message for the exception (often client-generated context).
- **status_code** (Optional[int]) – The HTTP status code from the API response (e.g., 400, 403).
- **api_short_error** (Optional[str]) – The short error description from the API (e.g., "Bad Request", "Forbidden").
- **error_details** (Union[str, dict, None]) – The detailed error message or structure from the API.
- **request_identifier** (Optional[str]) – The request identifier from the API response, if available.

__str__()

Provide a more informative string representation of the error.

Return type
str

property message: str

Provides direct access to the primary exception message.

This refers to the first argument passed to the exception, which is conventionally the main human-readable message.

exception pyUSPTO.exceptions.**USPTOApiNotFoundError**(*message*, *status_code=None*, *api_short_error=None*, *error_details=None*, *request_identifier=None*)

Bases: *USPTOApiError*

Resource not found error (HTTP 404).

exception pyUSPTO.exceptions.**USPTOApiPayloadTooLargeError**(*message*, *status_code=None*, *api_short_error=None*, *error_details=None*, *request_identifier=None*)

Bases: *USPTOApiError*

Payload Too Large error (HTTP 413).

exception pyUSPTO.exceptions.**USPTOApiRateLimitError**(*message*, *status_code=None*, *api_short_error=None*, *error_details=None*, *request_identifier=None*)

Bases: *USPTOApiError*

Rate limit exceeded error (HTTP 429).

exception pyUSPTO.exceptions.**USPTOApiServerError**(*message*, *status_code=None*, *api_short_error=None*, *error_details=None*, *request_identifier=None*)

Bases: *USPTOApiError*

Internal Server Error (HTTP 500 series).

exception pyUSPTO.exceptions.**USPTOConnectionError**(*message*, *status_code=None*, *api_short_error=None*, *error_details=None*, *request_identifier=None*)

Bases: *USPTOApiError*

Network-level connection error (DNS failure, refused connection, etc.).

exception pyUSPTO.exceptions.**USPTOTimeout**(*message*, *status_code=None*, *api_short_error=None*, *error_details=None*, *request_identifier=None*)

Bases: *USPTOApiError*

Request to USPTO API timed out.

pyUSPTO.exceptions.**get_api_exception**(*error_args*)

Determine and instantiate the appropriate USPTOApiError subclass.

Based on the status code in error_args.

Parameters

error_args (*APIErrorArgs*) – An instance of APIErrorArgs containing all necessary information to construct the exception.

Return type

USPTOApiError

Returns

An instance of a USPTOApiError subclass.

3.5 Warnings

warnings - Warning classes for pyUSPTO data parsing issues.

This module defines custom warning categories for different types of data parsing issues encountered when working with USPTO API responses. These warnings follow Python's standard warning framework and can be controlled using warnings.filterwarnings().

Example

Suppress all pyUSPTO data warnings import warnings from pyUSPTO.warnings import USPTODataWarning warnings.filterwarnings('ignore', category=USPTODataWarning)

Turn specific warnings into errors (strict mode) warnings.filterwarnings('error', category=USPTODateParseWarning)

exception pyUSPTO.warnings.**USPTOBooleanParseWarning**

Bases: *USPTODataWarning*

Warning for Y/N boolean string parsing failures.

Raised when a string that should be 'Y' or 'N' has an unexpected value. The field will be set to None.

exception pyUSPTO.warnings.**USPTODataMismatchWarning**

Bases: *USPTODataWarning*

Warning for data validation mismatches.

Raised when the API returns data that doesn't match the requested identifier (e.g., requesting application 12345678 but receiving 87654321). This indicates a potential API inconsistency or data integrity issue.

exception pyUSPTO.warnings.**USPTODataWarning**

Bases: UserWarning

Base warning class for USPTO data parsing issues.

All pyUSPTO data-related warnings inherit from this class, allowing users to filter all data warnings at once.

exception pyUSPTO.warnings.**USPTODateParseWarning**

Bases: *USPTODataWarning*

Warning for date/datetime string parsing failures.

Raised when a date or datetime string from the API cannot be parsed into a Python date/datetime object. The field will be set to None.

exception pyUSPTO.warnings.**USPTOEnumParseWarning**

Bases: *USPTODataWarning*

Warning for enum value parsing failures.

Raised when an API response contains a value that doesn't match any defined enum member. The field will be set to None.

exception pyUSPTO.warnings.**USPTOTimezoneWarning**

Bases: *USPTODataWarning*

Warning for timezone-related issues.

Raised when timezone data is not available or timezone conversion fails. Falls back to UTC timezone.

CHAPTER

FOUR

EXAMPLES

4.1 Bulk Data Examples

```python
"""Example usage of the uspto_api module for bulk data.

This example demonstrates how to use the BulkDataClient to interact with the USPTO Bulk Data API.
It shows how to retrieve product information, search for products, and download files.
"""

import os

import requests

from pyUSPTO.clients import BulkDataClient  # Import from top-level package
from pyUSPTO.config import USPTOConfig


def format_size(size_bytes: int | float) -> str:
    """Format a size in bytes to a human-readable string (KB, MB, GB, etc.).

    Args:
        size_bytes: The size in bytes to format

    Returns:
        A human-readable string representation of the size
    """
    if size_bytes == 0:
        return "0 B"

    size_names = ["B", "KB", "MB", "GB", "TB", "PB"]
    i = 0
    while size_bytes >= 1024 and i < len(size_names) - 1:
        size_bytes /= 1024
        i += 1

    # Round to 2 decimal places
    return f"{size_bytes:.2f} {size_names[i]}"


# Method 1: Initialize the client with direct API key
```

(continues on next page)

(continued from previous page)

```python
# This approach is simple but less flexible
print("Method 1: Initialize with direct API key")
api_key = "YOUR_API_KEY_HERE"  # Replace with your actual API key
client = BulkDataClient(api_key=api_key)

# Method 2: Initialize the client with USPTOConfig
# This approach provides more configuration options
print("\nMethod 2: Initialize with USPTOConfig")
config = USPTOConfig(
    api_key="YOUR_API_KEY_HERE",  # Replace with your actual API key
    bulk_data_base_url="https://api.uspto.gov/api/v1/datasets",
    patent_data_base_url="https://api.uspto.gov/api/v1/patent",
)
client = BulkDataClient(config=config)

# Method 3: Initialize the client with environment variables
# This is the most secure approach for production use
print("\nMethod 3: Initialize with environment variables")
# Set environment variable (in a real scenario, this would be set outside the script)
os.environ["USPTO_API_KEY"] = "YOUR_API_KEY_HERE"  # Replace with your actual API key
config_from_env = USPTOConfig.from_env()
client = BulkDataClient(config=config_from_env)

print("\nBeginning API requests with configured client:")

# Get all available products
response = client.get_products()
print(f"Found {response.count} products")

# Display information about each product
for product in response.bulk_data_product_bag:
    print(f"\nProduct: {product.product_title_text}")
    print(f"ID: {product.product_identifier}")
    print(f"Description: {product.product_description_text}")
    print(f"Date range: {product.product_from_date} to {product.product_to_date}")
    print(f"Total files: {product.product_file_total_quantity}")
    print(f"Total size: {format_size(size_bytes=product.product_total_file_size)}")

    # Get detailed product info with files included
    try:
        detailed_product = client.get_product_by_id(
            product.product_identifier, include_files=True
        )
        if (
            detailed_product.product_file_bag
            and detailed_product.product_file_bag.file_data_bag
        ):
            print(f"\nFiles ({detailed_product.product_file_bag.count}):")
            for file_data in detailed_product.product_file_bag.file_data_bag:
                print(f"  - {file_data.file_name} ({format_size(file_data.file_size)})")
                print(f"    Type: {file_data.file_type_text}")
                print(f"    Released: {file_data.file_release_date}")
```

(continues on next page)

(continued from previous page)

```
                if file_data.file_download_uri:
                    print(f"    Download URI: {file_data.file_download_uri}")
        else:
            print("\nNo files available for this product")
    except Exception as e:
        print(f"\nError retrieving detailed product info: {e}")

# Search for products by date range
date_filtered = client.search_products(from_date="2025-01-01", to_date="2025-03-31")
print(f"\nFound {date_filtered.count} products in date range")

# Search for products by label
try:
    # Using labels we saw in the API response
    label_filtered = client.search_products(labels=["Patent"])
    print(f"\nFound {label_filtered.count} products with label 'Patent'")
except requests.exceptions.HTTPError as e:
    print(f"Error searching by labels: {e}")

# Get a specific product by ID
product_id = "PEDSJSON"  # Using a real product ID from the output
try:
    product = client.get_product_by_id(product_id, include_files=True)
    print(f"\nRetrieved product: {product.product_title_text}")

    # Download a file from this product
    if product.product_file_bag and product.product_file_bag.file_data_bag:
        file_to_download = product.product_file_bag.file_data_bag[0]
        print(f"File download URI: {file_to_download.file_download_uri}")
        downloaded_path = client.download_file(
            file_data=file_to_download, destination="./downloads"
        )
        print(f"Downloaded file to: {downloaded_path}")
        print(f"File size: {format_size(size_bytes=file_to_download.file_size)}")

except Exception as e:
    print(f"Error retrieving product {product_id}: {e}")
```

4.2 Patent Data Examples

```
"""Example usage of the uspto_api module for patent data.

This example demonstrates how to use the PatentDataClient to interact with the USPTO␣
↪Patent Data API.
It shows how to retrieve patent applications, search for patents by various criteria,␣
↪and access
detailed patent information including inventors, applicants, assignments, and more.
"""

import json
import os
```

(continues on next page)

(continued from previous page)

```python

from pyUSPTO.clients.patent_data import PatentDataClient
from pyUSPTO.models.patent_data import ApplicationContinuityData

# --- Initialization ---
# Initialize the client with API key from ENV Var.
print("Initialize with direct API key")
api_key = os.environ.get("USPTO_API_KEY", "YOUR_API_KEY_HERE")
if api_key == "YOUR_API_KEY_HERE":
    raise ValueError(
        "WARNING: API key is not set. Please replace 'YOUR_API_KEY_HERE' or set USPTO_API_KEY environment variable."
    )
client = PatentDataClient(api_key=api_key)

DEST_PATH = "./download-example"

print("\nBeginning API requests with configured client:")

# Get some patent applications (default is 25)
try:
    print("\nAttempting to get some patent applications (default search)...")
    # Calling with no specific query, relying on API defaults or client defaults (e.g., limit)
    response = client.search_applications(limit=5)  # Example: get 5 results
    print(
        f"Found {response.count} total patent applications matching default/broad criteria."
    )
    print(
        f"Displaying first {len(response.patent_file_wrapper_data_bag)} applications from response:"
    )

    for patent_wrapper in response.patent_file_wrapper_data_bag:
        app_meta = patent_wrapper.application_meta_data
        if app_meta:
            print(f"\n  Application: {patent_wrapper.application_number_text}")
            print(f"  Title: {app_meta.invention_title}")
            print(f"  Status: {app_meta.application_status_description_text}")
            print(f"  Filing Date: {app_meta.filing_date}")

            if app_meta.patent_number:
                print(f"  Patent Number: {app_meta.patent_number}")
                print(f"  Grant Date: {app_meta.grant_date}")

            if app_meta.inventor_bag:
                print("  Inventors:")
                for inventor in app_meta.inventor_bag:
                    name_parts = [
                        part
                        for part in [inventor.first_name, inventor.last_name]
```

(continues on next page)

(continued from previous page)

```
                        if part
                    ]
                    print(f"    - {' '.join(name_parts).strip()}")
                    if inventor.correspondence_address_bag:
                        address = inventor.correspondence_address_bag[0]
                        if address.city_name and address.geographic_region_code:
                            print(
                                f"      ({address.city_name}, {address.geographic_region_
↪code})"
                            )

            if app_meta.applicant_bag:
                print("  Applicants:")
                for applicant in app_meta.applicant_bag:
                    print(f"    - {applicant.applicant_name_text}")
        print("-" * 20)

    # Example of using the to_csv method from PatentDataResponse
    if response.count > 0:
        print("\nGenerating CSV for the current response (first few rows shown):")
        csv_data = response.to_csv()
        # You could save this csv_data to a file:
        # with open("patent_search_results.csv", "w", newline="", encoding="utf-8") as f:
        # f.write(csv_data)
        # print("\nFull CSV data saved to patent_search_results.csv (example).")


except Exception as e:
    print(f"Error getting patent applications: {e}")

# Search for patents by inventor name using convenience _q parameter
try:
    print("\nSearching for patents with 'Smith' as inventor...")
    # Changed from search_patents to search_applications with inventor_name_q
    inventor_search_response = client.search_applications(
        inventor_name_q="Smith", limit=2
    )
    print(
        f"Found {inventor_search_response.count} patents with 'Smith' as inventor␣
↪(showing up to 2)."
    )
    for patent_wrapper in inventor_search_response.patent_file_wrapper_data_bag:
        if patent_wrapper.application_meta_data:
            print(
                f"  - App No: {patent_wrapper.application_number_text}, Title: {patent_
↪wrapper.application_meta_data.invention_title}"
            )
except Exception as e:
    print(f"Error searching by inventor: {e}")


# Search for patents filed in a date range using convenience _q parameters
```

(continues on next page)

(continued from previous page)

```
try:
    print("\nSearching for patents filed in 2020...")
    date_search_response = client.search_applications(
        filing_date_from_q="2020-01-01", filing_date_to_q="2020-12-31", limit=2
    )
    print(
        f"Found {date_search_response.count} patents filed in 2020 (showing up to 2)."
    )
    for patent_wrapper in date_search_response.patent_file_wrapper_data_bag:
        if patent_wrapper.application_meta_data:
            print(
                f"  - App No: {patent_wrapper.application_number_text}, Filing Date:
↪{patent_wrapper.application_meta_data.filing_date}"
            )
except Exception as e:
    print(f"Error searching by date range: {e}")

# Get a specific patent by application number
app_no_to_fetch = "18045436"  # Known application number, ensure it's valid
try:
    print(f"\nAttempting to retrieve patent application: {app_no_to_fetch}")
    patent_wrapper_detail = client.get_application_by_number(
        application_number=app_no_to_fetch
    )
    if patent_wrapper_detail:
        print(
            f"Successfully retrieved: {patent_wrapper_detail.application_number_text}"
        )
        if patent_wrapper_detail.application_meta_data:
            print(
                f"Title: {patent_wrapper_detail.application_meta_data.invention_title}"
            )

        print("\nRetrieving document information...")
        documents_bag = client.get_application_documents(
            application_number=app_no_to_fetch
        )
        print(f"Found {len(documents_bag)} documents for application {app_no_to_fetch}")

        if documents_bag.documents:
            document_to_download = documents_bag.documents[0]  # Example: first document
            print("\nFirst document details:")
            print(f"  Document ID: {document_to_download.document_identifier}")
            print(
                f"  Document Type: {document_to_download.document_code} - {document_to_
↪download.document_code_description_text}"
            )
            print(f"  Date: {document_to_download.official_date}")
            print(f"  Direction: {document_to_download.direction_category}")

            if (
                document_to_download.document_formats
```

(continues on next page)

(continued from previous page)

```
                and document_to_download.document_identifier
            ):
                print("\nAttempting to download first PDF document...")
                print(json.dumps(document_to_download.to_dict(), indent=2))
                downloaded_path = client.download_document(
                    document=document_to_download,
                    format="PDF",
                    destination=DEST_PATH,
                    overwrite=True,
                )
                print(f"Downloaded document to: {downloaded_path}")
            else:
                print(
                    "No downloadable formats available for the first document or␣
↪document identifier missing."
                )
        else:
            print("No documents listed for this application.")

        # Example: Download publication XML (grant or pgpub)
        print("\nChecking for publication files (grant/pgpub XML)...")
        if patent_wrapper_detail.grant_document_meta_data:
            grant_metadata = patent_wrapper_detail.grant_document_meta_data
            print(f"Grant document available: {grant_metadata.xml_file_name}")
            print(f"  Product: {grant_metadata.product_identifier}")
            print(f"  Created: {grant_metadata.file_create_date_time}")

            # Download grant XML to downloads folder with auto-generated filename
            print("\nDownloading grant XML...")
            grant_path = client.download_publication(
                printed_metadata=grant_metadata,
                destination=DEST_PATH,
                overwrite=True,
            )
            print(f"Downloaded grant XML to: {grant_path}")

        if patent_wrapper_detail.pgpub_document_meta_data:
            pgpub_metadata = patent_wrapper_detail.pgpub_document_meta_data
            print(f"\nPre-grant publication available: {pgpub_metadata.xml_file_name}")

            # Download with custom filename
            pgpub_path = client.download_publication(
                printed_metadata=pgpub_metadata,
                file_name="my_pgpub.xml",
                destination=DEST_PATH,
                overwrite=True,
            )
            print(f"Downloaded pgpub XML to: {pgpub_path}")

        if patent_wrapper_detail.assignment_bag:
            print("\nAssignments:")
            for assignment in patent_wrapper_detail.assignment_bag:
```

(continues on next page)

(continued from previous page)

```
                for assignee in assignment.assignee_bag:
                    print(
                        f"  - {assignee.assignee_name_text} (Recorded: {assignment.assignment_recorded_date})"
                    )
                    print(f"    Conveyance: {assignment.conveyance_text}")
    else:
        print(f"Could not retrieve details for application {app_no_to_fetch}")

except Exception as e:
    print(f"Error retrieving or processing patent application {app_no_to_fetch}: {e}")

# Search for a specific patent by patent number (using search_applications)
target_patent_number = "10000000"
try:
    print(f"\nSearching for patent US {target_patent_number} B2...")
    patent_search_response = client.search_applications(
        patent_number_q=target_patent_number, limit=1
    )

    if (
        patent_search_response.count > 0
        and patent_search_response.patent_file_wrapper_data_bag
    ):
        found_patent_wrapper = patent_search_response.patent_file_wrapper_data_bag[0]
        if (
            found_patent_wrapper.application_meta_data
            and found_patent_wrapper.application_meta_data.patent_number
        ):
            print(
                f"Retrieved patent: US {found_patent_wrapper.application_meta_data.patent_number}"
            )
        else:
            print(
                f"Retrieved patent application: {found_patent_wrapper.application_number_text}"
            )

        if found_patent_wrapper.patent_term_adjustment_data:
            pta = found_patent_wrapper.patent_term_adjustment_data
            print(f"Patent Term Adjustment: {pta.adjustment_total_quantity} days")
            if pta.a_delay_quantity is not None:
                print(f"  A Delay: {pta.a_delay_quantity} days")
            if pta.b_delay_quantity is not None:
                print(f"  B Delay: {pta.b_delay_quantity} days")
            if pta.c_delay_quantity is not None:
                print(f"  C Delay: {pta.c_delay_quantity} days")
            if pta.applicant_day_delay_quantity is not None:
                print(f"  Applicant Delay: {pta.applicant_day_delay_quantity} days")

        # Example of getting continuity data (assuming it's part of the wrapper)
```

(continues on next page)

(continued from previous page)

```
        continuity_data = ApplicationContinuityData.from_wrapper(
            wrapper=found_patent_wrapper
        )
        if continuity_data.parent_continuity_bag:
            print("\nParent Applications:")
            for p_continuity in continuity_data.parent_continuity_bag:
                print(f"  - App No: {p_continuity.parent_application_number_text}")
                print(
                    f"    Type: {p_continuity.claim_parentage_type_code_description_text}
↪"
                )
                print(f"    Filing Date: {p_continuity.parent_application_filing_date}")

        if continuity_data.child_continuity_bag:
            print("\nChild Applications:")
            for c_continuity in continuity_data.child_continuity_bag:
                print(f"  - App No: {c_continuity.child_application_number_text}")
                print(
                    f"    Type: {c_continuity.claim_parentage_type_code_description_text}
↪"
                )
                print(f"    Filing Date: {c_continuity.child_application_filing_date}")
    else:
        print(f"No patents found with patent number: {target_patent_number}")

except Exception as e:
    print(f"Error retrieving patent by number {target_patent_number}: {e}")

# Example of POST search for applications
try:
    print("\nAttempting POST search for applications with 'AI' in title...")
    post_search_body = {
        "q": "applicationMetaData.inventionTitle:AI",
        "pagination": {"offset": 0, "limit": 2},
    }
    post_response = client.search_applications(post_body=post_search_body)
    print(
        f"Found {post_response.count} applications via POST search (showing up to 2)."
    )
    for patent_wrapper in post_response.patent_file_wrapper_data_bag:
        if patent_wrapper.application_meta_data:
            print(
                f"  - App No: {patent_wrapper.application_number_text}, Title: {patent_
↪wrapper.application_meta_data.invention_title}"
            )
except Exception as e:
    print(f"Error with POST search: {e}")


# Example of getting status codes
try:
    print("\nGetting first 5 status codes...")
```

(continues on next page)

(continued from previous page)

```
    status_code_response = client.get_status_codes(params={"limit": 5})
    print(
        f"Retrieved {len(status_code_response.status_code_bag)} status codes (out of
↪{status_code_response.count} total)."
    )
    for code_obj in status_code_response.status_code_bag:
        print(f"  - Code: {code_obj.code}, Description: {code_obj.description}")
except Exception as e:
    print(f"Error getting status codes: {e}")
```

4.3 Image File Wrapper Example

```
"""Example usage of pyUSPTO for IFW data.

This example demonstrates how to use the PatentDataClient to interact with the USPTO␣
↪Patent Data API.
It shows how to retrieve IFW based on various identifying values.
"""

import json
import os

from pyUSPTO.clients.patent_data import PatentDataClient

api_key = os.environ.get("USPTO_API_KEY", "YOUR_API_KEY_HERE")
if api_key == "YOUR_API_KEY_HERE":
    raise ValueError(
        "WARNING: API key is not set. Please replace 'YOUR_API_KEY_HERE' or set USPTO_
↪API_KEY environment variable."
    )

client = PatentDataClient(api_key=api_key)


print("\nBeginning API requests with configured client:")

print("\nGet IFW Based on Application Number ->")
application_number = "14412875"
app_no_ifw = client.get_IFW_metadata(application_number=application_number)
if app_no_ifw and app_no_ifw.application_meta_data:
    print(f"Title: {app_no_ifw.application_meta_data.invention_title}")
    print(f" - IFW Found based on App No: {application_number}")


print("\nGet IFW Based on Patent Number ->")
patent_number = "10765880"
pat_no_ifw = client.get_IFW_metadata(patent_number=patent_number)
if pat_no_ifw and pat_no_ifw.application_meta_data:
    print(f"Title: {pat_no_ifw.application_meta_data.invention_title}")
    print(f" - IFW Found based on Pat No: {patent_number}")

```

(continues on next page)

(continued from previous page)

```

print("\nGet IFW Based on Publication Number ->")
publication_number = "*20150157873*"
pub_no_ifw = client.get_IFW_metadata(publication_number=publication_number)
if pub_no_ifw and pub_no_ifw.application_meta_data:
    print(f"Title: {pub_no_ifw.application_meta_data.invention_title}")
    print(f" - IFW Found based on Pub No: {publication_number}")


print("\nGet IFW Based on PCT App Number ->")
PCT_app_number = "PCT/US2008/12705"
pct_app_no_ifw = client.get_IFW_metadata(PCT_app_number=PCT_app_number)
if pct_app_no_ifw and pct_app_no_ifw.application_meta_data:
    print(f"Title: {pct_app_no_ifw.application_meta_data.invention_title}")
    print(f" - IFW Found based on PCT App No: {PCT_app_number}")


print("\nGet IFW Based on PCT Pub Number ->")
PCT_pub_number = "*2009064413*"
pct_pub_no_ifw = client.get_IFW_metadata(PCT_pub_number=PCT_pub_number)
if pct_pub_no_ifw and pct_pub_no_ifw.application_meta_data:
    print(f"Title: {pct_pub_no_ifw.application_meta_data.invention_title}")
    print(f" - IFW Found based on PCT Pub No: {PCT_pub_number}")


print("\nNow let's download the Patent Publication Text -->")
if app_no_ifw and app_no_ifw.pgpub_document_meta_data:
    pgpub_archive = app_no_ifw.pgpub_document_meta_data
    print(json.dumps(pgpub_archive.to_dict(), indent=2))
    download_path = "./download-example"
    file_path = client.download_archive(
        printed_metadata=pgpub_archive, destination=download_path, overwrite=True
    )
    print(f"-Downloaded document to: {file_path}")

print("\nNow let's download the Patent Grant Text -->")
if app_no_ifw and app_no_ifw.grant_document_meta_data:
    grant_archive = app_no_ifw.grant_document_meta_data
    print(json.dumps(grant_archive.to_dict(), indent=2))
    download_path = "./download-example"
    file_path = client.download_archive(
        printed_metadata=grant_archive, destination=download_path, overwrite=True
    )
    print(f"-Downloaded document to: {file_path}")
```

4.4 Petition Decisions Example

```
"""Example usage of the pyUSPTO module for Final Petition Decisions.

This example demonstrates how to use the FinalPetitionDecisionsClient to interact with␣
↪the
```

(continues on next page)

(continued from previous page)

```
USPTO Final Petition Decisions API. It shows how to search for petition decisions, retrieve
specific decisions by ID, download decision data, and access detailed information about
petitions and their associated documents.
"""

import json
import os

from pyUSPTO.clients import FinalPetitionDecisionsClient
from pyUSPTO.models.petition_decisions import PetitionDecisionDownloadResponse

# --- Initialization ---
# Initialize the client with direct API key
print("Initialize with direct API key")
api_key = os.environ.get("USPTO_API_KEY", "YOUR_API_KEY_HERE")
if api_key == "YOUR_API_KEY_HERE":
    raise ValueError(
        "WARNING: API key is not set. Please replace 'YOUR_API_KEY_HERE' or set USPTO_API_KEY environment variable."
    )
client = FinalPetitionDecisionsClient(api_key=api_key)

DEST_PATH = "./download-example"

print("\nBeginning API requests with configured client:")

# Basic search for petition decisions
try:
    print("\n" + "=" * 60)
    print("Example 1: Basic Search for Petition Decisions")
    print("=" * 60)

    response = client.search_decisions(limit=5)
    print(f"Found {response.count} total petition decisions.")
    print(f"Displaying first {len(response.petition_decision_data_bag)} decisions:")

    for decision in response.petition_decision_data_bag:
        print(f"\n  Decision ID: {decision.petition_decision_record_identifier}")
        print(f"  Application Number: {decision.application_number_text}")
        print(f"  Decision Type: {decision.decision_type_code}")
        print(f"  Decision Date: {decision.decision_date}")
        print(f"  Technology Center: {decision.technology_center}")

        if decision.first_applicant_name:
            print(f"  Applicant: {decision.first_applicant_name}")

        if decision.patent_number:
            print(f"  Patent Number: {decision.patent_number}")

        if decision.inventor_bag:
            print(f"  Inventors ({len(decision.inventor_bag)}):")
```

(continues on next page)

(continued from previous page)

```
            for inventor in decision.inventor_bag[:3]:  # Show first 3
                print(f"    - {inventor}")

        if decision.document_bag:
            print(f"  Documents: {len(decision.document_bag)}")

        print("-" * 40)

except Exception as e:
    print(f"Error in basic search: {e}")

# Search with query parameter
try:
    print("\n" + "=" * 60)
    print("Example 2: Search with Custom Query")
    print("=" * 60)

    # Search for decisions mentioning specific terms
    response = client.search_decisions(query="decisionTypeCode:C", limit=3)
    print(f"Found {response.count} decisions with C type.")
    print(f"Showing {len(response.petition_decision_data_bag)} results:")

    for decision in response.petition_decision_data_bag:
        print(
            f"  - {decision.petition_decision_record_identifier}: {decision.decision_
↪type_code}"
        )

except Exception as e:
    print(f"Error searching with query: {e}")

# Search using convenience parameters
try:
    print("\n" + "=" * 60)
    print("Example 3: Search Using Convenience Parameters")
    print("=" * 60)

    # Search by application number (if you have a specific one)
    print("\nSearching by date range...")
    response = client.search_decisions(
        decision_date_from_q="2023-01-01", decision_date_to_q="2023-12-31", limit=5
    )
    print(f"Found {response.count} decisions from 2023.")

    # Search by technology center
    print("\nSearching by technology center...")
    response = client.search_decisions(technology_center_q="2600", limit=3)
    print(f"Found {response.count} decisions from Technology Center 2600.")

except Exception as e:
    print(f"Error with convenience parameters: {e}")

```

(continues on next page)

(continued from previous page)

```python
# Get a specific decision by ID
try:
    print("\n" + "=" * 60)
    print("Example 4: Get Specific Decision by ID")
    print("=" * 60)

    # First, get a decision ID from search results
    response = client.search_decisions(limit=1)
    if response.count > 0:
        decision_id = response.petition_decision_data_bag[
            0
        ].petition_decision_record_identifier
        if decision_id:
            print(f"Retrieving decision: {decision_id}")
            decision = client.get_decision_by_id(decision_id)
            if decision:
                print("\nDecision Details:")
                print(f"  ID: {decision.petition_decision_record_identifier}")
                print(f"  Application: {decision.application_number_text}")
                print(f"  Patent: {decision.patent_number}")
                print(f"  Decision Type: {decision.decision_type_code}")
                print(f"  Decision Date: {decision.decision_date}")
                print(f"  Technology Center: {decision.technology_center}")
                print(f"  Group Art Unit: {decision.group_art_unit_number}")

                if decision.rule_bag:
                    print(f"\n  Rules Cited ({len(decision.rule_bag)}):")
                    for rule in decision.rule_bag[:5]:  # Show first 5
                        print(f"    - {rule}")

                if decision.statute_bag:
                    print(f"\n  Statutes Cited ({len(decision.statute_bag)}):")
                    for statute in decision.statute_bag[:5]:  # Show first 5
                        print(f"    - {statute}")

                if decision.document_bag:
                    print(f"\n  Associated Documents ({len(decision.document_bag)}):")
                    for doc in decision.document_bag[:3]:  # Show first 3
                        print(f"    - Doc ID: {doc.document_identifier}")
                        print(f"      Date: {doc.official_date}")
                        print(f"      Doc. Code: {doc.document_code_description_text}")
                        print(f"      Direction: {doc.direction_category}")
                        if doc.download_option_bag:
                            print(
                                f"      Download Options: {len(doc.download_option_bag)}"
                            )
                            for mime in doc.download_option_bag:
                                print(f"       >Mime Type: {mime.mime_type_identifier}")
                                print(f"       >>Pages: {mime.page_total_quantity}")

except Exception as e:
    print(f"Error retrieving decision by ID: {e}")
```

(continues on next page)

(continued from previous page)

```
157
158 # Download petition decisions data
159 try:
160     print("\n" + "=" * 60)
161     print("Example 5: Download Petition Decisions Data")
162     print("=" * 60)
163
164     # Download as JSON (returns response object)
165     print("\nDownloading decisions as JSON...")
166     response = client.download_decisions(
167         format="json", decision_date_from_q="2023-01-01", limit=5, overwrite=True
168     )
169     if isinstance(response, PetitionDecisionDownloadResponse):
170         print(
171             f"Downloaded JSON with {len(response.petition_decision_data)} decision␣
    ↪records"
172         )
173         print(json.dumps(response.to_dict(), indent=2))
174
175     # Download as CSV (automatically saves to file)
176     print("\nDownloading decisions as CSV...")
177     csv_path = client.download_decisions(
178         format="csv",
179         decision_date_from_q="2023-01-01",
180         limit=10,
181         destination=DEST_PATH,
182         overwrite=True,
183     )
184     print(f"Downloaded CSV to: {csv_path}")
185
186 except Exception as e:
187     print(f"Error downloading decisions: {e}")
188
189 # Pagination example
190 try:
191     print("\n" + "=" * 60)
192     print("Example 6: Paginating Through Results")
193     print("=" * 60)
194
195     page_size = 10
196     max_pages = 3  # Limit to 3 pages for example
197
198     print(
199         f"Paginating through results ({page_size} per page, max {max_pages} pages)..."
200     )
201
202     total_decisions = 0
203
204     for decision in client.paginate_decisions(
205         limit=page_size, query="decisionDate:[2023-01-01 TO 2023-12-31]"
206     ):
207         total_decisions += 1
```

(continues on next page)

(continued from previous page)

```
        if total_decisions % page_size == 0:
            print(f"  Retrieved {total_decisions} decisions so far...")

        if total_decisions >= (page_size * max_pages):
            print(f"  (Stopping after {max_pages} pages for example)")
            break

    print(f"\nTotal decisions retrieved: {total_decisions}")

except Exception as e:
    print(f"Error during pagination: {e}")

# Download a petition document
try:
    print("\n" + "=" * 60)
    print("Example 7: Download Petition Decision Document")
    print("=" * 60)

    # Find a decision with downloadable documents
    response = client.search_decisions(limit=20)

    document_found = False
    for decision in response.petition_decision_data_bag:
        d = client.get_decision_by_id(
            decision.petition_decision_record_identifier, include_documents=True
        )
        print(f"Getting docs for patent: {d.invention_title} with id: {d.petition_
↪decision_record_identifier}")  # type: ignore
        if d and d.document_bag:
            for doc in d.document_bag:
                if doc.download_option_bag and len(doc.download_option_bag) > 0:
                    download_option = doc.download_option_bag[0]

                    print("Found downloadable document:")
                    print(f"  Document ID: {doc.document_identifier}")
                    print(f"  MIME Type: {download_option.mime_type_identifier}")
                    print(f"  Pages: {download_option.page_total_quantity}")
                    print(f"  URL: {download_option.download_url}")

                    print("\nDownloading document...")
                    file_path = client.download_petition_document(
                        download_option=download_option,
                        destination=DEST_PATH,
                    )
                    print(f"Downloaded to: {file_path}")

                    document_found = True
                    break

        if document_found:
            break
```

(continues on next page)

(continued from previous page)

```
    if not document_found:
        print("No downloadable documents found in the first 20 results")

except Exception as e:
    print(f"Error downloading document: {e}")

# Advanced search example
try:
    print("\n" + "=" * 60)
    print("Example 8: Advanced Search with Multiple Criteria")
    print("=" * 60)

    # Search with multiple parameters
    response = client.search_decisions(
        application_number_q="16*",  # Applications starting with 16
        decision_date_from_q="2020-01-01",
        technology_center_q="2600",
        limit=10,
    )

    print("Search criteria:")
    print("  - Application numbers starting with '16'")
    print("  - Decision date from 2020-01-01")
    print("  - Technology Center 2600")
    print(f"\nFound {response.count} matching decisions")

    if response.count > 0:
        print(f"Showing first {len(response.petition_decision_data_bag)} results:")
        for decision in response.petition_decision_data_bag:
            print(
                f"  - App: {decision.application_number_text}, "
                f"TC: {decision.technology_center}, "
                f"Date: {decision.decision_date}"
            )

except Exception as e:
    print(f"Error in advanced search: {e}")

print("\n" + "=" * 60)
print("Examples completed!")
print("=" * 60)
```

4.5 PTAB Appeals Example

```
"""Example usage of the pyUSPTO module for PTAB Appeals API.

This example demonstrates how to use the PTABAppealsClient to interact with the USPTO␣
↪PTAB
(Patent Trial and Appeal Board) Appeals API. It shows how to search for ex parte appeal
decisions using various search criteria.
```

(continues on next page)

(continued from previous page)

```

PTAB Appeals include ex parte appeals from patent application examinations to the Board.
"""

import os

from pyUSPTO import PTABAppealsClient

# --- Initialization ---
# Initialize the client with direct API key
print("Initialize with direct API key")
api_key = os.environ.get("USPTO_API_KEY", "YOUR_API_KEY_HERE")
if api_key == "YOUR_API_KEY_HERE":
    raise ValueError(
        "WARNING: API key is not set. Please replace 'YOUR_API_KEY_HERE' or set USPTO_API_KEY environment variable."
    )
client = PTABAppealsClient(api_key=api_key)


print("\nBeginning PTAB Appeals API requests with configured client:")
# ================================================================================
# 1. Search Appeal Decisions by Technology Center
# ================================================================================

print("\n" + "=" * 80)
print("1. Searching for appeal decisions by technology center")
print("=" * 80)

try:
    # Search for decisions from Technology Center 3600 (Business Methods/Software)
    response = client.search_decisions(
        technology_center_number_q="3600",
        decision_date_from_q="2023-01-01",
        decision_date_to_q="2023-12-31",
        limit=5,
    )

    print(f"\nFound {response.count} appeal decisions from TC 3600 in 2023")
    print(f"Displaying first {len(response.patent_appeal_data_bag)} results:")

    for decision in response.patent_appeal_data_bag:
        print(f"\n  Appeal Number: {decision.appeal_number}")

        if decision.appeal_meta_data:
            meta = decision.appeal_meta_data
            print(f"  Application Type: {meta.application_type_category}")
            print(f"  Filing Date: {meta.appeal_filing_date}")

        if decision.appellant_data:
            appellant = decision.appellant_data
            print(f"  Application Number: {appellant.application_number_text}")
```

(continues on next page)

(continued from previous page)

```
            print(f"  Technology Center: {appellant.technology_center_number}")

            if appellant.inventor_name:
                print(f"  Inventor: {appellant.inventor_name}")

        if decision.decision_data:
            dec = decision.decision_data
            print(f"  Decision Type: {dec.decision_type_category}")
            print(f"  Decision Date: {dec.decision_issue_date}")

except Exception as e:
    print(f"Error searching appeal decisions: {e}")

# ============================================================================
# 2. Search by Decision Type
# ============================================================================

print("\n" + "=" * 80)
print("2. Searching for 'Affirmed' decisions")
print("=" * 80)

try:
    # Search for decisions where the examiner was affirmed
    response = client.search_decisions(
        decision_type_category_q="Decision",
        decision_date_from_q="2024-01-01",
        limit=5,
    )

    print(f"\nFound {response.count} 'Decision's since 2024")
    print(f"Displaying first {len(response.patent_appeal_data_bag)} results:")

    for decision in response.patent_appeal_data_bag:
        print(f"\n  Appeal Number: {decision.appeal_number}")

        if decision.appellant_data:
            print(f"  Application: {decision.appellant_data.application_number_text}")
            print(f"  Inventor: {decision.appellant_data.inventor_name or 'N/A'}")

        if decision.decision_data:
            print(f"  Decision: {decision.decision_data.decision_type_category}")
            print(f"  Outcome: {decision.decision_data.appeal_outcome_category}")
            print(f"  Date: {decision.decision_data.decision_issue_date}")

except Exception as e:
    print(f"Error searching by decision type: {e}")

# ============================================================================
# 3. Search by Application Number
# ============================================================================

print("\n" + "=" * 80)
```

(continues on next page)

(continued from previous page)

```python
print("3. Searching for decisions by application number pattern")
print("=" * 80)

try:
    # Search for decisions related to applications starting with "15"
    response = client.search_decisions(
        application_number_text_q="15*",
        decision_date_from_q="2023-01-01",
        limit=3,
    )

    print(f"\nFound {response.count} decisions for applications starting with '15/'")
    print(f"Displaying first {len(response.patent_appeal_data_bag)} results:")

    for decision in response.patent_appeal_data_bag:
        print(f"\n  Appeal Number: {decision.appeal_number}")

        if decision.appellant_data:
            print(f"  Application: {decision.appellant_data.application_number_text}")
            print(f"  TC Number: {decision.appellant_data.technology_center_number}")

        if decision.document_data:
            doc = decision.document_data
            print(f"  Document Name: {doc.document_name}")
            if doc.file_download_uri:
                print(f"  Download URL: {doc.file_download_uri}")

except Exception as e:
    print(f"Error searching by application number: {e}")

# ============================================================================
# 4. Pagination Example
# ============================================================================

print("\n" + "=" * 80)
print("4. Paginating through appeal decisions")
print("=" * 80)

try:
    print("\nIterating through first 10 appeal decisions from 2024...")
    count = 0
    for decision in client.paginate_decisions(
        decision_date_from_q="2024-01-01",
        limit=5,  # Fetch 5 per page
    ):
        count += 1
        decision_type = (
            decision.decision_data.decision_type_category
            if decision.decision_data
            else "N/A"
        )
        print(f"{count}. {decision.appeal_number} - {decision_type}")
```

(continues on next page)

(continued from previous page)

```
        if count >= 10:  # Stop after 10 results for this example
            break

    print(f"\nDisplayed {count} decisions using pagination")

except Exception as e:
    print(f"Error paginating decisions: {e}")

# ================================================================================
# 5. Advanced Search with Multiple Criteria
# ================================================================================

print("\n" + "=" * 80)
print("5. Advanced search with multiple criteria")
print("=" * 80)

try:
    # Search with multiple convenience parameters
    response = client.search_decisions(
        technology_center_number_q="2100",  # Electronics
        decision_type_category_q="Decision",
        decision_date_from_q="2023-01-01",
        decision_date_to_q="2023-12-31",
        sort="decisionData.decisionIssueDate desc",
        limit=3,
    )

    print(f"\nFound {response.count} Decisions from TC 2100 (Electronics) in 2023")
    print(f"Displaying first {len(response.patent_appeal_data_bag)} results:")

    for decision in response.patent_appeal_data_bag:
        print(f"\n  Appeal Number: {decision.appeal_number}")

        if decision.appellant_data:
            print(f"  Application: {decision.appellant_data.application_number_text}")

        if decision.decision_data:
            print(f"  Decision: {decision.decision_data.decision_type_category}")
            print(f"  Date: {decision.decision_data.decision_issue_date}")

except Exception as e:
    print(f"Error with advanced search: {e}")

# ================================================================================
# 6. Direct Query String Example
# ================================================================================

print("\n" + "=" * 80)
print("6. Using direct query string for complex searches")
print("=" * 80)

```

(continues on next page)

(continued from previous page)

```
try:
    # Use a direct query string for more complex searches
    response = client.search_decisions(
        query="appellantData.technologyCenterNumber:3600 AND decisionData.
↪appealOutcomeCategory:(Affirmed OR Reversed)",
        limit=10,
    )

    print(f"\nFound {response.count} Affirmed/Reversed decisions from TC 3600")
    print(f"Displaying first {len(response.patent_appeal_data_bag)} results:")

    for decision in response.patent_appeal_data_bag:
        print(f"\n  Appeal Number: {decision.appeal_number}")
        print(f"  >App. Number: {decision.appellant_data.application_number_text}")  #␣
↪type: ignore
        if decision.decision_data:
            print(f"  >Decision: {decision.decision_data.decision_type_category}")
            print(f"  >Outcome: {decision.decision_data.appeal_outcome_category}")

except Exception as e:
    print(f"Error with direct query: {e}")

# ============================================================================
# 7. Error Handling Example
# ============================================================================

print("\n" + "=" * 80)
print("7. Error handling demonstration")
print("=" * 80)

try:
    # Attempt a search that might return no results
    print("\nAttempting search with unlikely parameters...")
    response = client.search_decisions(
        appeal_number_q="INVALID-APPEAL-NUMBER",
        limit=1,
    )

    if response.count == 0:
        print("No results found for the given search criteria")
    else:
        print(f"Found {response.count} results")

except Exception as e:
    print(f"Expected error occurred: {type(e).__name__}: {e}")

print("\n" + "=" * 80)
print("PTAB Appeals API example completed successfully!")
print("=" * 80)
```

4.6 PTAB Interferences Example

```python
"""Example usage of the pyUSPTO module for PTAB Interferences API.

This example demonstrates how to use the PTABInterferencesClient to interact with the USPTO PTAB
(Patent Trial and Appeal Board) Interferences API. It shows how to search for interference
decisions using various search criteria.

PTAB Interferences are proceedings to determine priority of invention when two or more parties
claim the same patentable invention.
"""

import os

from pyUSPTO import PTABInterferencesClient

# --- Initialization ---
# Initialize the client with direct API key
print("Initialize with direct API key")
api_key = os.environ.get("USPTO_API_KEY", "YOUR_API_KEY_HERE")
if api_key == "YOUR_API_KEY_HERE":
    raise ValueError(
        "WARNING: API key is not set. Please replace 'YOUR_API_KEY_HERE' or set USPTO_API_KEY environment variable."
    )
client = PTABInterferencesClient(api_key=api_key)

print("\nBeginning PTAB Interferences API requests with configured client:")

# ================================================================================
# 1. Search Interference Decisions
# ================================================================================

print("\n" + "=" * 80)
print("1. Searching for interference decisions")
print("=" * 80)

try:
    # Search for recent interference decisions
    response = client.search_decisions(
        decision_date_from_q="2023-01-01",
        limit=5,
    )

    print(f"\nFound {response.count} interference decisions since 2023")
    print(f"Displaying first {len(response.patent_interference_data_bag)} results:")

    for decision in response.patent_interference_data_bag:
        print(f"\n  Interference Number: {decision.interference_number}")

```

(continues on next page)

(continued from previous page)

```
        if decision.interference_meta_data:
            meta = decision.interference_meta_data
            print(f"  Style Name: {meta.interference_style_name}")
            print(f"  Last Modified: {meta.interference_last_modified_date}")

        if decision.senior_party_data:
            senior = decision.senior_party_data
            print(f"  Senior Party: {senior.patent_owner_name}")
            if senior.patent_number:
                print(f"  Senior Patent: {senior.patent_number}")

        if decision.junior_party_data:
            junior = decision.junior_party_data
            print(f"  Junior Party: {junior.patent_owner_name}")
            if junior.publication_number:
                print(f"  Junior Publication: {junior.publication_number}")

        if decision.document_data:
            doc = decision.document_data
            print(f"  Outcome: {doc.interference_outcome_category}")
            print(f"  Decision Type: {doc.decision_type_category}")

except Exception as e:
    print(f"Error searching interference decisions: {e}")

# ============================================================================
# 2. Search by Interference Outcome
# ============================================================================

print("\n" + "=" * 80)
print("2. Searching for decisions by outcome")
print("=" * 80)

try:
    # Search for decisions with specific outcomes
    response = client.search_decisions(
        interference_outcome_category_q="Final Decision",
        decision_date_from_q="2012-01-01",
        limit=3,
    )

    print(f"\nFound {response.count} final decisions since 2012")
    print(f"Displaying first {len(response.patent_interference_data_bag)} results:")

    for decision in response.patent_interference_data_bag:
        print(f"\n  Interference Number: {decision.interference_number}")

        if decision.senior_party_data:
            print(f"  Senior Party: {decision.senior_party_data.patent_owner_name}")
            print(
                f"  Senior Application: {decision.senior_party_data.application_number_
↪text}"
```

(continues on next page)

(continued from previous page)

```
            )

        if decision.junior_party_data:
            print(f"  Junior Party: {decision.junior_party_data.patent_owner_name}")

        if decision.document_data:
            print(f"  Outcome: {decision.document_data.interference_outcome_category}")
            print(f"  Decision Date: {decision.document_data.decision_issue_date}")

except Exception as e:
    print(f"Error searching by outcome: {e}")

# ============================================================================
# 3. Search by Party Name
# ============================================================================

print("\n" + "=" * 80)
print("3. Searching for decisions by party name")
print("=" * 80)

try:
    # Search for decisions involving a specific senior party
    response = client.search_decisions(
        senior_party_name_q="*Corp*",  # Any company with "Corp" in the name
        limit=3,
    )

    print(f"\nFound {response.count} decisions with 'Corp' in senior party name")
    print(f"Displaying first {len(response.patent_interference_data_bag)} results:")

    for decision in response.patent_interference_data_bag:
        print(f"\n  Interference Number: {decision.interference_number}")

        if decision.senior_party_data:
            senior = decision.senior_party_data
            print(f"  Senior Party: {senior.patent_owner_name}")
            if senior.counsel_name:
                print(f"  Senior Counsel: {senior.counsel_name}")

        if decision.junior_party_data:
            junior = decision.junior_party_data
            print(f"  Junior Party: {junior.patent_owner_name}")
            if junior.counsel_name:
                print(f"  Junior Counsel: {junior.counsel_name}")

except Exception as e:
    print(f"Error searching by party name: {e}")

# ============================================================================
# 4. Search by Application Numbers
# ============================================================================

```

(continues on next page)

(continued from previous page)

```
print("\n" + "=" * 80)
print("4. Searching for decisions by application numbers")
print("=" * 80)

try:
    # Search for decisions involving specific application numbers
    response = client.search_decisions(
        senior_party_application_number_q="12*",  # Applications starting with 12/
        limit=3,
    )

    print(
        f"\nFound {response.count} decisions with senior applications starting with '12'"
    )
    print(f"Displaying first {len(response.patent_interference_data_bag)} results:")

    for decision in response.patent_interference_data_bag:
        print(f"\n  Interference Number: {decision.interference_number}")

        if decision.senior_party_data:
            print(
                f"  Senior Application: {decision.senior_party_data.application_number_
↪text}"
            )

        if decision.junior_party_data:
            print(
                f"  Junior Publication: {decision.junior_party_data.publication_number}"
            )

        if decision.document_data:
            print(f"  Decision Type: {decision.document_data.decision_type_category}")

except Exception as e:
    print(f"Error searching by application numbers: {e}")

# ============================================================================
# 5. Pagination Example
# ============================================================================

print("\n" + "=" * 80)
print("5. Paginating through interference decisions")
print("=" * 80)

try:
    print("\nIterating through first 5 interference decisions from 2023...")
    count = 0
    for decision in client.paginate_decisions(
        decision_date_from_q="2023-01-01",
        limit=3,  # Fetch 3 per page
    ):
        count += 1
```

(continues on next page)

(continued from previous page)

```
        outcome = (
            decision.document_data.interference_outcome_category
            if decision.document_data
            else "N/A"
        )
        print(f"{count}. {decision.interference_number} - {outcome}")

        if count >= 5:  # Stop after 5 results for this example
            break

    print(f"\nDisplayed {count} decisions using pagination")

except Exception as e:
    print(f"Error paginating decisions: {e}")

# ============================================================================
# 6. Advanced Search with Multiple Criteria
# ============================================================================

print("\n" + "=" * 80)
print("6. Advanced search with multiple criteria")
print("=" * 80)

try:
    # Search with multiple convenience parameters
    response = client.search_decisions(
        decision_type_category_q="Decision",
        decision_date_from_q="2020-01-01",
        decision_date_to_q="2023-12-31",
        sort="documentData.decisionIssueDate desc",
        limit=3,
    )

    print(f"\nFound {response.count} Decisions between 2020-2023")
    print(f"Displaying first {len(response.patent_interference_data_bag)} results:")

    for decision in response.patent_interference_data_bag:
        print(f"\n  Interference Number: {decision.interference_number}")

        if decision.interference_meta_data:
            print(f"  Style: {decision.interference_meta_data.interference_style_name}")

        if decision.document_data:
            print(f"  Decision Type: {decision.document_data.decision_type_category}")
            print(f"  Decision Date: {decision.document_data.decision_issue_date}")
            print(f"  Outcome: {decision.document_data.interference_outcome_category}")

        # Show additional parties if present
        if decision.additional_party_data_bag:
            print(f"  Additional Parties: {len(decision.additional_party_data_bag)}")
            for party in decision.additional_party_data_bag:
                print(f"    - {party.additional_party_name}")
```

(continues on next page)

(continued from previous page)

```

except Exception as e:
    print(f"Error with advanced search: {e}")

# ============================================================================
# 7. Direct Query String Example
# ============================================================================

print("\n" + "=" * 80)
print("7. Using direct query string for complex searches")
print("=" * 80)

try:
    # Use a direct query string for more complex searches
    response = client.search_decisions(
        query='documentData.interferenceOutcomeCategory:"Final Decision"',
        limit=3,
    )

    print(f"\nFound {response.count} final decisions.")
    print(f"Displaying first {len(response.patent_interference_data_bag)} results:")

    for decision in response.patent_interference_data_bag:
        print(f"\n  Interference Number: {decision.interference_number}")

        if decision.document_data:
            print(f"  Outcome: {decision.document_data.interference_outcome_category}")

except Exception as e:
    print(f"Error with direct query: {e}")

print("\n" + "=" * 80)
print("PTAB Interferences API example completed successfully!")
print("=" * 80)
```

4.7 PTAB Trials Example

```
"""Example usage of the pyUSPTO module for PTAB Trials API.

This example demonstrates how to use the PTABTrialsClient to interact with the USPTO PTAB
(Patent Trial and Appeal Board) Trials API. It shows how to search for trial proceedings,
documents, and decisions using various search criteria.

PTAB Trials include:
- IPR (Inter Partes Review)
- PGR (Post-Grant Review)
- CBM (Covered Business Method)
- DER (Derivation) proceedings
"""

import os
```

(continues on next page)

(continued from previous page)

```python

from pyUSPTO import PTABTrialsClient

# --- Initialization ---
# Initialize the client with direct API key
print("Initialize with direct API key")
api_key = os.environ.get("USPTO_API_KEY", "YOUR_API_KEY_HERE")
if api_key == "YOUR_API_KEY_HERE":
    raise ValueError(
        "WARNING: API key is not set. Please replace 'YOUR_API_KEY_HERE' or set USPTO_API_KEY environment variable."
    )
client = PTABTrialsClient(api_key=api_key)

print("\nBeginning PTAB Trials API requests with configured client:")

# ================================================================================
# 1. Search Trial Proceedings
# ================================================================================

print("\n" + "=" * 80)
print("1. Searching for IPR trial proceedings")
print("=" * 80)

try:
    # Search for IPR proceedings filed in 2023
    response = client.search_proceedings(
        trial_type_code_q="IPR",
        petition_filing_date_from_q="2023-01-01",
        petition_filing_date_to_q="2023-12-31",
        limit=5,
    )

    print(f"\nFound {response.count} IPR proceedings filed in 2023")
    print(f"Displaying first {len(response.patent_trial_proceeding_data_bag)} results:")

    for proceeding in response.patent_trial_proceeding_data_bag:
        print(f"\n  Trial Number: {proceeding.trial_number}")

        if proceeding.trial_meta_data:
            meta = proceeding.trial_meta_data
            print(f"  Trial Type: {meta.trial_type_code}")
            print(f"  Status: {meta.trial_status_category}")
            print(f"  Filing Date: {meta.petition_filing_date}")

        if proceeding.patent_owner_data:
            print(f"  Patent Owner: {proceeding.patent_owner_data.patent_owner_name}")
            print(f"  Patent Number: {proceeding.patent_owner_data.patent_number}")

        if proceeding.regular_petitioner_data:
            print(
                f"  Petitioner: {proceeding.regular_petitioner_data.real_party_in_
```

(continues on next page)

(continued from previous page)

```
↪interest_name}"
            )

except Exception as e:
    print(f"Error searching proceedings: {e}")

# ============================================================================
# 2. Search Trial Documents
# ============================================================================

print("\n" + "=" * 80)
print("2. Searching for trial documents")
print("=" * 80)

try:
    # Search for documents in a specific trial
    # Using the new convenience parameters for petitioner and patent owner
    response = client.search_documents(
        trial_number_q="IPR2025-01319",
        limit=10,
    )

    print(f"\nFound {response.count} documents")
    print(f"Displaying first {len(response.patent_trial_document_data_bag)} results:")

    for item in response.patent_trial_document_data_bag:
        print(f"\n  Trial Number: {item.trial_number}")

        if item.document_data:
            doc = item.document_data
            print(f"  Document Type: {doc.document_type_description_text}")
            print(f"  Filing Date: {doc.document_filing_date}")

            if doc.file_download_uri:
                print(f"  Download URL: {doc.file_download_uri}")

except Exception as e:
    print(f"Error searching documents: {e}")

# ============================================================================
# 3. Search Trial Decisions with New Convenience Parameters
# ============================================================================

print("\n" + "=" * 80)
print("3. Searching for trial decisions with new parameters")
print("=" * 80)

try:
    # Using all the new convenience parameters
    response = client.search_decisions(
        trial_type_code_q="IPR",
        decision_type_category_q="Decision",
```

(continues on next page)

(continued from previous page)

```
        patent_owner_name_q="*",
        trial_status_category_q="Terminated",
        decision_date_from_q="2023-01-01",
        limit=5,
    )

    print(f"\nFound {response.count} Decisions in IPR proceedings")
    print(f"Displaying first {len(response.patent_trial_document_data_bag)} results:")

    for item in response.patent_trial_document_data_bag:
        print(f"\n  Trial Number: {item.trial_number}")

        if item.trial_meta_data:
            print(f"  Trial Type: {item.trial_meta_data.trial_type_code}")
            print(f"  Status: {item.trial_meta_data.trial_status_category}")

        if item.decision_data:
            decision = item.decision_data
            print(f"  Decision Type: {decision.decision_type_category}")
            print(f"  Decision Date: {decision.decision_issue_date}")

except Exception as e:
    print(f"Error searching decisions: {e}")

# ============================================================================
# 4. Pagination Example
# ============================================================================

print("\n" + "=" * 80)
print("4. Paginating through proceedings")
print("=" * 80)

try:
    print("\nIterating through first 10 IPR proceedings from 2024...")
    count = 0
    for proceeding in client.paginate_proceedings(
        trial_type_code_q="IPR",
        petition_filing_date_from_q="2024-01-01",
        limit=5,  # Fetch 5 per page
    ):
        count += 1
        print(f"{count}. {proceeding.trial_number}")

        if count >= 10:  # Stop after 10 results for this example
            break

    print(f"\nDisplayed {count} proceedings using pagination")

except Exception as e:
    print(f"Error paginating proceedings: {e}")

# ============================================================================
```

(continues on next page)

(continued from previous page)

```
# 5. Advanced Query with Additional Parameters
# ============================================================================

print("\n" + "=" * 80)
print("5. Advanced search with additional query parameters")
print("=" * 80)

try:
    # Search using additional_query_params for custom filters
    response = client.search_proceedings(
        trial_type_code_q="PGR",
        trial_status_category_q="Terminated",
        sort="trialMetaData.petitionFilingDate desc",
        fields="trialNumber,lastModifiedDateTime",
        limit=3,
    )

    print(f"\nFound {response.count} Instituted PGR proceedings")
    print(f"Displaying first {len(response.patent_trial_proceeding_data_bag)} results:")

    for proceeding in response.patent_trial_proceeding_data_bag:
        print(f"\n  Trial Number: {proceeding.trial_number}")
        print(f"  Last Modified: {proceeding.last_modified_date_time}")

except Exception as e:
    print(f"Error with advanced search: {e}")
```

CHAPTER

FIVE

DEVELOPMENT

5.1 Contributing

See the CONTRIBUTING.md file for details on how to contribute to the project.

5.2 Testing

The library includes a comprehensive test suite:

```
# Run all tests
python -m pytest

# Run with coverage report
python -m pytest --cov=pyUSPTO
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

pyUSPTO.clients.bulk_data, 5
pyUSPTO.clients.patent_data, 7
pyUSPTO.clients.petition_decisions, 17
pyUSPTO.clients.ptab_appeals, 22
pyUSPTO.clients.ptab_interferences, 25
pyUSPTO.clients.ptab_trials, 29
pyUSPTO.config, 99
pyUSPTO.exceptions, 101
pyUSPTO.http_config, 100
pyUSPTO.models.bulk_data, 34
pyUSPTO.models.patent_data, 36
pyUSPTO.models.ptab, 75
pyUSPTO.models.utils, 97
pyUSPTO.warnings, 105

INDEX

Symbols

__format__() (*pyUSPTO.models.patent_data.DocumentMimeType method*), 58
__getitem__() (*pyUSPTO.models.patent_data.DocumentBag method*), 55
__getitem__() (*pyUSPTO.models.patent_data.StatusCodeCollection method*), 72
__init__() (*pyUSPTO.clients.bulk_data.BulkDataClient method*), 5
__init__() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 8
__init__() (*pyUSPTO.clients.petition_decisions.FinalPetitionDecisionsClient method*), 18
__init__() (*pyUSPTO.clients.ptab_appeals.PTABAppealsClient method*), 22
__init__() (*pyUSPTO.clients.ptab_interferences.PTABInterferencesClient method*), 25
__init__() (*pyUSPTO.clients.ptab_trials.PTABTrialsClient method*), 29
__init__() (*pyUSPTO.config.USPTOConfig method*), 99
__init__() (*pyUSPTO.exceptions.FormatNotAvailableError method*), 102
__init__() (*pyUSPTO.exceptions.USPTOApiError method*), 103
__init__() (*pyUSPTO.models.patent_data.DocumentBag method*), 55
__init__() (*pyUSPTO.models.patent_data.StatusCodeCollection method*), 72
__iter__() (*pyUSPTO.models.patent_data.DocumentBag method*), 55
__iter__() (*pyUSPTO.models.patent_data.StatusCodeCollection method*), 72
__len__() (*pyUSPTO.models.patent_data.DocumentBag method*), 55
__len__() (*pyUSPTO.models.patent_data.StatusCodeCollection method*), 73
__post_init__() (*pyUSPTO.http_config.HTTPConfig method*), 100
__repr__() (*pyUSPTO.models.patent_data.Document method*), 53
__repr__() (*pyUSPTO.models.patent_data.DocumentBag method*), 55
__repr__() (*pyUSPTO.models.patent_data.DocumentFormat method*), 57
__repr__() (*pyUSPTO.models.patent_data.StatusCodeCollection method*), 73
__str__() (*pyUSPTO.exceptions.USPTOApiError method*), 103
__str__() (*pyUSPTO.models.patent_data.Document method*), 53
__str__() (*pyUSPTO.models.patent_data.DocumentBag method*), 56
__str__() (*pyUSPTO.models.patent_data.DocumentFormat method*), 57
__str__() (*pyUSPTO.models.patent_data.StatusCode method*), 71
__str__() (*pyUSPTO.models.patent_data.StatusCodeCollection method*), 73

A

a_delay_quantity (*pyUSPTO.models.patent_data.PatentTermAdjustment attribute*), 65, 66
accorded_filing_date (*pyUSPTO.models.ptab.TrialMetaData attribute*), 95, 96
ACTIVE (*pyUSPTO.models.patent_data.ActiveIndicator attribute*), 36
active_indicator (*pyUSPTO.models.patent_data.Attorney attribute*), 48
ActiveIndicator (*class in pyUSPTO.models.patent_data*), 36
additional_party_data_bag (*pyUSPTO.models.ptab.PTABInterferenceDecision attribute*), 85
additional_party_name (*pyUSPTO.models.ptab.AdditionalPartyData attribute*), 75
AdditionalPartyData (*class in pyUSPTO.models.ptab*), 75
Address (*class in pyUSPTO.models.patent_data*), 37
address_line_four_text (*pyUSPTO.models.patent_data.Address attribute*), 37, 38

address_line_one_text (*pyUSPTO.models.patent_data.Address attribute*), 37, 38
address_line_three_text (*pyUSPTO.models.patent_data.Address attribute*), 37, 38
address_line_two_text (*pyUSPTO.models.patent_data.Address attribute*), 37, 38
adjustment_total_quantity (*pyUSPTO.models.patent_data.PatentTermAdjustmentData attribute*), 65, 66
api_short_error (*pyUSPTO.exceptions.APIErrorArgs attribute*), 102
APIErrorArgs (*class in pyUSPTO.exceptions*), 101
appeal_document_category (*pyUSPTO.models.ptab.PTABAppealDecision attribute*), 83
appeal_filing_date (*pyUSPTO.models.ptab.AppealMetaData attribute*), 77
appeal_last_modified_date (*pyUSPTO.models.ptab.AppealMetaData attribute*), 77, 78
appeal_last_modified_date_time (*pyUSPTO.models.ptab.AppealMetaData attribute*), 77, 78
appeal_meta_data (*pyUSPTO.models.ptab.PTABAppealDecision attribute*), 83
appeal_number (*pyUSPTO.models.ptab.PTABAppealDecision attribute*), 82, 83
appeal_outcome_category (*pyUSPTO.models.ptab.DecisionData attribute*), 78, 79
AppealDocumentData (*class in pyUSPTO.models.ptab*), 76
AppealMetaData (*class in pyUSPTO.models.ptab*), 77
appellant_data (*pyUSPTO.models.ptab.PTABAppealDecision attribute*), 83
AppellantData (*class in pyUSPTO.models.ptab*), 78
Applicant (*class in pyUSPTO.models.patent_data*), 39
applicant_bag (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 43
applicant_day_delay_quantity (*pyUSPTO.models.patent_data.PatentTermAdjustmentData attribute*), 65, 66
applicant_day_delay_quantity (*pyUSPTO.models.patent_data.PatentTermAdjustmentHistoryData attribute*), 67
applicant_name_text (*pyUSPTO.models.patent_data.Applicant attribute*), 39
application_confirmation_number (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 43
application_meta_data (*pyUSPTO.models.patent_data.PatentFileWrapper attribute*), 64
application_number_text (*pyUSPTO.models.patent_data.Continuity attribute*), 50, 51
application_number_text (*pyUSPTO.models.patent_data.Document attribute*), 53
application_number_text (*pyUSPTO.models.patent_data.ForeignPriority attribute*), 60
application_number_text (*pyUSPTO.models.patent_data.PatentFileWrapper attribute*), 63, 64
application_number_text (*pyUSPTO.models.ptab.AdditionalPartyData attribute*), 75
application_status_code (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 43
application_status_date (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 43
application_status_description_text (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 43
application_type_category (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 43
application_type_category (*pyUSPTO.models.ptab.AppealMetaData attribute*), 77, 78
application_type_code (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 43
application_type_label_name (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 43
ApplicationContinuityData (*class in pyUSPTO.models.patent_data*), 39
ApplicationMetaData (*class in pyUSPTO.models.patent_data*), 40
Assignee (*class in pyUSPTO.models.patent_data*), 45
assignee_address (*pyUSPTO.models.patent_data.Assignee attribute*), 45
assignee_bag (*pyUSPTO.models.patent_data.Assignment attribute*), 46, 47
assignee_name_text (*pyUSPTO.models.patent_data.Assignee attribute*), 45
Assignment (*class in pyUSPTO.models.patent_data*), 45
assignment_bag (*pyUSPTO.models.patent_data.PatentFileWrapper attribute*), 64
assignment_document_location_uri

(*pyUSPTO.models.patent_data.Assignment attribute*), 46, 47
assignment_mailed_date (*pyUSPTO.models.patent_data.Assignment attribute*), 46, 47
assignment_received_date (*pyUSPTO.models.patent_data.Assignment attribute*), 46, 47
assignment_recorded_date (*pyUSPTO.models.patent_data.Assignment attribute*), 46, 47
Assignor (*class in pyUSPTO.models.patent_data*), 47
assignor_bag (*pyUSPTO.models.patent_data.Assignment attribute*), 46, 47
assignor_name (*pyUSPTO.models.patent_data.Assignor attribute*), 47, 48
Attorney (*class in pyUSPTO.models.patent_data*), 48
attorney_address_bag (*pyUSPTO.models.patent_data.Attorney attribute*), 48
attorney_bag (*pyUSPTO.models.patent_data.RecordAttorney attribute*), 71
attorney_docket_number (*pyUSPTO.models.patent_data.Assignment attribute*), 46, 47
available_formats (*pyUSPTO.exceptions.FormatNotAvailableError attribute*), 102

B

b_delay_quantity (*pyUSPTO.models.patent_data.PatentTermAdjustmentData attribute*), 65, 66
backoff_factor (*pyUSPTO.http_config.HTTPConfig attribute*), 100, 101
bulk_data_product_bag (*pyUSPTO.models.bulk_data.BulkDataResponse attribute*), 35
BulkDataClient (*class in pyUSPTO.clients.bulk_data*), 5
BulkDataProduct (*class in pyUSPTO.models.bulk_data*), 34
BulkDataResponse (*class in pyUSPTO.models.bulk_data*), 35
business_entity_status_category (*pyUSPTO.models.patent_data.EntityStatus attribute*), 58

C

c_delay_quantity (*pyUSPTO.models.patent_data.PatentTermAdjustmentData attribute*), 65, 66
child_application_filing_date (*pyUSPTO.models.patent_data.ChildContinuity attribute*), 50
child_application_number_text (*pyUSPTO.models.patent_data.ChildContinuity attribute*), 49, 50
child_application_number_text (*pyUSPTO.models.patent_data.ParentContinuity attribute*), 61
child_application_status_code (*pyUSPTO.models.patent_data.ChildContinuity attribute*), 49, 50
child_application_status_description_text (*pyUSPTO.models.patent_data.ChildContinuity attribute*), 49, 50
child_continuity_bag (*pyUSPTO.models.patent_data.ApplicationContinuityData attribute*), 40
child_continuity_bag (*pyUSPTO.models.patent_data.PatentFileWrapper attribute*), 64
child_patent_number (*pyUSPTO.models.patent_data.ChildContinuity attribute*), 50
ChildContinuity (*class in pyUSPTO.models.patent_data*), 49
city_name (*pyUSPTO.models.patent_data.Address attribute*), 37, 38
claim_parentage_type_code (*pyUSPTO.models.patent_data.Continuity attribute*), 51
claim_parentage_type_code_description_text (*pyUSPTO.models.patent_data.Continuity attribute*), 51
class_field (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 43
code (*pyUSPTO.models.patent_data.StatusCode attribute*), 71
connect_timeout (*pyUSPTO.http_config.HTTPConfig attribute*), 100, 101
Continuity (*class in pyUSPTO.models.patent_data*), 50
conveyance_text (*pyUSPTO.models.patent_data.Assignment attribute*), 46, 47
correspondence_address (*pyUSPTO.models.patent_data.Assignment attribute*), 46, 47
correspondence_address_bag (*pyUSPTO.models.patent_data.Applicant attribute*), 39
correspondence_address_bag (*pyUSPTO.models.patent_data.Inventor attribute*), 60
correspondence_address_bag (*pyUSPTO.models.patent_data.PatentFileWrapper attribute*), 64
correspondent_name_text (*pyUSPTO.models.patent_data.Address attribute*), 38
counsel_name (*pyUSPTO.models.ptab.RegularPetitionerData*

attribute), 91
count (*pyUSPTO.models.bulk_data.BulkDataResponse attribute*), 35
count (*pyUSPTO.models.bulk_data.ProductFileBag attribute*), 36
count (*pyUSPTO.models.patent_data.PatentDataResponse attribute*), 62
count (*pyUSPTO.models.patent_data.StatusCodeSearchResponse attribute*), 74
count (*pyUSPTO.models.ptab.PTABAppealResponse attribute*), 84
count (*pyUSPTO.models.ptab.PTABInterferenceResponse attribute*), 86
count (*pyUSPTO.models.ptab.PTABTrialDocumentResponse attribute*), 88
count (*pyUSPTO.models.ptab.PTABTrialProceedingResponse attribute*), 90
country_code (*pyUSPTO.models.patent_data.Address attribute*), 37, 38
country_code (*pyUSPTO.models.patent_data.Person attribute*), 68
country_name (*pyUSPTO.models.patent_data.Address attribute*), 37, 38
country_or_state_code (*pyUSPTO.models.patent_data.Address attribute*), 38
cpc_classification_bag (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 43
custom_headers (*pyUSPTO.http_config.HTTPConfig attribute*), 100, 101
customer_number (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 43
customer_number_correspondence_data (*pyUSPTO.models.patent_data.RecordAttorney attribute*), 70, 71
CustomerNumberCorrespondence (*class in pyUSPTO.models.patent_data*), 51

D

days_of_week_text (*pyUSPTO.models.bulk_data.BulkDataProduct attribute*), 34
decision_data (*pyUSPTO.models.ptab.PTABAppealDecision attribute*), 83
decision_data (*pyUSPTO.models.ptab.PTABTrialDocument attribute*), 87
decision_issue_date (*pyUSPTO.models.ptab.DecisionData attribute*), 78, 79
decision_issue_date (*pyUSPTO.models.ptab.InterferenceDocumentData attribute*), 80
decision_issue_date (*pyUSPTO.models.ptab.TrialDecisionData attribute*), 93
decision_type_category (*pyUSPTO.models.ptab.DecisionData attribute*), 79
decision_type_category (*pyUSPTO.models.ptab.InterferenceDocumentData attribute*), 80
decision_type_category (*pyUSPTO.models.ptab.TrialDecisionData attribute*), 93
DecisionData (*class in pyUSPTO.models.ptab*), 78
declaration_date (*pyUSPTO.models.ptab.InterferenceMetaData attribute*), 81, 82
DEFAULT_UNKNOWN_MESSAGE (*pyUSPTO.exceptions.USPTOApiError attribute*), 103
derivation_petitioner_data (*pyUSPTO.models.ptab.PTABTrialDocument attribute*), 87
derivation_petitioner_data (*pyUSPTO.models.ptab.PTABTrialProceeding attribute*), 89
DerivationPetitionerData (*class in pyUSPTO.models.ptab*), 79
description (*pyUSPTO.models.patent_data.StatusCode attribute*), 71
direction_category (*pyUSPTO.models.patent_data.Document attribute*), 53
DirectionCategory (*class in pyUSPTO.models.patent_data*), 52
docket_notice_mailed_date (*pyUSPTO.models.ptab.AppealMetaData attribute*), 77, 78
docket_number (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 41, 43
Document (*class in pyUSPTO.models.patent_data*), 52
document (*pyUSPTO.exceptions.FormatNotAvailableError attribute*), 102
document_category (*pyUSPTO.models.ptab.TrialDocumentData attribute*), 94, 95
document_code (*pyUSPTO.models.patent_data.Document attribute*), 53
document_code_description_text (*pyUSPTO.models.patent_data.Document attribute*), 53
document_data (*pyUSPTO.models.ptab.PTABAppealDecision attribute*), 83
document_data (*pyUSPTO.models.ptab.PTABInterferenceDecision attribute*), 85
document_data (*pyUSPTO.models.ptab.PTABTrialDocument attribute*), 87
document_filing_date (*pyUSPTO.models.ptab.AppealDocumentData attribute*), 76

document_filing_date (*pyUSPTO.models.ptab.InterferenceDocumentData attribute*), 80
document_filing_date (*pyUSPTO.models.ptab.TrialDocumentData attribute*), 94, 95
document_formats (*pyUSPTO.models.patent_data.Document attribute*), 53
document_identifier (*pyUSPTO.models.patent_data.Document attribute*), 53
document_identifier (*pyUSPTO.models.ptab.AppealDocumentData attribute*), 76
document_identifier (*pyUSPTO.models.ptab.InterferenceDocumentData attribute*), 80
document_identifier (*pyUSPTO.models.ptab.TrialDocumentData attribute*), 94, 95
document_name (*pyUSPTO.models.ptab.AppealDocumentData attribute*), 76, 77
document_name (*pyUSPTO.models.ptab.InterferenceDocumentData attribute*), 80, 81
document_name (*pyUSPTO.models.ptab.TrialDocumentData attribute*), 94, 95
document_number (*pyUSPTO.models.ptab.TrialDocumentData attribute*), 94, 95
document_ocr_text (*pyUSPTO.models.ptab.AppealDocumentData attribute*), 76, 77
document_ocr_text (*pyUSPTO.models.ptab.InterferenceDocumentData attribute*), 80, 81
document_ocr_text (*pyUSPTO.models.ptab.TrialDocumentData attribute*), 94, 95
document_size_quantity (*pyUSPTO.models.ptab.AppealDocumentData attribute*), 76, 77
document_size_quantity (*pyUSPTO.models.ptab.InterferenceDocumentData attribute*), 80, 81
document_size_quantity (*pyUSPTO.models.ptab.TrialDocumentData attribute*), 94, 95
document_status (*pyUSPTO.models.ptab.TrialDocumentData attribute*), 94
document_title_text (*pyUSPTO.models.ptab.InterferenceDocumentData attribute*), 80, 81
document_title_text (*pyUSPTO.models.ptab.TrialDocumentData attribute*), 94, 95
document_type_description_text (*pyUSPTO.models.ptab.AppealDocumentData attribute*), 76, 77
document_type_description_text (*pyUSPTO.models.ptab.TrialDocumentData attribute*), 94, 95
DocumentBag (*class in pyUSPTO.models.patent_data*), 55
DocumentFormat (*class in pyUSPTO.models.patent_data*), 57
DocumentMimeType (*class in pyUSPTO.models.patent_data*), 58
documents (*pyUSPTO.models.patent_data.DocumentBag attribute*), 55
documents (*pyUSPTO.models.patent_data.DocumentBag property*), 56
domestic_representative (*pyUSPTO.models.patent_data.Assignment attribute*), 47
download_appeal_archive() (*pyUSPTO.clients.ptab_appeals.PTABAppealsClient method*), 22
download_appeal_document() (*pyUSPTO.clients.ptab_appeals.PTABAppealsClient method*), 23
download_appeal_documents() (*pyUSPTO.clients.ptab_appeals.PTABAppealsClient method*), 23
download_archive() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 8
download_chunk_size (*pyUSPTO.http_config.HTTPConfig attribute*), 100, 101
download_decisions() (*pyUSPTO.clients.petition_decisions.FinalPetitionDecisionsClient method*), 18
download_document() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 9
download_file() (*pyUSPTO.clients.bulk_data.BulkDataClient method*), 5
download_interference_archive() (*pyUSPTO.clients.ptab_interferences.PTABInterferencesClient method*), 26
download_interference_document() (*pyUSPTO.clients.ptab_interferences.PTABInterferencesClient method*), 26
download_interference_documents() (*pyUSPTO.clients.ptab_interferences.PTABInterferencesClient method*), 26
download_petition_document() (*pyUSPTO.clients.petition_decisions.FinalPetitionDecisionsClient method*), 19
download_publication() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 9
download_trial_archive()

(*pyUSPTO.clients.ptab_trials.PTABTrialsClient method*), 29
download_trial_document() (*pyUSPTO.clients.ptab_trials.PTABTrialsClient method*), 29
download_trial_documents() (*pyUSPTO.clients.ptab_trials.PTABTrialsClient method*), 30
download_url (*pyUSPTO.models.patent_data.DocumentFormat attribute*), 57

E

earliest_publication_date (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 43
earliest_publication_number (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 44
effective_filing_date (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 44
ENDPOINTS (*pyUSPTO.clients.bulk_data.BulkDataClient attribute*), 5
ENDPOINTS (*pyUSPTO.clients.patent_data.PatentDataClient attribute*), 7
ENDPOINTS (*pyUSPTO.clients.petition_decisions.FinalPetitionDecisionsClient attribute*), 17
ENDPOINTS (*pyUSPTO.clients.ptab_appeals.PTABAppealsClient attribute*), 22
ENDPOINTS (*pyUSPTO.clients.ptab_interferences.PTABInterferencesClient attribute*), 25
ENDPOINTS (*pyUSPTO.clients.ptab_trials.PTABTrialsClient attribute*), 29
entity_status_data (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 41, 44
EntityStatus (*class in pyUSPTO.models.patent_data*), 58
error_details (*pyUSPTO.exceptions.APIErrorArgs attribute*), 102
event_code (*pyUSPTO.models.patent_data.EventData attribute*), 59
event_data_bag (*pyUSPTO.models.patent_data.PatentFileWrapper attribute*), 64
event_date (*pyUSPTO.models.patent_data.EventData attribute*), 59
event_date (*pyUSPTO.models.patent_data.PatentTermAdjustmentHistoryData attribute*), 67
event_description_text (*pyUSPTO.models.patent_data.EventData attribute*), 59
event_description_text (*pyUSPTO.models.patent_data.PatentTermAdjustmentHistoryData attribute*), 67
event_sequence_number (*pyUSPTO.models.patent_data.PatentTermAdjustmentHistoryData attribute*), 67
EventData (*class in pyUSPTO.models.patent_data*), 59
examiner_name_text (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 43, 44
execution_date (*pyUSPTO.models.patent_data.Assignor attribute*), 48
extension_number (*pyUSPTO.models.patent_data.TelecommunicationNumber attribute*), 74, 75

F

FALSE (*pyUSPTO.models.patent_data.ActiveIndicator attribute*), 37
file_create_date_time (*pyUSPTO.models.patent_data.PrintedMetaData attribute*), 69
file_data_bag (*pyUSPTO.models.bulk_data.ProductFileBag attribute*), 36
file_data_from_date (*pyUSPTO.models.bulk_data.FileData attribute*), 36
file_data_to_date (*pyUSPTO.models.bulk_data.FileData attribute*), 36
file_date (*pyUSPTO.models.bulk_data.FileData attribute*), 36
file_download_uri (*pyUSPTO.models.bulk_data.FileData attribute*), 36
file_download_uri (*pyUSPTO.models.ptab.AppealDocumentData attribute*), 76, 77
file_download_uri (*pyUSPTO.models.ptab.AppealMetaData attribute*), 77, 78
file_download_uri (*pyUSPTO.models.ptab.InterferenceDocumentData attribute*), 80, 81
file_download_uri (*pyUSPTO.models.ptab.InterferenceMetaData attribute*), 81, 82
file_download_uri (*pyUSPTO.models.ptab.TrialDocumentData attribute*), 94, 95
file_download_uri (*pyUSPTO.models.ptab.TrialMetaData attribute*), 96
file_last_modified_date_time (*pyUSPTO.models.bulk_data.FileData attribute*), 36
file_location_uri (*pyUSPTO.models.patent_data.PrintedMetaData attribute*), 69
file_name (*pyUSPTO.models.bulk_data.FileData attribute*), 36
file_release_date (*pyUSPTO.models.bulk_data.FileData attribute*), 36
file_size (*pyUSPTO.models.bulk_data.FileData attribute*), 36
file_type_text (*pyUSPTO.models.bulk_data.FileData attribute*), 36
FileData (*class in pyUSPTO.models.bulk_data*), 35

filing_date (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 44
filing_date (*pyUSPTO.models.patent_data.Continuity attribute*), 51
filing_date (*pyUSPTO.models.patent_data.ForeignPriority attribute*), 60
filing_party_category (*pyUSPTO.models.ptab.TrialDocumentData attribute*), 94, 95
filter_by_format() (*pyUSPTO.models.patent_data.DocumentBag method*), 56
FinalPetitionDecisionsClient (*class in pyUSPTO.clients.petition_decisions*), 17
find_by_code() (*pyUSPTO.models.patent_data.StatusCodeCollection method*), 73
first_applicant_name (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 44
first_inventor_name (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 44
first_inventor_to_file_indicator (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 41, 44
first_inventor_to_file_indicator (*pyUSPTO.models.patent_data.Continuity attribute*), 50, 51
first_name (*pyUSPTO.models.patent_data.Person attribute*), 68
foreign_priority_bag (*pyUSPTO.models.patent_data.PatentFileWrapper attribute*), 64
ForeignPriority (*class in pyUSPTO.models.patent_data*), 59
FormatNotAvailableError, 102
frame_number (*pyUSPTO.models.patent_data.Assignment attribute*), 46, 47
from_dict() (*pyUSPTO.models.bulk_data.BulkDataProduct class method*), 34
from_dict() (*pyUSPTO.models.bulk_data.BulkDataResponse class method*), 35
from_dict() (*pyUSPTO.models.bulk_data.FileData class method*), 36
from_dict() (*pyUSPTO.models.bulk_data.ProductFileBag class method*), 36
from_dict() (*pyUSPTO.models.patent_data.Address class method*), 38
from_dict() (*pyUSPTO.models.patent_data.Applicant class method*), 39
from_dict() (*pyUSPTO.models.patent_data.ApplicationMetaData class method*), 44
from_dict() (*pyUSPTO.models.patent_data.Assignee class method*), 45
from_dict() (*pyUSPTO.models.patent_data.Assignment class method*), 47
from_dict() (*pyUSPTO.models.patent_data.Assignor class method*), 48
from_dict() (*pyUSPTO.models.patent_data.Attorney class method*), 49
from_dict() (*pyUSPTO.models.patent_data.ChildContinuity class method*), 50
from_dict() (*pyUSPTO.models.patent_data.CustomerNumberCorrespond class method*), 52
from_dict() (*pyUSPTO.models.patent_data.Document class method*), 54
from_dict() (*pyUSPTO.models.patent_data.DocumentBag class method*), 56
from_dict() (*pyUSPTO.models.patent_data.DocumentFormat class method*), 57
from_dict() (*pyUSPTO.models.patent_data.EntityStatus class method*), 58
from_dict() (*pyUSPTO.models.patent_data.EventData class method*), 59
from_dict() (*pyUSPTO.models.patent_data.ForeignPriority class method*), 60
from_dict() (*pyUSPTO.models.patent_data.Inventor class method*), 60
from_dict() (*pyUSPTO.models.patent_data.ParentContinuity class method*), 61
from_dict() (*pyUSPTO.models.patent_data.PatentDataResponse class method*), 62
from_dict() (*pyUSPTO.models.patent_data.PatentFileWrapper class method*), 64
from_dict() (*pyUSPTO.models.patent_data.PatentTermAdjustmentData class method*), 66
from_dict() (*pyUSPTO.models.patent_data.PatentTermAdjustmentHistor class method*), 67
from_dict() (*pyUSPTO.models.patent_data.PrintedMetaData class method*), 69
from_dict() (*pyUSPTO.models.patent_data.RecordAttorney class method*), 71
from_dict() (*pyUSPTO.models.patent_data.StatusCode class method*), 71
from_dict() (*pyUSPTO.models.patent_data.StatusCodeSearchResponse class method*), 74
from_dict() (*pyUSPTO.models.patent_data.Telecommunication class method*), 75
from_dict() (*pyUSPTO.models.ptab.AdditionalPartyData class method*), 75
from_dict() (*pyUSPTO.models.ptab.AppealDocumentData class method*), 77
from_dict() (*pyUSPTO.models.ptab.AppealMetaData class method*), 78
from_dict() (*pyUSPTO.models.ptab.DecisionData class method*), 79
from_dict() (*pyUSPTO.models.ptab.InterferenceDocumentData class method*), 81
from_dict() (*pyUSPTO.models.ptab.InterferenceMetaData*

class method), 82
from_dict() (*pyUSPTO.models.ptab.PTABAppealDecision class method*), 83
from_dict() (*pyUSPTO.models.ptab.PTABAppealResponse class method*), 84
from_dict() (*pyUSPTO.models.ptab.PTABInterferenceDecision class method*), 85
from_dict() (*pyUSPTO.models.ptab.PTABInterferenceResponse class method*), 86
from_dict() (*pyUSPTO.models.ptab.PTABTrialDocument class method*), 88
from_dict() (*pyUSPTO.models.ptab.PTABTrialDocumentResponse class method*), 88
from_dict() (*pyUSPTO.models.ptab.PTABTrialProceeding class method*), 89
from_dict() (*pyUSPTO.models.ptab.PTABTrialProceedingResponse class method*), 90
from_dict() (*pyUSPTO.models.ptab.RegularPetitionerData class method*), 91
from_dict() (*pyUSPTO.models.ptab.RequestorData class method*), 92
from_dict() (*pyUSPTO.models.ptab.TrialDecisionData class method*), 93
from_dict() (*pyUSPTO.models.ptab.TrialDocumentData class method*), 95
from_dict() (*pyUSPTO.models.ptab.TrialMetaData class method*), 96
from_env() (*pyUSPTO.config.USPTOConfig class method*), 100
from_env() (*pyUSPTO.http_config.HTTPConfig class method*), 101
from_http_error() (*pyUSPTO.exceptions.APIErrorArgs class method*), 102
from_request_exception() (*pyUSPTO.exceptions.APIErrorArgs class method*), 102
from_wrapper() (*pyUSPTO.models.patent_data.ApplicationContinuityData class method*), 40
from_wrapper() (*pyUSPTO.models.patent_data.PrintedPublication class method*), 70

G

geographic_region_code (*pyUSPTO.models.patent_data.Address attribute*), 37, 38
geographic_region_name (*pyUSPTO.models.patent_data.Address attribute*), 37, 38
get_api_exception() (*in module pyUSPTO.exceptions*), 104
get_application_adjustment() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 11
get_application_assignment() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 11
get_application_associated_documents() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 12
get_application_attorney() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 12
get_application_by_number() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 12
get_application_continuity() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 13
get_application_documents() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 13
get_application_foreign_priority() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 14
get_application_metadata() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 14
get_application_transactions() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 14
get_decision_by_id() (*pyUSPTO.clients.petition_decisions.FinalPetitionDecisionsClien method*), 20
get_format() (*pyUSPTO.models.patent_data.Document method*), 54
get_IFW_metadata() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 10
get_product_by_id() (*pyUSPTO.clients.bulk_data.BulkDataClient method*), 5
get_products() (*pyUSPTO.clients.bulk_data.BulkDataClient method*), 6
get_search_results() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 15
get_status_codes() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 15
get_timeout_tuple() (*pyUSPTO.http_config.HTTPConfig method*), 101
grant_date (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 44
grant_document_meta_data (*pyUSPTO.models.patent_data.PatentFileWrapper attribute*), 64, 65
grant_document_meta_data (*pyUSPTO.models.patent_data.PrintedPublication attribute*), 70

group_art_unit_number (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 44

H

has_format() (*pyUSPTO.models.patent_data.Document method*), 54
HTTPConfig (*class in pyUSPTO.http_config*), 100

I

ict_country_code (*pyUSPTO.models.patent_data.Address attribute*), 38
ict_state_code (*pyUSPTO.models.patent_data.Address attribute*), 38
image_available_status_code (*pyUSPTO.models.patent_data.Assignment attribute*), 46, 47
INCOMING (*pyUSPTO.models.patent_data.DirectionCategory attribute*), 52
institution_decision_date (*pyUSPTO.models.ptab.TrialMetaData attribute*), 96
interference_last_modified_date (*pyUSPTO.models.ptab.InterferenceMetaData attribute*), 81, 82
interference_last_modified_date_time (*pyUSPTO.models.ptab.InterferenceMetaData attribute*), 81, 82
interference_meta_data (*pyUSPTO.models.ptab.PTABInterferenceDecision attribute*), 85
interference_number (*pyUSPTO.models.ptab.PTABInterferenceDecision attribute*), 85
interference_outcome_category (*pyUSPTO.models.ptab.InterferenceDocumentData attribute*), 80, 81
interference_style_name (*pyUSPTO.models.ptab.InterferenceMetaData attribute*), 81, 82
InterferenceDocumentData (*class in pyUSPTO.models.ptab*), 79
InterferenceMetaData (*class in pyUSPTO.models.ptab*), 81
international_registration_number (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 43, 44
international_registration_publication_date (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 43, 44
invention_title (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 44
Inventor (*class in pyUSPTO.models.patent_data*), 60
inventor_bag (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 43, 44
inventor_name (*pyUSPTO.models.ptab.AdditionalPartyData attribute*), 75, 76
inventor_name_text (*pyUSPTO.models.patent_data.Inventor attribute*), 60, 61
ip_office_adjustment_delay_quantity (*pyUSPTO.models.patent_data.PatentTermAdjustmentData attribute*), 66
ip_office_day_delay_quantity (*pyUSPTO.models.patent_data.PatentTermAdjustmentHistoryDat attribute*), 67
ip_office_name (*pyUSPTO.models.patent_data.ForeignPriority attribute*), 59, 60
is_aia (*pyUSPTO.models.patent_data.ApplicationMetaData property*), 44
is_aia (*pyUSPTO.models.patent_data.Continuity property*), 51
is_pre_aia (*pyUSPTO.models.patent_data.ApplicationMetaData property*), 44
is_pre_aia (*pyUSPTO.models.patent_data.Continuity property*), 51
issue_type_bag (*pyUSPTO.models.ptab.DecisionData attribute*), 79
issue_type_bag (*pyUSPTO.models.ptab.InterferenceDocumentData attribute*), 80, 81
issue_type_bag (*pyUSPTO.models.ptab.TrialDecisionData attribute*), 93

J

junior_party_data (*pyUSPTO.models.ptab.PTABInterferenceDecision attribute*), 85
JuniorPartyData (*class in pyUSPTO.models.ptab*), 82

L

last_ingestion_date_time (*pyUSPTO.models.patent_data.PatentFileWrapper attribute*), 64, 65
last_modified_date_time (*pyUSPTO.models.bulk_data.BulkDataProduct attribute*), 34
last_modified_date_time (*pyUSPTO.models.ptab.PTABAppealDecision attribute*), 82, 83
last_modified_date_time (*pyUSPTO.models.ptab.PTABInterferenceDecision attribute*), 85
last_modified_date_time (*pyUSPTO.models.ptab.PTABTrialDocument attribute*), 87, 88
last_modified_date_time (*pyUSPTO.models.ptab.PTABTrialProceeding attribute*), 89, 90

last_name (*pyUSPTO.models.patent_data.Person attribute*), 68
latest_decision_date (*pyUSPTO.models.ptab.TrialMetaData attribute*), 96

M

max_retries (*pyUSPTO.http_config.HTTPConfig attribute*), 100, 101
message (*pyUSPTO.exceptions.APIErrorArgs attribute*), 102
message (*pyUSPTO.exceptions.USPTOApiError property*), 103
middle_name (*pyUSPTO.models.patent_data.Person attribute*), 68
mime_type_identifier (*pyUSPTO.models.patent_data.DocumentFormat attribute*), 57, 58
mime_type_identifier (*pyUSPTO.models.ptab.TrialDocumentData attribute*), 94
mime_type_identifier_array_text (*pyUSPTO.models.bulk_data.BulkDataProduct attribute*), 34
module
    pyUSPTO.clients.bulk_data, 5
    pyUSPTO.clients.patent_data, 7
    pyUSPTO.clients.petition_decisions, 17
    pyUSPTO.clients.ptab_appeals, 22
    pyUSPTO.clients.ptab_interferences, 25
    pyUSPTO.clients.ptab_trials, 29
    pyUSPTO.config, 99
    pyUSPTO.exceptions, 101
    pyUSPTO.http_config, 100
    pyUSPTO.models.bulk_data, 34
    pyUSPTO.models.patent_data, 36
    pyUSPTO.models.ptab, 75
    pyUSPTO.models.utils, 97
    pyUSPTO.warnings, 105
MS_WORD (*pyUSPTO.models.patent_data.DocumentMimeType attribute*), 58

N

name_line_one_text (*pyUSPTO.models.patent_data.Address attribute*), 37, 38
name_line_two_text (*pyUSPTO.models.patent_data.Address attribute*), 37, 38
name_prefix (*pyUSPTO.models.patent_data.Person attribute*), 68
name_suffix (*pyUSPTO.models.patent_data.Person attribute*), 68
national_stage_indicator (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 41, 44
NO (*pyUSPTO.models.patent_data.ActiveIndicator attribute*), 37
non_overlapping_day_delay_quantity (*pyUSPTO.models.patent_data.PatentTermAdjustmentData attribute*), 66
non_overlapping_day_quantity (*pyUSPTO.models.patent_data.PatentTermAdjustmentData attribute*), 65, 66

O

official_date (*pyUSPTO.models.patent_data.Document attribute*), 53, 54
organization_standard_name (*pyUSPTO.models.patent_data.CustomerNumberCorrespondence attribute*), 52
originating_event_sequence_number (*pyUSPTO.models.patent_data.PatentTermAdjustmentHistoryDat attribute*), 67
OUTGOING (*pyUSPTO.models.patent_data.DirectionCategory attribute*), 52
overlapping_day_quantity (*pyUSPTO.models.patent_data.PatentTermAdjustmentData attribute*), 66

P

page_total_quantity (*pyUSPTO.models.patent_data.Assignment attribute*), 46, 47
page_total_quantity (*pyUSPTO.models.patent_data.DocumentFormat attribute*), 57, 58
paginate_applications() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 15
paginate_decisions() (*pyUSPTO.clients.petition_decisions.FinalPetitionDecisionsClien method*), 20
paginate_decisions() (*pyUSPTO.clients.ptab_appeals.PTABAppealsClient method*), 23
paginate_decisions() (*pyUSPTO.clients.ptab_interferences.PTABInterferencesClient method*), 27
paginate_proceedings() (*pyUSPTO.clients.ptab_trials.PTABTrialsClient method*), 30
paginate_products() (*pyUSPTO.clients.bulk_data.BulkDataClient method*), 6
parent_application_filing_date (*pyUSPTO.models.patent_data.ParentContinuity attribute*), 61, 62
parent_application_number_text (*pyUSPTO.models.patent_data.ChildContinuity*

attribute), 49, 50
parent_application_number_text (*pyUSPTO.models.patent_data.ParentContinuity attribute*), 61, 62
parent_application_status_code (*pyUSPTO.models.patent_data.ParentContinuity attribute*), 61, 62
parent_application_status_description_text (*pyUSPTO.models.patent_data.ParentContinuity attribute*), 61, 62
parent_continuity_bag (*pyUSPTO.models.patent_data.ApplicationContinuityData attribute*), 40
parent_continuity_bag (*pyUSPTO.models.patent_data.PatentFileWrapper attribute*), 64, 65
parent_patent_number (*pyUSPTO.models.patent_data.ParentContinuity attribute*), 61, 62
ParentContinuity (*class in pyUSPTO.models.patent_data*), 61
parse_to_date() (*in module pyUSPTO.models.utils*), 97
parse_to_datetime_utc() (*in module pyUSPTO.models.utils*), 97
parse_yn_to_bool() (*in module pyUSPTO.models.utils*), 97
patent_appeal_data_bag (*pyUSPTO.models.ptab.PTABAppealResponse attribute*), 84
patent_file_wrapper_data_bag (*pyUSPTO.models.patent_data.PatentDataResponse attribute*), 62, 63
patent_interference_data_bag (*pyUSPTO.models.ptab.PTABInterferenceResponse attribute*), 86
patent_number (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 44
patent_number (*pyUSPTO.models.patent_data.Continuity attribute*), 51
patent_number (*pyUSPTO.models.ptab.AdditionalPartyData attribute*), 75, 76
patent_owner_data (*pyUSPTO.models.ptab.PTABTrialDocument attribute*), 87, 88
patent_owner_data (*pyUSPTO.models.ptab.PTABTrialProceeding attribute*), 89, 90
patent_term_adjustment_data (*pyUSPTO.models.patent_data.PatentFileWrapper attribute*), 64, 65
patent_term_adjustment_history_data_bag (*pyUSPTO.models.patent_data.PatentTermAdjustmentData attribute*), 66
patent_trial_document_data_bag (*pyUSPTO.models.ptab.PTABTrialDocumentResponse attribute*), 89
patent_trial_proceeding_data_bag (*pyUSPTO.models.ptab.PTABTrialProceedingResponse attribute*), 90, 91
PatentDataClient (*class in pyUSPTO.clients.patent_data*), 7
PatentDataResponse (*class in pyUSPTO.models.patent_data*), 62
PatentFileWrapper (*class in pyUSPTO.models.patent_data*), 63
PatentOwnerData (*class in pyUSPTO.models.ptab*), 91
PatentTermAdjustmentData (*class in pyUSPTO.models.patent_data*), 65
PatentTermAdjustmentHistoryData (*class in pyUSPTO.models.patent_data*), 66
patron_identifier (*pyUSPTO.models.patent_data.CustomerNumberCo attribute*), 52
pct_publication_date (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 44
pct_publication_number (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 42, 44
PDF (*pyUSPTO.models.patent_data.DocumentMimeType attribute*), 58
Person (*class in pyUSPTO.models.patent_data*), 68
petition_filing_date (*pyUSPTO.models.ptab.TrialMetaData attribute*), 95, 96
pgpub_document_meta_data (*pyUSPTO.models.patent_data.PatentFileWrapper attribute*), 64, 65
pgpub_document_meta_data (*pyUSPTO.models.patent_data.PrintedPublication attribute*), 70
pool_connections (*pyUSPTO.http_config.HTTPConfig attribute*), 100, 101
pool_maxsize (*pyUSPTO.http_config.HTTPConfig attribute*), 100, 101
postal_address_category (*pyUSPTO.models.patent_data.Address attribute*), 38
postal_code (*pyUSPTO.models.patent_data.Address attribute*), 37, 39
power_of_attorney_address_bag (*pyUSPTO.models.patent_data.CustomerNumberCorrespondence attribute*), 52
power_of_attorney_bag (*pyUSPTO.models.patent_data.RecordAttorney attribute*), 71
preferred_name (*pyUSPTO.models.patent_data.Person attribute*), 68
PrintedMetaData (*class in pyUSPTO.models.patent_data*), 69

PrintedPublication (*class in pyUSPTO.models.patent_data*), 70
product_dataset_array_text (*pyUSPTO.models.bulk_data.BulkDataProduct attribute*), 34
product_dataset_category_array_text (*pyUSPTO.models.bulk_data.BulkDataProduct attribute*), 34
product_description_text (*pyUSPTO.models.bulk_data.BulkDataProduct attribute*), 34
product_file_bag (*pyUSPTO.models.bulk_data.BulkDataProduct attribute*), 34
product_file_total_quantity (*pyUSPTO.models.bulk_data.BulkDataProduct attribute*), 35
product_frequency_text (*pyUSPTO.models.bulk_data.BulkDataProduct attribute*), 35
product_from_date (*pyUSPTO.models.bulk_data.BulkDataProduct attribute*), 35
product_identifier (*pyUSPTO.models.bulk_data.BulkDataProduct attribute*), 35
product_identifier (*pyUSPTO.models.patent_data.PrintedMetaData attribute*), 69
product_label_array_text (*pyUSPTO.models.bulk_data.BulkDataProduct attribute*), 35
product_title_text (*pyUSPTO.models.bulk_data.BulkDataProduct attribute*), 35
product_to_date (*pyUSPTO.models.bulk_data.BulkDataProduct attribute*), 35
product_total_file_size (*pyUSPTO.models.bulk_data.BulkDataProduct attribute*), 35
ProductFileBag (*class in pyUSPTO.models.bulk_data*), 36
pta_pte_code (*pyUSPTO.models.patent_data.PatentTermAdjustmentHistoryData attribute*), 67
PTABAppealDecision (*class in pyUSPTO.models.ptab*), 82
PTABAppealResponse (*class in pyUSPTO.models.ptab*), 84
PTABAppealsClient (*class in pyUSPTO.clients.ptab_appeals*), 22
PTABInterferenceDecision (*class in pyUSPTO.models.ptab*), 84
PTABInterferenceResponse (*class in pyUSPTO.models.ptab*), 86
PTABInterferencesClient (*class in pyUSPTO.clients.ptab_interferences*), 25
PTABTrialDocument (*class in pyUSPTO.models.ptab*), 87
PTABTrialDocumentResponse (*class in pyUSPTO.models.ptab*), 88
PTABTrialProceeding (*class in pyUSPTO.models.ptab*), 89
PTABTrialProceedingResponse (*class in pyUSPTO.models.ptab*), 90
PTABTrialsClient (*class in pyUSPTO.clients.ptab_trials*), 29
publication_category_bag (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 41, 44
publication_date_bag (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 41, 44
publication_sequence_number_bag (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 41, 44
pyUSPTO.clients.bulk_data module, 5
pyUSPTO.clients.patent_data module, 7
pyUSPTO.clients.petition_decisions module, 17
pyUSPTO.clients.ptab_appeals module, 22
pyUSPTO.clients.ptab_interferences module, 25
pyUSPTO.clients.ptab_trials module, 29
pyUSPTO.config module, 99
pyUSPTO.exceptions module, 101
pyUSPTO.http_config module, 100
pyUSPTO.models.bulk_data module, 34
pyUSPTO.models.patent_data module, 36
pyUSPTO.models.ptab module, 75
pyUSPTO.models.utils module, 97
pyUSPTO.warnings module, 105

R

raw_data (*pyUSPTO.models.bulk_data.BulkDataResponse attribute*), 35
raw_data (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 43, 45
raw_data (*pyUSPTO.models.patent_data.PatentDataResponse attribute*), 62, 63
raw_data (*pyUSPTO.models.ptab.PTABAppealDecision attribute*), 83

raw_data (*pyUSPTO.models.ptab.PTABAppealResponse attribute*), 84
raw_data (*pyUSPTO.models.ptab.PTABInterferenceDecision attribute*), 85, 86
raw_data (*pyUSPTO.models.ptab.PTABInterferenceResponse attribute*), 86
raw_data (*pyUSPTO.models.ptab.PTABTrialDocument attribute*), 87, 88
raw_data (*pyUSPTO.models.ptab.PTABTrialDocumentResponse attribute*), 89
raw_data (*pyUSPTO.models.ptab.PTABTrialProceeding attribute*), 89, 90
raw_data (*pyUSPTO.models.ptab.PTABTrialProceedingResponse attribute*), 90, 91
real_party_in_interest_name (*pyUSPTO.models.ptab.RegularPetitionerData attribute*), 91, 92
record_attorney (*pyUSPTO.models.patent_data.PatentFileWrapper attribute*), 64, 65
RecordAttorney (*class in pyUSPTO.models.patent_data*), 70
reel_and_frame_number (*pyUSPTO.models.patent_data.Assignment attribute*), 46, 47
reel_number (*pyUSPTO.models.patent_data.Assignment attribute*), 46, 47
registered_practitioner_category (*pyUSPTO.models.patent_data.Attorney attribute*), 48, 49
registration_number (*pyUSPTO.models.patent_data.Attorney attribute*), 48, 49
regular_petitioner_data (*pyUSPTO.models.ptab.PTABTrialDocument attribute*), 87, 88
regular_petitioner_data (*pyUSPTO.models.ptab.PTABTrialProceeding attribute*), 89, 90
RegularPetitionerData (*class in pyUSPTO.models.ptab*), 91
request_identifier (*pyUSPTO.exceptions.APIErrorArgs attribute*), 102
request_identifier (*pyUSPTO.models.patent_data.PatentDataResponse attribute*), 62, 63
request_identifier (*pyUSPTO.models.patent_data.StatusCodeSearchResponse attribute*), 74
request_identifier (*pyUSPTO.models.ptab.PTABAppealResponse attribute*), 84
request_identifier (*pyUSPTO.models.ptab.PTABInterferenceResponse attribute*), 86
request_identifier (*pyUSPTO.models.ptab.PTABTrialDocumentResponse attribute*), 89
request_identifier (*pyUSPTO.models.ptab.PTABTrialProceedingResponse attribute*), 90, 91
requested_format (*pyUSPTO.exceptions.FormatNotAvailableError attribute*), 102
requestor_data (*pyUSPTO.models.ptab.PTABAppealDecision attribute*), 83
RequestorData (*class in pyUSPTO.models.ptab*), 92
respondent_data (*pyUSPTO.models.ptab.PTABTrialDocument attribute*), 87, 88
respondent_data (*pyUSPTO.models.ptab.PTABTrialProceeding attribute*), 89, 90
RespondentData (*class in pyUSPTO.models.ptab*), 92
retry_status_codes (*pyUSPTO.http_config.HTTPConfig attribute*), 100, 101

S

sanitize_application_number() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 16
search_applications() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 17
search_by_description() (*pyUSPTO.models.patent_data.StatusCodeCollection method*), 73
search_decisions() (*pyUSPTO.clients.petition_decisions.FinalPetitionD method*), 21
search_decisions() (*pyUSPTO.clients.ptab_appeals.PTABAppealsClien method*), 24
search_decisions() (*pyUSPTO.clients.ptab_interferences.PTABInterfer method*), 27
search_decisions() (*pyUSPTO.clients.ptab_trials.PTABTrialsClient method*), 30
search_documents() (*pyUSPTO.clients.ptab_trials.PTABTrialsClient method*), 32
search_proceedings() (*pyUSPTO.clients.ptab_trials.PTABTrialsClient method*), 33
search_products() (*pyUSPTO.clients.bulk_data.BulkDataClient method*), 6
search_status_codes() (*pyUSPTO.clients.patent_data.PatentDataClient method*), 17
senior_party_data (*pyUSPTO.models.ptab.PTABInterferenceDecision attribute*), 85, 86
SeniorPartyData (*class in pyUSPTO.models.ptab*), 93
serialize_bool_to_yn() (*in module pyUSPTO.models.utils*), 98
serialize_date() (*in module pyUSPTO.models.utils*), 98
serialize_datetime_as_iso() (*in module pyUSPTO.models.utils*), 98
serialize_datetime_as_naive() (*in module pyUSPTO.models.utils*), 98
small_entity_status_indicator (*pyUSPTO.models.patent_data.EntityStatus*

attribute), 58
status_code (*pyUSPTO.exceptions.APIErrorArgs attribute*), 102
status_code (*pyUSPTO.models.patent_data.Continuity attribute*), 51
status_code_bag (*pyUSPTO.models.patent_data.StatusCodeSearchResponse attribute*), 74
status_codes (*pyUSPTO.models.patent_data.StatusCodeCollection attribute*), 72
status_description_text (*pyUSPTO.models.patent_data.Continuity attribute*), 51
StatusCode (*class in pyUSPTO.models.patent_data*), 71
StatusCodeCollection (*class in pyUSPTO.models.patent_data*), 72
StatusCodeSearchResponse (*class in pyUSPTO.models.patent_data*), 74
statute_and_rule_bag (*pyUSPTO.models.ptab.DecisionData attribute*), 78, 79
statute_and_rule_bag (*pyUSPTO.models.ptab.InterferenceDocumentData attribute*), 80, 81
statute_and_rule_bag (*pyUSPTO.models.ptab.TrialDecisionData attribute*), 93
subclass (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 43, 45

T

telecom_type_code (*pyUSPTO.models.patent_data.Telecommunication attribute*), 74, 75
Telecommunication (*class in pyUSPTO.models.patent_data*), 74
telecommunication_address_bag (*pyUSPTO.models.patent_data.Attorney attribute*), 48, 49
telecommunication_address_bag (*pyUSPTO.models.patent_data.CustomerNumberCorrespondence attribute*), 52
telecommunication_number (*pyUSPTO.models.patent_data.Telecommunication attribute*), 74, 75
termination_date (*pyUSPTO.models.ptab.TrialMetaData attribute*), 96
third_party_name (*pyUSPTO.models.ptab.RequestorData attribute*), 92
timeout (*pyUSPTO.http_config.HTTPConfig attribute*), 100, 101
to_camel_case() (*in module pyUSPTO.models.utils*), 99
to_csv() (*pyUSPTO.models.patent_data.PatentDataResponse method*), 63
to_dict() (*pyUSPTO.models.bulk_data.BulkDataResponse method*), 35
to_dict() (*pyUSPTO.models.patent_data.Address method*), 39
to_dict() (*pyUSPTO.models.patent_data.Applicant method*), 39
to_dict() (*pyUSPTO.models.patent_data.ApplicationContinuityData method*), 40
to_dict() (*pyUSPTO.models.patent_data.ApplicationMetaData method*), 45
to_dict() (*pyUSPTO.models.patent_data.Assignee method*), 45
to_dict() (*pyUSPTO.models.patent_data.Assignment method*), 47
to_dict() (*pyUSPTO.models.patent_data.Assignor method*), 48
to_dict() (*pyUSPTO.models.patent_data.Attorney method*), 49
to_dict() (*pyUSPTO.models.patent_data.ChildContinuity method*), 50
to_dict() (*pyUSPTO.models.patent_data.Continuity method*), 51
to_dict() (*pyUSPTO.models.patent_data.CustomerNumberCorresponden method*), 52
to_dict() (*pyUSPTO.models.patent_data.Document method*), 55
to_dict() (*pyUSPTO.models.patent_data.DocumentBag method*), 56
to_dict() (*pyUSPTO.models.patent_data.DocumentFormat method*), 58
to_dict() (*pyUSPTO.models.patent_data.EntityStatus method*), 59
to_dict() (*pyUSPTO.models.patent_data.EventData method*), 59
to_dict() (*pyUSPTO.models.patent_data.ForeignPriority method*), 60
to_dict() (*pyUSPTO.models.patent_data.Inventor method*), 61
to_dict() (*pyUSPTO.models.patent_data.ParentContinuity method*), 62
to_dict() (*pyUSPTO.models.patent_data.PatentDataResponse method*), 63
to_dict() (*pyUSPTO.models.patent_data.PatentFileWrapper method*), 65
to_dict() (*pyUSPTO.models.patent_data.PatentTermAdjustmentData method*), 66
to_dict() (*pyUSPTO.models.patent_data.PatentTermAdjustmentHistoryD method*), 68
to_dict() (*pyUSPTO.models.patent_data.Person method*), 68
to_dict() (*pyUSPTO.models.patent_data.PrintedMetaData method*), 69
to_dict() (*pyUSPTO.models.patent_data.PrintedPublication method*), 70

to_dict() (*pyUSPTO.models.patent_data.RecordAttorney method*), 71
to_dict() (*pyUSPTO.models.patent_data.StatusCode method*), 72
to_dict() (*pyUSPTO.models.patent_data.StatusCodeCollection method*), 73
to_dict() (*pyUSPTO.models.patent_data.StatusCodeSearchResponse method*), 74
to_dict() (*pyUSPTO.models.patent_data.Telecommunication method*), 75
to_dict() (*pyUSPTO.models.ptab.AdditionalPartyData method*), 76
to_dict() (*pyUSPTO.models.ptab.AppealDocumentData method*), 77
to_dict() (*pyUSPTO.models.ptab.AppealMetaData method*), 78
to_dict() (*pyUSPTO.models.ptab.DecisionData method*), 79
to_dict() (*pyUSPTO.models.ptab.InterferenceDocumentData method*), 81
to_dict() (*pyUSPTO.models.ptab.InterferenceMetaData method*), 82
to_dict() (*pyUSPTO.models.ptab.PTABAppealDecision method*), 83
to_dict() (*pyUSPTO.models.ptab.PTABAppealResponse method*), 84
to_dict() (*pyUSPTO.models.ptab.PTABInterferenceDecision method*), 86
to_dict() (*pyUSPTO.models.ptab.PTABInterferenceResponse method*), 86
to_dict() (*pyUSPTO.models.ptab.PTABTrialDocument method*), 88
to_dict() (*pyUSPTO.models.ptab.PTABTrialDocumentResponse method*), 89
to_dict() (*pyUSPTO.models.ptab.PTABTrialProceeding method*), 90
to_dict() (*pyUSPTO.models.ptab.PTABTrialProceedingResponse method*), 91
to_dict() (*pyUSPTO.models.ptab.RegularPetitionerData method*), 92
to_dict() (*pyUSPTO.models.ptab.RequestorData method*), 92
to_dict() (*pyUSPTO.models.ptab.TrialDecisionData method*), 94
to_dict() (*pyUSPTO.models.ptab.TrialDocumentData method*), 95
to_dict() (*pyUSPTO.models.ptab.TrialMetaData method*), 96
trial_document_category (*pyUSPTO.models.ptab.PTABTrialDocument attribute*), 87, 88
trial_last_modified_date (*pyUSPTO.models.ptab.TrialMetaData attribute*), 96
trial_last_modified_date_time (*pyUSPTO.models.ptab.TrialMetaData attribute*), 96, 97
trial_meta_data (*pyUSPTO.models.ptab.PTABTrialDocument attribute*), 87, 88
trial_meta_data (*pyUSPTO.models.ptab.PTABTrialProceeding attribute*), 89, 90
trial_number (*pyUSPTO.models.ptab.PTABTrialDocument attribute*), 87, 88
trial_number (*pyUSPTO.models.ptab.PTABTrialProceeding attribute*), 89, 90
trial_outcome_category (*pyUSPTO.models.ptab.TrialDecisionData attribute*), 93, 94
trial_record_identifier (*pyUSPTO.models.ptab.PTABTrialProceeding attribute*), 89
trial_status_category (*pyUSPTO.models.ptab.TrialMetaData attribute*), 96, 97
trial_type_code (*pyUSPTO.models.ptab.PTABTrialDocument attribute*), 87, 88
trial_type_code (*pyUSPTO.models.ptab.TrialMetaData attribute*), 96, 97
TrialDecisionData (*class in pyUSPTO.models.ptab*), 93
TrialDocumentData (*class in pyUSPTO.models.ptab*), 94
TrialMetaData (*class in pyUSPTO.models.ptab*), 95
TRUE (*pyUSPTO.models.patent_data.ActiveIndicator attribute*), 37

U

uspc_symbol_text (*pyUSPTO.models.patent_data.ApplicationMetaData attribute*), 43, 45
USPTOApiAuthError, 103
USPTOApiBadRequestError, 103
USPTOApiError, 103
USPTOApiNotFoundError, 104
USPTOApiPayloadTooLargeError, 104
USPTOApiRateLimitError, 104
USPTOApiServerError, 104
USPTOBooleanParseWarning, 105
USPTOConfig (*class in pyUSPTO.config*), 99
USPTOConnectionError, 104
USPTODataMismatchWarning, 105
USPTODataWarning, 105
USPTODateParseWarning, 105
USPTOEnumParseWarning, 105
USPTOTimeout, 104
USPTOTimezoneWarning, 105

X

XML (*pyUSPTO.models.patent_data.DocumentMimeType*

attribute), 58
xml_file_name (*pyUSPTO.models.patent_data.PrintedMetaData attribute*), 69

Y

YES (*pyUSPTO.models.patent_data.ActiveIndicator attribute*), 37

Z

zip_file_name (*pyUSPTO.models.patent_data.PrintedMetaData attribute*), 69, 70