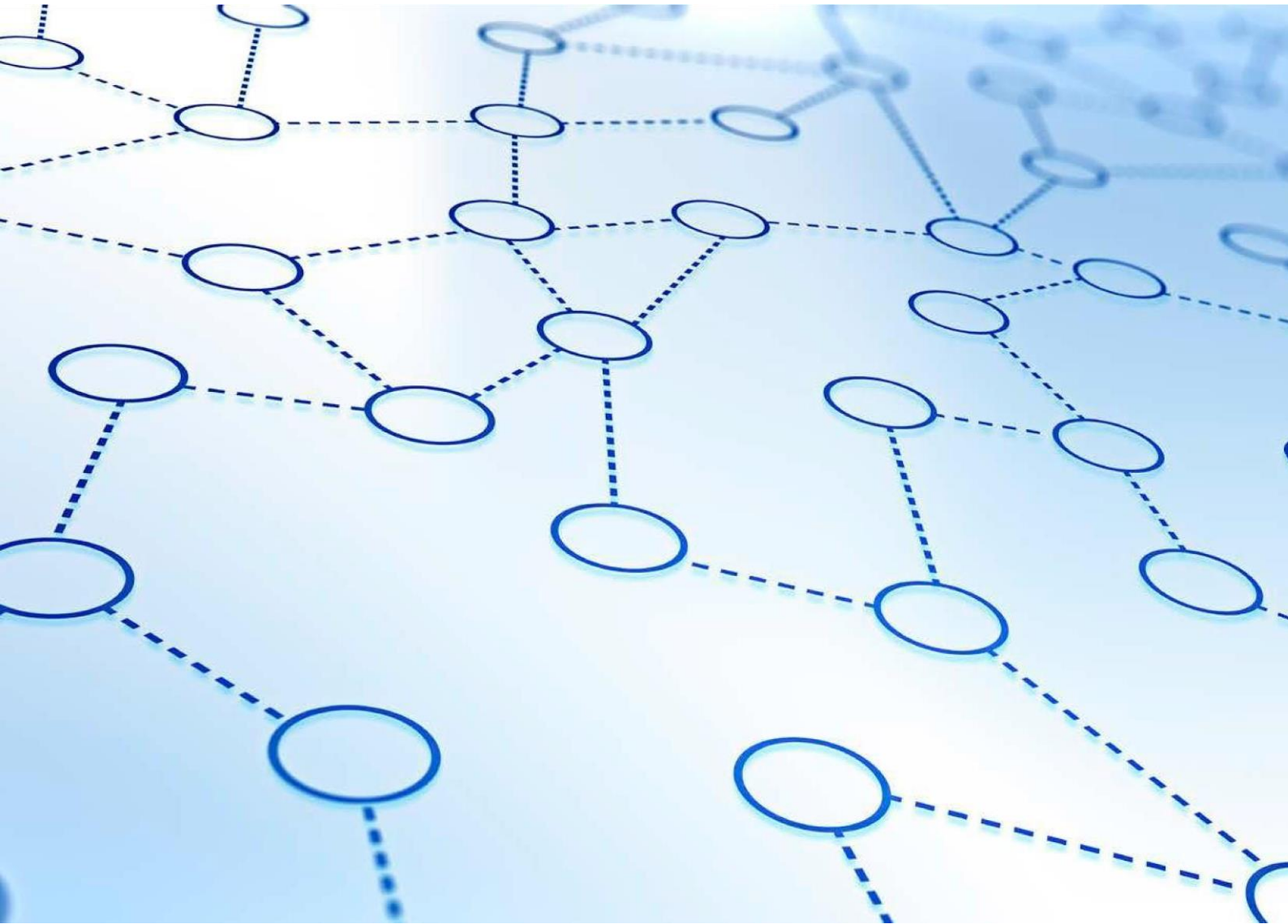
ELSEVIER

Reaxys®

Reaxys Data API

Getting Started Guide

Revision 2026-Rev02



Contents

Authentication

The Reaxys Data API uses the Bearer authentication to authorize API requests, thus an API client must send an access token in the Authorization HTTP header when making a request.

```
HTTP Header: Authorization: Bearer <access token>
```

The access token is generated using the OAuth2.0 client credential code flow. For requesting an access token a POST request must be send to https://access.identity.elsevier.com/realms/digital/protocol/openid-connect/token. The request body must contain the grant_type parameter set to the value client_credentials and the scope parameter set to the value ae-data-api. To authenticate the token requests client_id and client_secret parameters must be specified using the credentials you received as values. The request header Content-Type set to application/x-www-form-urlencoded and the header Accept set to application/json must be provided

```
Request method: POST
Request endpoint: https://access.identity.elsevier.com/realms/digital/protocol/openid-connect/token

Request headers:
Content-Type= application/x-www-form-urlencoded
Accept= application/json

Request body:
grant_type=client_credentials
&scope=ae-data-api
&client_id=xxxxxxxxxx
&client_secret=xxxxxxxxxx
```

An access token is valid for 3600 seconds. After the expiration of the access token a new token must be requested.

In python the following code can be used to request the access token using the client credentials.

```
import requests
from authlib.integrations.requests_client import OAuth2Session

client_id = 'YOUR_CLIENT_ID'
client_secret = 'YOUR_CLIENT_SECRET'
```

```
scope = 'openid ae-data-api'
token_endpoint = "https://access.identity.elsevier.com/realms/digital/protocol/openid-
connect/token "

client = OAuth2Session(client_id, client_secret, scope=scope)
token = client.fetch_token(token_endpoint)

access_token = token['access_token']

headers = {'Authorization': f'Bearer {access_token}'}
```

Technical Details

Design

Rest API, using OData, max page size, stateless

Rate Limits

Rate limits categorized in request types and apply to a given Client ID

"Non-Structure Search and Retrieval": 7500 requests per minute

"Synchronous Structure Search": 1500 requests per minute

"Synchronous Structure Search Retrieval": 7500 requests per minute

"Asynchronous Structure Search": 60 requests per minute, 1500 per hour

Introduction to OData

OData (Open Data Protocol) is a standardized protocol for consuming queryable and interoperable RESTful APIs. It supports URL-based querying capabilities over HTTP and can serve diverse client applications with varying data needs. At its core, OData simplifies data querying by supporting a set of built-in query options, making it easy to filter, project, and paginate large datasets over HTTP. The official description of the OData Protocol is accessible here: https://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part1-protocol.html

The root path for the Reaxys Data API is https://apigw.healthcare.elsevier.com/ls/rx-data-api/v1/data/ and needs to be added for the examples in the following sections.

Top-Level Resources

In an OData service, top-level resources represent the main entry points into your data. These resources correspond to entity sets, which are collections of entities (like database tables). Each entity set is exposed as a URL path at the root of the OData service.

The Reaxys Data API exposes the following top-level resources:

/Substances – the collection of Reaxys substance entities

/Citations – the collection of Reaxys document entities

/Reactions – the collection of Reaxys reaction entities

/Datapoints – the collection of Reaxys bioactivity entities

/Targets – the collection of Reaxys biological target entities

These top-level endpoints are the starting points for building queries using OData parameters. From here, you can apply filters, expand relationships, select specific fields, or paginate through results.

OData Metadata Schema

The OData metadata schema for the Reaxys Data API can be accessed at https://apigw.healthcare.elsevier.com/ls/rx-data-api/v1/data/$metadata. The OData metadata schema defines the structure and organization of the data exposed by the Reaxys Data API. It provides a machine-readable description of the data model, including the entities, complex types, properties, and relationships between entities. This schema is essential to understand the available data and how to interact with it. By defining the data model, the metadata schema ensures consistency and interoperability, allowing clients to dynamically discover and utilize the OData service's capabilities. In the context of the Reaxys API, the metadata schema outlines the various chemical entities and their relationships, enabling users to perform complex queries and retrieve structured data efficiently.

Below is a simplified representation of metadata definitions for Substances. Properties defined by the <Property> element are defining the core properties of the entity and will be by default included when fetching a substance. The element <NavigationProperty> reflects relationship between associated entities. These properties are not included by default when fetching a substance. Instead these properties must be explicitly requested using the $expand OData query parameter.

```
<EntityType Name="Substance">
  <Key>
    <PropertyRef Name="ReaxysRegistryNumber"/>
  </Key>
  <Property Name="ReaxysRegistryNumber" Type="Edm.Int64" Nullable="false"/>
  <Property Name="ChemicalNames" Type="Collection(Edm.String)" MaxLength="2147483647"/>
  <Property Name="InChiKey" Type="Edm.String" MaxLength="2147483647"/>
  <Property Name="MolecularWeight" Type="Edm.Decimal" Precision="20" Scale="10"/>
  <Property Name="MolecularFormula" Type="Edm.String" MaxLength="2147483647"/>
  <Property Name="CasRegistryNumbers" Type="Collection(Edm.String)" MaxLength="2147483647"/>
  <Property Name="SubstanceIdentificationFragments" Type="Collection(data.SubstanceIdentificationFragment)"/>
  <NavigationProperty Name="Datapoints" Type="Collection(data.Datapoint)"/>
  <NavigationProperty Name="Targets" Type="Collection(data.Target)"/>
  <NavigationProperty Name="Citations" Type="Collection(data.Citation)"/>
  <NavigationProperty Name="Reactions" Type="Collection(data.Reaction)"/>
  <NavigationProperty Name="Structures" Type="Collection(data.Structure)">
    <ReferentialConstraint Property="ReaxysRegistryNumber" ReferencedProperty="ReaxysRegistryNumber"/>
  </NavigationProperty>
  <NavigationProperty Name="MeltingPoints" Type="Collection(data.MeltingPoint)">
    <ReferentialConstraint Property="ReaxysRegistryNumber" ReferencedProperty="ReaxysRegistryNumber"/>
  </NavigationProperty>
```

```
</EntityType>
```

Navigation Properties and Entity Relationships

In OData, navigation properties represent relationships between entities - such as one-to-many or many-to-one. They allow you to navigate from one entity to related entities, leveraging the relationships defined in the data model. This is particularly useful for accessing related data without needing to perform multiple separate queries.

In the Reaxys API, navigation properties establish connections between the top-level resources substances, datapoints, citations, reactions, targets, but also between a top-level resource entity and associated properties like connecting substances with reported melting points. Understanding and using these relationships can help you retrieve comprehensive and interconnected data.

Common Relationships:

A Substance has multiple NMR spectra (one-to-many)

A Citation has multiple Substances (one-to-many)

A Datapoint may have one Target (one-to-zero)

Using $expand to Include Related Entities

When requesting an entity, associated entities connected through navigation properties are not automatically retrieved. Including these connected entities in the response can be done using the OData query parameter $expand. Here are some examples:

Navigate between top-level resources

Retrieve Substances and Their Related Datapoints

GET /Substances?$expand=Datapoints

Returns Substances with their related Datapoint entities included in the response.

Retrieve Citations and Their Related Substances

GET /Citations?$expand=Substances

Returns Citations with their related Substance entities included in the response.

Retrieve a reaction and its related substances and citations

GET /Reactions?$expand=Substances,Citations

Returns Reactions with their related Substances and Citations included in the response

Navigate entity property relationships

Retrieve Substances and Their chemical Structures

GET /Substances?$expand=Structures

Returns Substances with their chemical Structures included in the response.

Retrieve Citations and Their PatentBibliographies which include Patent Claims

GET /Citations?$expand=PatentBibliographies

Returns Citations with Patent Bibliographic Data included in the response for patents.

Retrieve Datapoints and Their Target MedChemIds including Uniprot IDs

GET /Datapoints?$expand=MedChemIds

Returns Datapoints with related identifiers for the biological target included in the response.

Combined and nested navigations

Retrieve Reactions, Their related Products, and the Product chemical Structures

GET /Reactions?$expand=Products($expand=Structures)

This returns Reactions with the Substances that act as Products and the chemical Structure for each Product.

Current $expand Limitations

When filtering a top-level resource by a property of another related top-level resource, then the $expand parameter can only be used to expand the top-level resource the filter property resides within. This means that if you need to expand multiple related entities, you may need to perform separate requests or adjust your query strategy accordingly.

Example Query (Currently Unsupported):

GET /Datapoints?$filter=Citations/any(c:c/CommonPatentNumber eq 'WO2023235490')&$expand=Substances

Note: This type of query currently returns an HTTP 500 Internal Server Error.

Entity Relationship Diagrams

The getting started material includes Entity Relationship Diagrams (ERDs) describing the Reaxys Data Structure and the relationship between the entities and how to navigate them. The sample diagram below describes the main features using the Citations resource as an example.

Each entity is described by a box and the relationships are indicated by arrows. For each entity Properties and Navigation Properties are defined. All Properties are included by default when requesting an entity and do not need to be actively requested using $expand, i.e. for a Citation properties like CitationNumberId, Authors, ImpactFactor, and CitationPatent are included by default in a response. CitationPatent is defined as complexType, meaning that CitationPatent is defined itself by multiple properties. The definition for these complexTypes are described in the corresponding boxes and → → the

relationship is indicated using the style arrow. No $expand is required for these relationships. Relationships indicated by the style arrow are navigation properties which require to be actively included in the response using the $expand query parameter.

Example:

The following query will retrieve the patent with the CitationNumberId 106173231 and include all default properties:

GET /Citations(106173231)

Response:

The response will contain all properties including complexTypes but not the Navigation Properties.


```
{
	"@odata.context": "$metadata#data.Citation",
	"PreferredPublicationYear": 2019,
	"Pui": -21387257,
	"DocumentType": "Patent",
	"CommonPatentNumber": null,
	"CitationNumberId": 106173231,
	"DocumentQuality": "1",
	"JournalPublicationStatus": null,
	"CitationPatent": [
		{
			"PatentLanguageCode": "English",
			"PatentCountryCode": "CN",
			"PatentNumber": "CN110199008",
			"PatentYear": 2019,
			"PatentKindCode": "A",
			"CitationPatentId": 1
		}
	],
	"RegistrationKey": "CN_110199008_A",
	"AbstractPresence": "Y",
	"PatentAssignee": "JNC CORPORAITON;JNC PETROCHEMICAL CORPORAITON",
	"BioactivityPresence": null,
	"Pages": null,
	"ImpactFactor": null,
	"IndexingStatus": "automatic",
	"CitationJournal": [],
	"CitationAvailability": "CI_1,CJ_1,CL_1,CK_1,ID_7",
	"DocumentTitle": null,
	"CitationEntryDate": "2021/11/21",
	"Authors": null,
	"CitationUpdateDate": "2024/06/27"
}
```


The following query will retrieve the patent with the CitationNumberId 106173231 and include all default properties and the requested Keywords.

GET /Citations(106173231)?$expand=Keywords

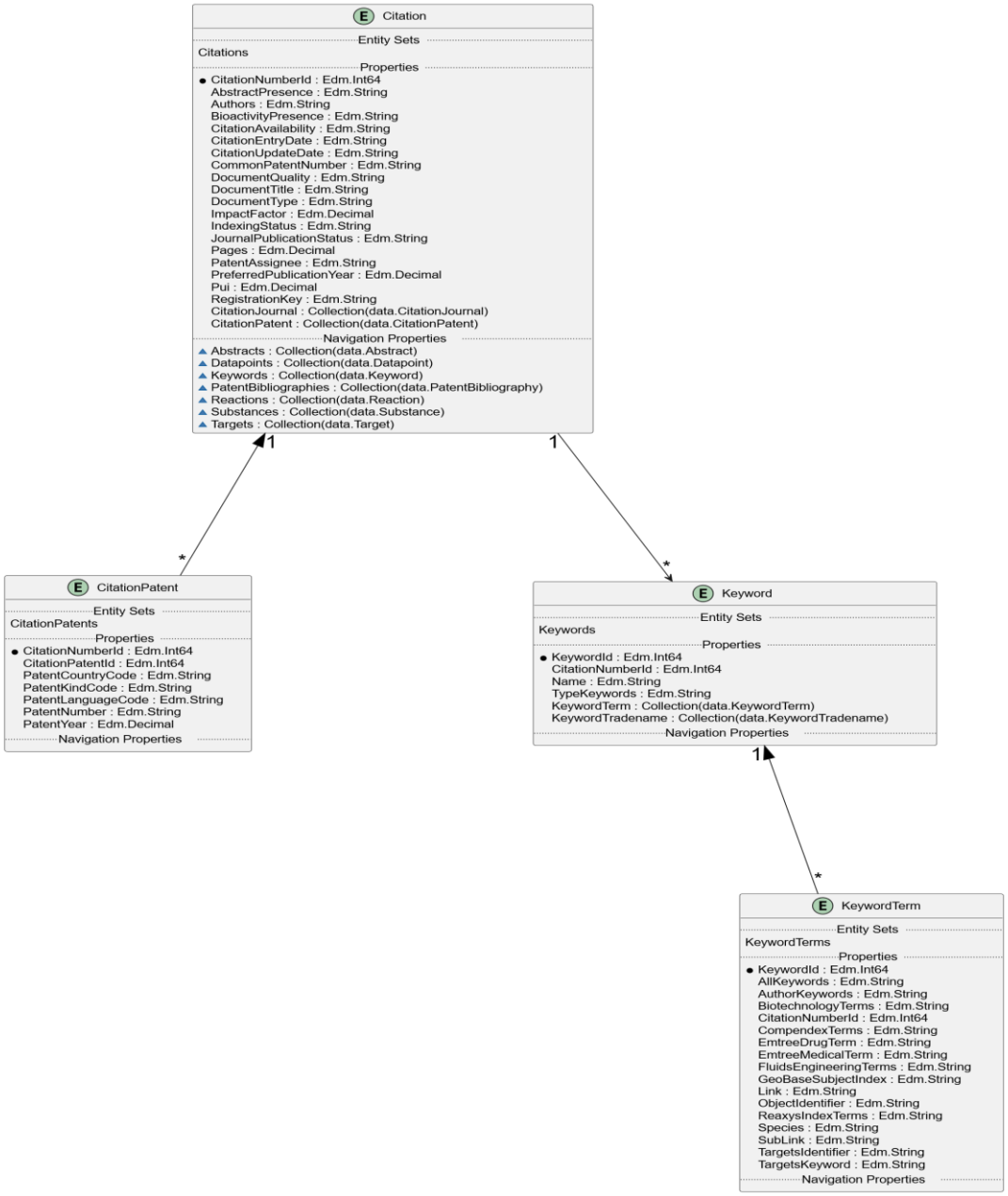
Response:

The response will contain all properties including complexTypes and the requested Keywords Navigation Property. The Keywords will include the complexType Property KeywordTerm by default.


```
{
```

```
"@odata.context": "$metadata#data.Citation",
"PreferredPublicationYear": 2019,
"Pui": -21387257,
"DocumentType": "Patent",
"CommonPatentNumber": null,
"CitationNumberId": 106173231,
"DocumentQuality": "1",
"JournalPublicationStatus": null,
"CitationPatent": [
        {
                "PatentLanguageCode": "English",
                "PatentCountryCode": "CN",
                "PatentNumber": "CN110199008",
                "PatentYear": 2019,
                "PatentKindCode": "A",
                "CitationPatentId": 1
        }
],
"RegistrationKey": "CN_110199008_A",
"AbstractPresence": "Y",
"PatentAssignee": "JNC CORPORAITON;JNC PETROCHEMICAL CORPORAITON",
"BioactivityPresence": null,
"Pages": null,
"ImpactFactor": null,
"IndexingStatus": "automatic",
"CitationJournal": [],
"CitationAvailability": "CI_1,CJ_1,CL_1,CK_1,ID_7",
"DocumentTitle": null,
"CitationEntryDate": "2021/11/21",
"Authors": null,
"CitationUpdateDate": "2024/06/27",
"Keywords": [
        {
                "KeywordId": 1,
                "TypeKeywords": "RXK",
                "KeywordTradename": [],
                "Name": "Reaxys Index Terms",
                "CitationNumberId": 106173231,
                "KeywordTerm": [
                        {
                                "AllKeywords": null,
                                "KeywordTermId": 1,
                                "BiotechnologyTerms": null,
                                "SubLink": null,
                                "GeoBaseSubjectIndex": null,
                                "Link": null,
                                "EmtreeMedicalTerm": null,
                                "CompendexTerms": null,
                                "TargetsKeyword": null,
                                "Species": null,
                                "EmtreeDrugTerm": null,
                                "AuthorKeywords": null,
                                "FluidsEngineeringTerms": null,
                                "ObjectIdentifier": "817100020425",
                                "TargetsIdentifier": null,
                                "ReaxysIndexTerms": "absorbent"
                        }
                ]
        }
]

}
```



Core Query Options

Below is an introduction to some of the most commonly used OData query parameters: $filter, $expand, $select, $count, $top, and $skip. These options allow you to filter, limit, and paginate data.

$filter: Filters the results based on a Boolean condition.

Example: Retrieve substances with a molecular weight greater than 300.

GET /Substances?$filter=MolecularWeight gt 300

$expand: Expands related entities inline with the main entity.

Example: Retrieve substances and their related structures.

GET /Substances?$expand=Structures

$select: Selects a subset of properties to include in the response.

Example: Retrieve only the ReaxysRegistryNumber and MolecularFormula properties of substances.

GET /Substances?$select=ReaxysRegistryNumber,MolecularFormula

$count: Count number of entities in result

Example:

GET /Substances?$filter=MolecularWeight gt 300&$count=true

$top: Limits the number of results returned. The max value for $top is 100.

Example: Retrieve the top 5 substances.

GET /Substances?$top=5

$skip: Skips a specified number of results.

Example: Skip the first 10 substances and retrieve the next set.

GET /Substances?$skip=10

$filter

The $filter parameter allows you to narrow down your result set based on specific criteria. It uses a simple syntax to compare properties within your data. For example, you can filter for substances by any core property or properties of associated entities:

Retrieve substances with a molecular weight greater than 300.

GET /Substances?$filter=MolecularWeight gt 300

In this example, only Substances with a MolecularWeight greater than 300 will be returned.

$expand

The $expand parameter lets you include related entities in your results. For instance, you can fetch the melting points information along with the substance data:

GET /Substances?$expand=MeltingPoints

Here, each substance returned would include its associated melting point details, reducing the need for additional API calls.

The $expand parameter also lets you include related top-level resources and their related property entities. For instance, you can fetch datapoints and their associated citations along with the patent details for each citation:

GET /Datapoints?$Citations($expand=PatentBibliographies)

$select

The $select parameter is used to specify which fields or properties you want to include in your response. This is particularly useful for reducing bandwidth and improving performance by returning only the necessary data. For example, if you are interested only in a substance Reaxys registry number and molecular formula:

GET /Substances?$select=ReaxysRegistryNumber,MolecularFormula

This query will return only the ReaxysRegistryNumber and MolecularFormula properties for each substance.

$top

The $top parameter limits the number of records returned by the query, which is especially useful when dealing with large datasets or implementing pagination. For instance, to retrieve only the first 5 substances:

GET /Substances?$top=5

This query ensures that your response contains no more than 5 substances.

$skip

The $skip parameter is used in conjunction with $top to paginate your results by skipping a specified number of records. For example, to skip the first 20 substances and then return the next 5:

GET /Substances?$skip=20&$top=5

This is ideal for building user interfaces that display data across several pages.

$orderby (Standard Property Sorting)

Orders results by one or more primitive/core properties in ascending or descending order. This is used for standard OData sorting on resources such as Citations. Null values are expected to be placed at the end of the sorted list.

Example: Order citations by publication year (default ascending).
GET /Citations?$orderby=PreferredPublicationYear

Example: Order citations by entry date descending.
GET /Citations?$orderby=CitationEntryDate desc

Example: Order citations by document type ascending (string sorting).
GET /Citations?$orderby=DocumentType asc

$orderby (Similarity-Based Sorting for Substances)

Orders **Substances** by similarity to a reference structure using the OData function data.SortBySimilarity(). This is only applicable to Substances collections produced by structure or similarity search and is not supported for other resources.

Example: Order structure search results by similarity descending.
POST /Substances?data.SearchByMolecule&$orderby=data.SortBySimilarity() desc

Example: Order similarity search results by similarity ascending and return similarity scores.
POST /Substances?data.SearchBySimilarity&$expand=SubstanceSimilarities&$orderby=data.SortBySimilarity() asc

Note: Similarity sorting is not applicable to non-substance collections. Requests such as:
GET /Citations?$orderby=data.SortBySimilarity()
will respond with HTTP 40x (unsupported).

Bringing It All Together

You can combine these query parameters to create powerful and precise queries. For example, if you want to retrieve the first 5 substances with a molecular weight between 463 and 465 and their NMR spectra details but only the Reaxys Registry Number and the Chemical Names, your query might look like this:

GET /Substances?$filter=MolecularWeight ge 463 and MolecularWeight le 465&$expand=NmrSpectroscopies&$select=ReaxysRegistryNumber,ChemicalNames&$top=5

This comprehensive query does the following:

Filters substances to include only those with a molecular weight between 463 and 465.

Expands the response to include NMR spectra details.

Selects only the Reaxys Registry Number and Chemical Names fields of the substances.

Limits the response to the first 5 records.

Conclusion

OData's query parameters offer a robust way to interact with your data, enhancing both flexibility and performance. Whether you need to filter records, include related entities, select specific fields, or implement pagination, these tools simplify complex queries and minimize the amount of data transferred. By mastering parameters like $filter, $expand, $select, $top, and $skip, developers can build efficient, scalable APIs that meet the diverse needs of modern web applications.

Advanced OData Filtering

OData offers a robust set of query parameters, and in addition to the foundational ones like $filter, $expand, $select, $top, and $skip, it also supports several advanced features that allow for more expressive filtering. Here's an extended guide with examples covering some of these advanced functions, including any, contains, startswith, and endswith.

Any

Check whether a collection contains at least one matching element.

The any function filters entities based on collections (multi-value properties or navigation properties). It returns true when at least one element in the collection satisfies a condition.

Any with an inner condition (predicate)

Use this form when you want to match specific values inside a collection.

Example:

A Substance has the ChemicalNames property (an array of strings). The following query returns Substances that have the chemical name "pyrrole":

GET /Substances?$filter=ChemicalNames/any(cn: cn eq 'pyrrole')

Explanation:

For each Substance, the query evaluates the ChemicalNames collection. If any element (represented by the alias cn) equals 'pyrrole', that Substance is included in the result set.

Any without an inner condition (presence check)

You can also use any() without an inner condition to test whether a collection is not empty (i.e., the entity has at least one value for that collection property).

Examples:

Return Substances that have at least one solubility entry:

GET /Substances?$filter=SolubilityMcsList/any()

Return Substances that have at least one solubility product entry:

GET /Substances?$filter=SolubilityProductMcsList/any()

Combine presence checks using logical operators:

GET /Substances?$filter=SolubilityProductMcsList/any() and SolubilityMcsList/any()

GET /Substances?$filter=SolubilityProductMcsList/any() or SolubilityMcsList/any()

Explanation:

In these queries, CollectionProperty/any() evaluates to true when the collection contains one or more items. This is commonly used as a “has data” filter for multi-value properties or relationships.

Contains

Check if a string property contains a value.

The contains function checks whether a given property (typically a string) contains a specified substring. This is useful for text searches within properties.

Example:

To retrieve citations where the current patent assignee includes the substring “jingpu”:

GET /Citations?$filter=contains(PatentAssignee, ‘jingpu‘)

Explanation:

This query filters the citations, returning only those whose PatentAssignee property includes “jingpu” anywhere within the text.

Startswith

Check if a string property starts with a specific prefix.

The startswith function determines if the value of a property begins with a specific substring. This is especially useful when you want to fetch entries that have a common prefix.

Example:

If you want to get citations whose PatentAssignee starts with the word “jingpu”, you could write:

GET /Citations?$filter=startswith(PatentAssignee, ‘jingpu‘)

Explanation:

This query filters the citations, returning only those whose PatentAssignee property starts with “jingpu”.

Endswith

Check if a string property ends with a specific suffix.

Similarly, the endswith function checks if a property ends with a given substring. This might be useful for locating records that follow follow a specific naming convention such as chemical names.

Example:

If you want to get citations whose PatentAssignee ends with the word “jingpu”, you could write:

GET /Citations?$filter=endswith(PatentAssignee, ‘jingpu‘)

Explanation:

This query filters the citations, returning only those whose PatentAssignee property ends with “jingpu”.

Working with Complex Types

In addition to simple properties (like InChiKey, MolecularWeight, or PreferredPublicationYear), entities in OData can include complex types. A complex type is a structured property that itself contains multiple properties, but unlike navigation properties, complex types do not have keys or identity.

Example of a Complex Type:

The citations property CitationJournals provides details on the documents source journal and is modeled as a complex type:

```
{
  CitationNumberId : Edm.Int64
  ArticleNumber : Edm.String
```

```
  CitationJournalId : Edm.Int64
  CitationUnresolved : Edm.String
  CitedBy : Edm.Decimal
  Coden : Edm.String
  CountryCode : Edm.String
  Doi : Edm.String
  Editor : Edm.String
  Isbn : Edm.String
  Issn : Edm.String
  JournalReviewWithoutCoden : Edm.String
  JournalTitle : Edm.String
  JournalTitleShort : Edm.String
  LanguageCode : Edm.String
  LocationOfPublisher : Edm.String
  Number : Edm.String
  Page : Edm.String
  Pii : Edm.String
  PublicationYear : Edm.Decimal
  Publisher : Edm.String
  ScopusId : Edm.String
  SeriesVolume : Edm.String
}
```

Querying Complex Types

You can access individual properties of a complex type using / notation in your queries. Because Reaxys is modeling these complex types as collection, i.e. multi-value properties, the OData any function must be used.

$filter on a complex property:

GET /Citations?$filter=CitationJournals/any(cj:cj/Doi eq '10.1002/jcc.27080 ')

This returns the citations for the DOI 10.1002/jcc.27080.

This notation is also used to access navigation properties and their individual properties, e.g. filter on related entities or related top-level resources.

Filter substances that are reported in a given patent

GET /Substances?$filter=Citations/any(c:c/CommonPatentNumber eq 'US202417248 ')

Filter substances with a reported 25Mg NMR Nucleus

GET /Substances?$filter=NmrSpectroscopies/any(n:n/NmrSpectroscopyNuclei/any(nu:nu eq '25mg'))

Explanation:

The any function must be applied twice because the NMR nucleus is defined as a list of strings and a substance can have multiple reported NMR spectra

Current Limitations

When working with our OData API, it's important to be aware of some current limitations regarding advanced features. While the API supports a wide range of standard OData queries, several key functionalities have not yet been implemented. These limitations include:

- **Aggregation Features:**
  The API does not currently support the use of the apply operator for aggregations. This means that you cannot perform complex aggregation operations directly on the server side using the OData query language.
- **Sorting Features:**
  The API does not currently support sorting results and all records are returned sorted by the primary resource ID in ascending order.
- **Search and Compute Functions:**
  Additionally, features such as search and compute are not available at this time. Users will need to handle any full-text search or computed fields at the client level or via alternative backend processing.
- **Expand Limitations:**
  When filtering a resource using a related top-level resource property, then $expand query option currently only allows to expand into the resource the filter property resides within. Request to expand into another top-level resource will fail. This means that if you need to expand multiple related resources when using a related resource filter, you may need to perform separate requests or adjust your query strategy accordingly.
- **Server-Side Pagination:**
  At present, server-side pagination only supported for top-level resources, i.e. the nextlink metadata is currently only included for the main resources if results exceed the current page size. For entities included using $expand the nextlink metadata is currently not included, which requires clients to manage paging of datasets through client-side processing using the $skip option.
- **Structure Search Limitation:**
  The Reaxys API defines the OData actions data.SearchByMolecule, data.SearchByReaction, and data.SearchBySimilarityStructure for filtering by chemical structures. These actions currently can’t be combined with factual $filter statements, but allow to add $expand to include related entities.
- **Comparison and Logical Operators:**
  The Reaxys API currently does not support all OData comparison and logical operators. Not supported are:
  Comparison Operators: **ne**

Logical Operators: **not**
The comparison operators eq, gt, lt, ge, and le as well as the logical operators and, or are supported.

- **any() without inner predicate is not supported for collection properties**
    - The API currently does **not** support using any() *without an inner filter* on certain collection properties (including collections of complex types and some nested collection paths).
    - Requests using this pattern return **HTTP 501 Not Implemented** with the message:
      "any() without inner filter with collection properties is currently not supported."
    - **Examples (currently unsupported):**
        - GET /Citations?$filter=CitationJournal/any()
        - GET /Citations?$filter=PatentBibliographies/any(p:p/PatentBibliographyFamilyMember/any())
        - GET /Substances?$filter=Datapoints/any(d:d/DatapointSubstance/any())
    - **Workarounds:**
        - Use any(<alias>: <predicate>) when you can filter on a specific inner property, e.g.
          …/any(x: x/SomeProperty eq 'value')

These limitations are part of our ongoing development roadmap. We are continuously working to enhance the API and plan to support these features in future releases. Meanwhile, understanding these constraints will help you design efficient queries and avoid encountering unsupported operations during implementation.