cornerstone

Data Exporter API

Integration Guide

The Data Exporter allows you to retrieve your organizations data from the Cornerstone Real-time Data Warehouse for import to third-party tools. This is the same database that powers Custom Reports and Reporting 2.0 in your portal.

This document provides guidance regarding the setup and best practices for consuming CSOD's Data Exporter API, intended for the client's technical team responsible for building the client's side of the integration with the DEAPI. It is expected that this technical team has prior experience with REST web services and the OData protocol.

Cornerstone Edge CSX

Version history

	Date	Description
1.0	February 2021	Original Document
2.0	August 2021	Overall refactor to add more in depth information. Added sections comparing DEAPI with RAPI, accessing metadata, filtering rules, OData change tracking, post processing best practices, and relationship diagrams.

Table of contents

1. Introduction

The Data Exporter API (DEAPI) is a public facing web service that allows clients programmatic read-only access to their Cornerstone data via the Real-time Data Warehouse (RTDW). The API is RESTful, adheres to the OData protocol (http://www.odata.org/) and dynamically adjusts to reflect any client's schema. Currently, it allows access to all Data Exporter core objects.

The purpose of this document is to provide guidance regarding the setup and best practices for consuming Cornerstone's DEAPI. Note that this document is a supplemental aid provided by Cornerstone. It is meant to augment the DEAPI documentation and the data warehouse documentation available in the API Explorer or Developer Portal.

This document is intended for the client's technical team that will build the client's side of the integration with the DEAPI. It is expected that this technical team has prior experience with REST web services and the OData protocol. Business stakeholders or technical team members who may not have this knowledge may still refer to this document to get additional insights into the DEAPI, however, please direct any questions specific to the REST or OData protocols to your team members with this knowledge.

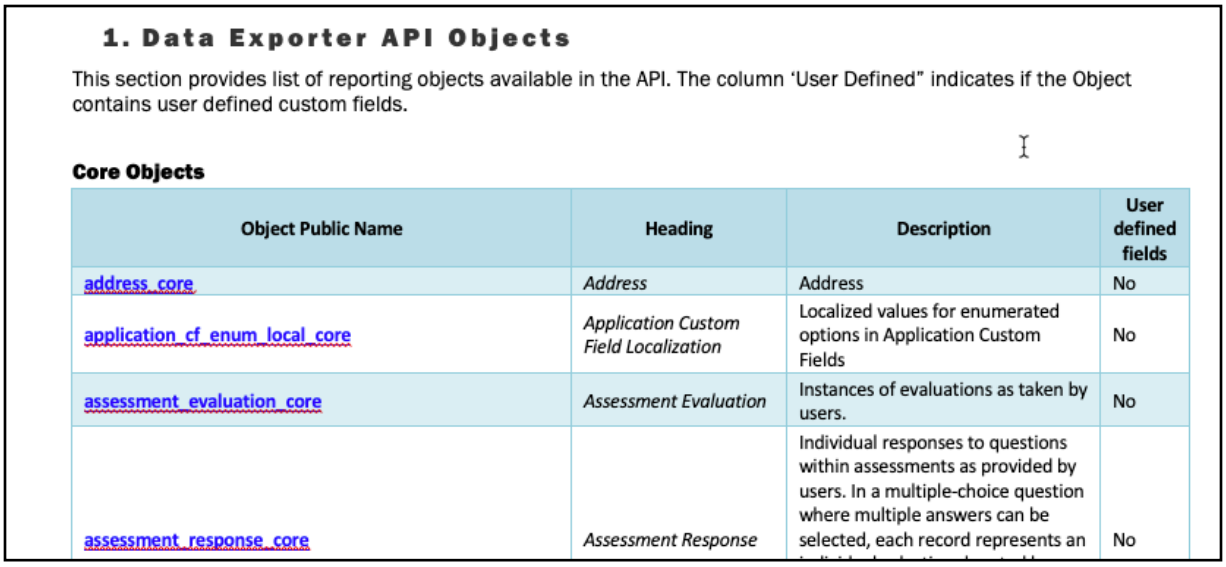
If you believe any of the information below is inaccurate or contains ambiguous information, please contact your Integration Consultant or log a case with Global Customer Support (GCS).

2. Product Documentation

While this document provides a good starting point to understand the DEAPI, you will need to reference additional documentation while developing your code. Following are additional resources that can and should be referenced.

2.1. Cornerstone Success Center

Documentation describing the schema for the various objects is available in the DataExporter and Reporting API community within the Cornerstone Success Center (CSC). As you develop code and/or ETL to consume CSOD's DEAPI, you will need to refer to this documentation for schema details and data dictionary.

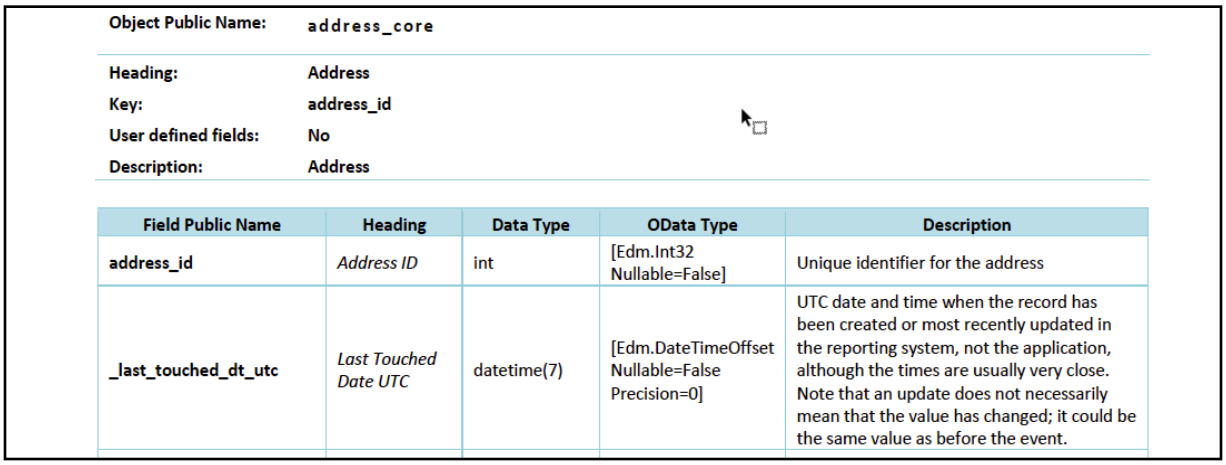


1. Data Exporter API Objects

This section provides list of reporting objects available in the API. The column 'User Defined" indicates if the Object contains user defined custom fields.

Core Objects

Object Public Name	Heading	Description	User defined fields
address_core	*Address*	Address	No
application_cf_enum_local_core	*Application Custom Field Localization*	Localized values for enumerated options in Application Custom Fields	No
assessment_evaluation_core	*Assessment Evaluation*	Instances of evaluations as taken by users.	No
assessment_response_core	*Assessment Response*	Individual responses to questions within assessments as provided by users. In a multiple-choice question where multiple answers can be selected, each record represents an	No



Object Public Name: address_core

Heading: Address
Key: address_id
User defined fields: No
Description: Address

Field Public Name	Heading	Data Type	OData Type	Description
address_id	*Address ID*	int	[Edm.Int32 Nullable=False]	Unique identifier for the address
_last_touched_dt_utc	*Last Touched Date UTC*	datetime(7)	[Edm.DateTimeOffset Nullable=False Precision=0]	UTC date and time when the record has been created or most recently updated in the reporting system, not the application, although the times are usually very close. Note that an update does not necessarily mean that the value has changed; it could be the same value as before the event.

You can also post questions to the group in the CSC community and receive updates on patches and releases.

2.2. API Explorer and Developer Portal

Current documentation on DEAPI endpoints, authentication mechanism, sample code, and supported OData queries is available via the portal in the API Explorer: Admin > Tools > EDGE > API Explorer > Reporting API > Data Exporter

- If you wish to provide access to the API Explorer to your team members who do not have access to your CSOD portal, you can refer them to the publicly available link for the API Explorer. No permissions or backend settings are needed to access this link.

In the second half of 2022, the publicly available CSOD Developer Portal (https://csod.dev/) will replace the API Explorer. Specific documentation for each API endpoint, including business rules, validation errors, sample request and response payloads, and downloadable OpenAPI YAML files will be added to the Developer Portal.

3. Considerations

3.1. Replacement for the Reporting API?

While DEAPI doesn't cover as many reporting objects as the Reporting API yet (additional endpoints are added all the time), if a particular business item is covered by both APIs you should give strong consideration to DEAPI over the Reporting API (RAPI).

Cornerstone's RAPI has been around for several years and been the de facto integration mechanism for external reporting systems. RAPI endpoints call database views that attempt to pull data on the fly from many different tables and then present a denormalized representation of that data in the response. Viewed in the context of an Extract, Transform and Load (ETL) paradigm, Cornerstone would extract and transform the data and the end user only needed to receive and load that data (although additional user transformation may be required if Cornerstone data is combined with other external data sources). While this paradigm can be an effective way to pull certain types of data, it does not scale effectively, particularly over an API interface, as the amount of data increases and more frequent reporting updates are desired.

The DEAPI is the spiritual successor to RAPI. It recognizes that may customers are starting to load data from multiple sources into their own data warehouse and data lake solutions where an Extract, Load and Transform (ELT) paradigm is becoming more prevalent. DEAPI surfaces only "core" data objects built for DEAPI and the file-based Data Exporter. These core objects contain streamlined, normalized data that are more efficient and quicker to transfer via API (compared to RAPI), and are easier for end users to combine and transform in various ways on their end. Core objects provide extended metadata that inform users how to most effectively interact with those objects, for example which objects support change tracking, which fields are filterable, and/or how to join with data in other objects. **Most importantly, core objects provide two different mechanisms for customers to receive changed data instead of pulling an entire data set**: querying by '_last_touched_dt_utc' field, or by using built-in OData change tracking functionality (rolling out to specific core objects beginning in the August 2022 release). With smaller data sets containing only changed data, customers can consider solutions where their data is retrieved and updated multiple times per day instead of once-per-day common with many ETL implementations.

The table below lists some of the key differences between RAPI and DEAPI.

	Reporting API	Data Exporter API
Core objects available	No	Yes
Keyed	Sometimes	Always
Pagination by Key Time Complexity (single Object, per page)	Up to O(n log n)	O(log n)
Normalized	Sometimes	Always
Extended Metadata	No	Yes

Enhanced Data Types	No	Yes
"Last Touched Date" availability	Sometimes	Always
OData Delta availability1	No	Yes*
Automatic Per-Client Schema Evolution	No	Yes
Compression	gzip	broti/gzip
Throttling Limit	Variable	120 calls per min
Max page size	Variable	up to 10,000 per call
*Starting in August 2022 with the 'user_ou_core' and 'user_ou_multi_core' objects. Support for additional objects will be rolled out and announced on the CSC and can be determined by inspecting an object's metadata.		

3.2. Ongoing Maintenance and Testing

For information related to patches and releases, please check the Cornerstone Success Center to retrieve in-depth information on patch/release schedules as well as documentation outlining the upcoming changes. Specifically for the Data Exporter API, please monitor any changes regarding data warehouse as they are published in the DataExporter and Reporting API community. The release calendar available in the Cornerstone Success Center outlines the exact dates for each release and patch.

- If you cannot access the Cornerstone Success Center, please advise your Integration Consultant or Client Success Manager.

Best Practices

- Please note that during a release, changes are pushed to the staging environment (typically 3 weeks) prior to the push to production/pilot. It is highly recommended that you maintain a test environment before and after implementation to monitor for any breaks to existing API integrations.

3.3. Security

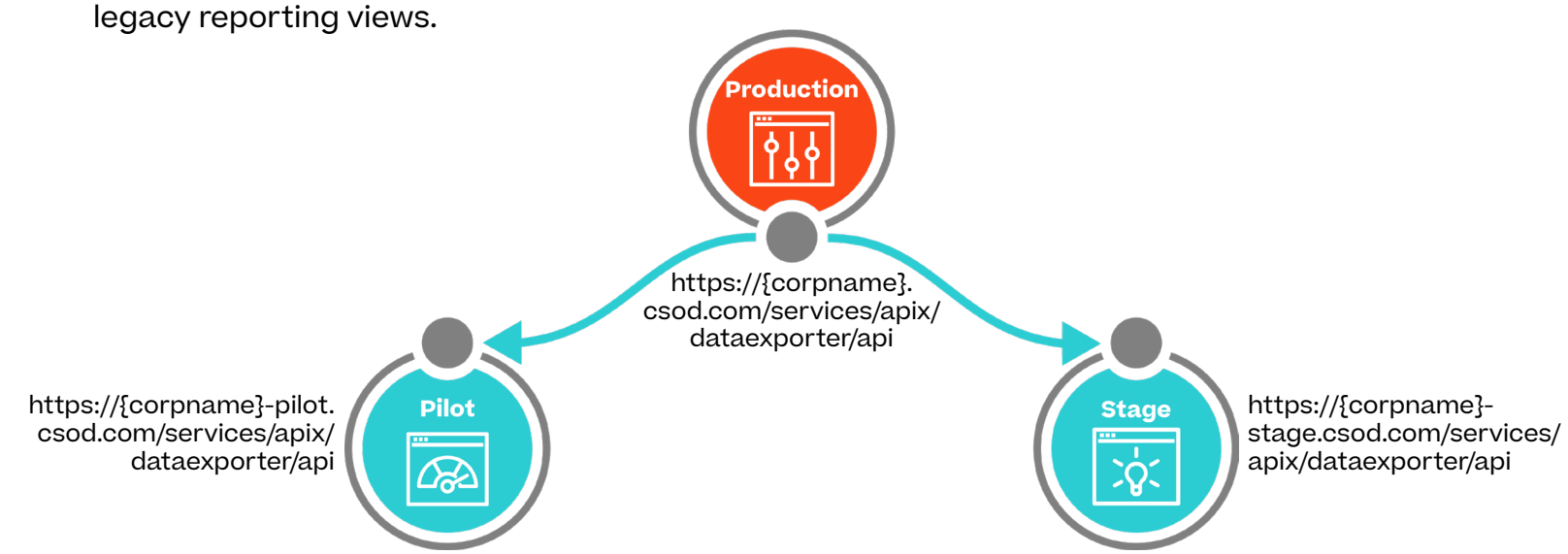
There are limited data access controls with the Data Exporter API. You can control the Objects your API keys have access to by adjusting the scope of the application. However, **there are no available constraints to grant/deny access at a field or data level**. For example, using OAuth 2.0 scopes, you can restrict access to your application to just the user and transcript views. However, there is no facility to further restrict access to data for users from specific division or region. This type of security must be managed by the client. As a result, it is very important to not share any API keys or codes to any individuals who should not have access to your data within CSOD.

Best Practices

- OAuth 2.0 Applications – Cornerstone's APIs use the OAuth 2.0 authorization framework. Currently, Cornerstone only supports OAuth 2.0's Client Credential grant type. You can adjust the scope of the application and the tokens. To get a client ID/secret, you must register your application by navigating to Admin > Tools > Edge > API Management > Register OAuth 2.0 Application > Register New Application. On this page, **the user that you associate with the OAuth 2.0 application must have the 'Reporting API – Read Only' permission.**
  - You can control access to the DE API by adding/removing the 'Reporting API – Read Only' permission to the user associated with the OAuth 2.0 application.
  - You can control access to specific endpoints in the DEAPI by adjusting the scopes for your application. See section 5.1 for additional information on scopes.
- You can also temporarily revoke access by disabling the OAuth 2.0 application in the API Management page. You can permanently revoke access by uninstalling the OAuth 2.0 application in the API Management page. Please note that this will immediately revoke any access tokens issued by Cornerstone for that application.

3.4. Reporting Environments

Every CSOD client typically has three environments – Pilot, Stage and Production. The DEAPI can be enabled in all three environments. Clients should ideally maintain a stage or test environment on their end to consume and test the API. ** Please note the endpoint for the DEAPI is different than the one for the Reporting API. The Reporting API is used to access the legacy reporting views.

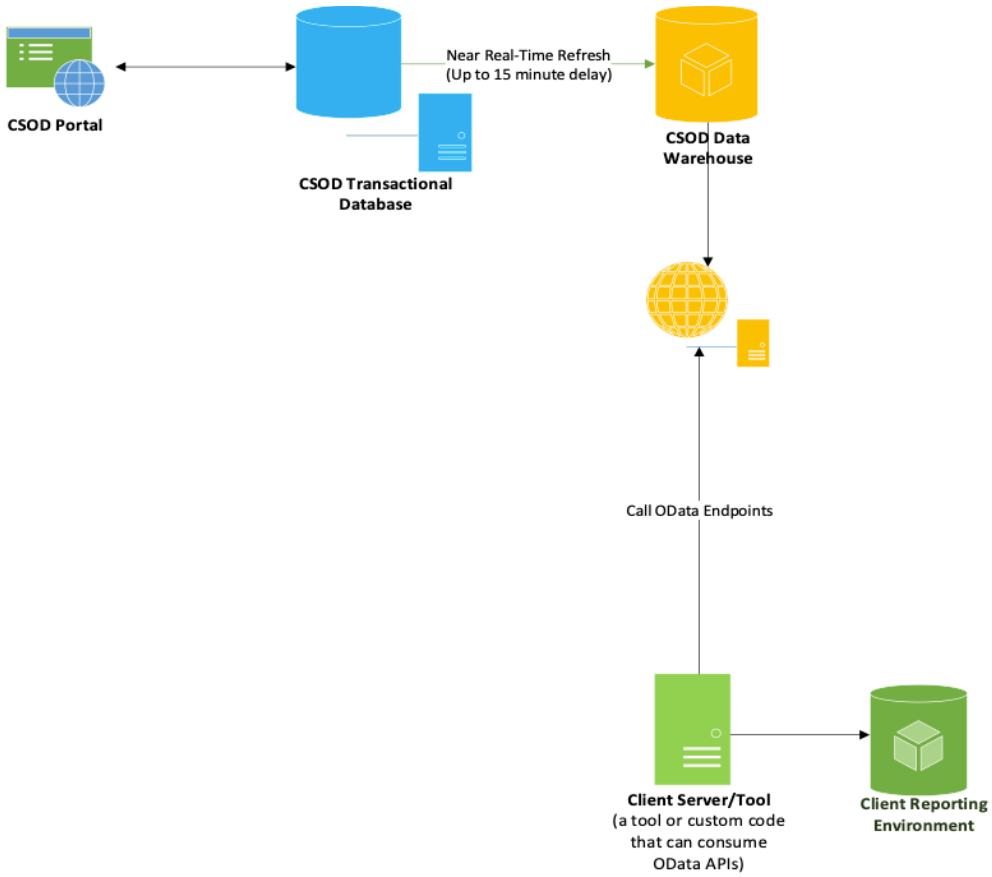


Best Practices

- Please check with your implementation team or system administrators to confirm which environment has the most relevant data for testing.
- It is recommended to use the CSOD pilot or staging environment while testing. This will provide the team the ability to stage data for validation purposes. **It is highly recommended not to stage any data in the production environment. This information CANNOT be deleted.**
- As previously indicated, it is also important for clients to create and maintain a test environment for release planning.
- It is important to note that the data warehouse may have certain database ID's (such as lookup IDs and form question IDs) that can be out of sync between environments. If possible, it is recommended to schedule a copy-down from production to pilot/stage for consistency with database ID fields. To schedule a copy down, please contact your Integration Consultant or log a case with Global Customer Support (GCS). **Please inform all implementation teams or your Client Success Manager prior to scheduling a copy down as this could affect other work being done across environments.**

3.5. CSOD Data Warehouse

Every portal and environment has two sets of databases – a transactional database and a data warehouse database (officially referred as the Real-time Data Warehouse (RTDW)). The RTDW is refreshed near real-time from the transactional database. The diagram below illustrates this architecture.



Best Practices

- It is important to note that there could be up to a 15-minute delay between the syncing of the transactional database and the data warehouse. This should be noted as you build out any custom delta processes using filters. For example, if you are using a date in a $filter condition, you may want to allow for a 15-minute window as shown in the table below. This becomes less important when using OData 4.0 change tracking capabilities (described in greater detail in section 5.2.5).

Sample Scheduling

Condition	Reporting API
Where modified date >= 1:00 AM and < 2:00 AM	2:15 AM
Where modified date >= 2:00 AM and < 3:00 AM	3:15 AM
Where modified date >= 3:00 AM and < 4:00 AM	4:15 AM

4. Enabling Data Exporter API

There is no charge to access the DEAPI in the Pilot and Stage environments. Navigate to Admin > Tools > Edge > Marketplace. To use the Data Exporter API, you must enable the Legacy **Reporting API**. Search for and click on the Reporting API. Click the Install button to get started. Accept the terms and conditions and click on **Configure Now**. On the API Management page, ensure the toggle for Reporting API is turned on. Then, follow the steps below to get your API credentials.

When ready to move to Production, please reach out to your Cornerstone Client Executive to have the integration installed in your CSX Production portal.

Registering your OAuth 2.0 Application

1. Log in to your portal. Ensure you have the 'Edge APIs - Manage' security permission.
2. Navigate to Admin > Tools > Edge > API Management > Manage OAuth 2.0 Applications.
3. Click on Register New Application. Review the Setup Instructions on that page to create your application. Note that the user associated with the OAuth 2.0 application must have the 'Reporting API - Read Only' security permission.
4. Copy the Client Secret and Client ID and use it in your external application that needs access to the Reporting API.

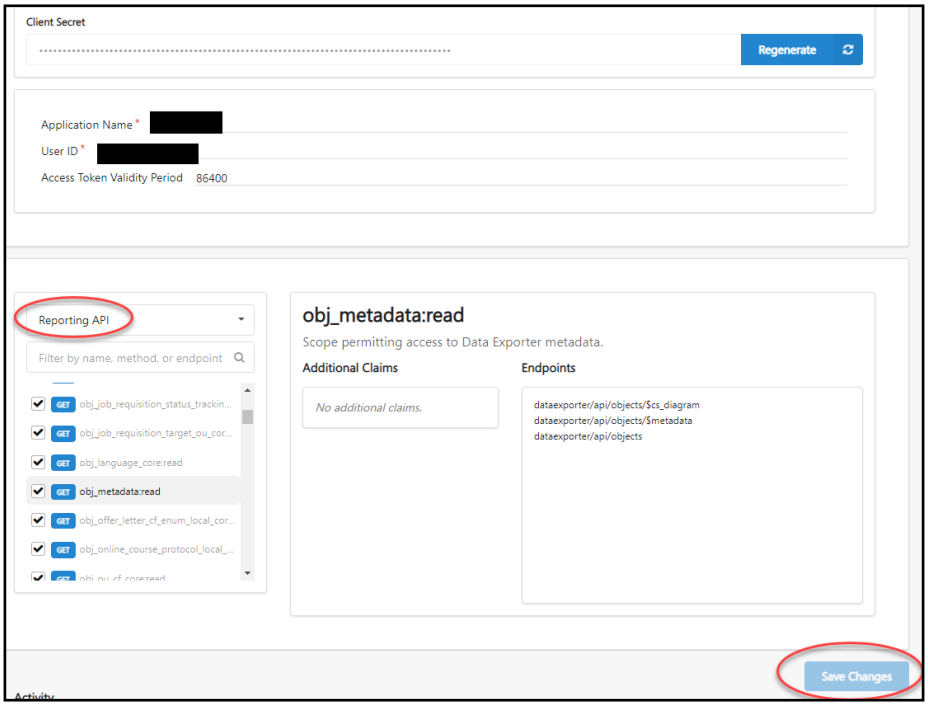
5. Consuming Data Exporter API

5.1. Authentication

Cornerstone APIs use the OAuth 2.0 authorization framework. Currently, Cornerstone only supports OAuth 2.0's Client Credential grant type. To retrieve an encrypted authentication token, pass your application's Client ID and Client Secret to the following URL: **https://{environment}.csod.com/services/api/oauth2/token**. The received token can then be passed in the request header of subsequent API calls. The token will include all endpoints that the application is allowed to access. You can adjust the endpoint scope of an application by clicking on the application on the 'Manage OAuth 2.0 Applications' page, selecting 'Reporting API' in the menu, then assigning specific endpoints. Be sure to click 'Save Changes' when done.

Best Practices

- Please note that there is an expires_in field in the response of the OAuth 2.0 token endpoint. This field contains the validity period of the OAuth 2.0 access token issued by Cornerstone in seconds (maximum = 86400 seconds = 24 hours). It is not required to generate a new access token for every API call. You can reuse the same access token until the token expires.
- You should limit the scope of your OAuth 2.0 application to just the endpoints you need access to. **You should also include 'obj_metadata:read' if you wish to access and review extended metadata information like field filter options, relationship diagrams, and delta capabilities.**
- Data Exporter API objects will be listed under the "Reporting API" category and always start with the prefix "obj" (other endpoints prefixed by 'vw_rpt' are associated with the legacy Reporting API).
- For a full list of object available in the DEAPI, please check the Data Exporter reference documentation.



5.2. OData Queries

CSOD's Data Exporter APIs use the OData protocol. For more information regarding the OData protocol, please check http://www.odata.org/. Cornerstone is currently using OData version 4.0. Please note that not every OData option is available. Please see the API Explorer and/or the Developer Portal for more details on the options you have with CSOD's implementation of OData.

5.2.1. Base Queries

After you have your authentication token, you can begin making API calls to retrieve your data by sending a GET request to **https://{environment}.csod.com/services/api/x/dataexporter/api/objects/{core_object}**. A base query, without any filters applied, will retrieve all data in the core object for that environment.

Best Practices

- After you have thoroughly tested your solution and are ready to go live in a production environment, you should strive to call an unfiltered base query for each object once in order to establish a local baseline, then only request changes thereafter. This lets you take advantage of the speed and efficiency of smaller data packages moving forward.

5.2.2. Extended Metadata

You can inspect the definition and capabilities of each core object by sending a GET request to **https://{environment}.csod.com/services/api/x/dataexporter/api/objects/$metadata**. The response will contain an OData compliant XML document that you can parse to generate a local data dictionary. It also describes how you can interact with various objects, such as whether change tracking is supported, which fields can be used in filters, etc. This metadata document includes:

- **Core object description elements**, such as ObjectPublicName, ObjectDescription, Object Heading, and ObjectIsUserDefined (i.e., whether object contains custom fields or not).
- **Core object key field(s)**. Fields that uniquely identify a record in the object's data set. See section 6.2 for additional information.
- **Core object index definitions**. Fields used as indexes in the RTDW. Only fields defined in an index can be used with the $filter querystring parameter. See section 5.2.4.2 for additional information.
- **Field definition elements**, such as FieldPublicName, FieldDescription, data type, whether the field is nullable, etc. See section 6.4 for additional information.
- **Navigation / Relationship definitions**, which can be used to determine which tables can should joined in your local data store after you have retrieved your API data. URLs are also available that expose graphical diagrams that describe the relationships between tables. See section 6.3 for additional information.
- **Miscellaneous object capabilities**, such as OData change tracking. See section 5.2.5 for more information.

Best Practices

- As you build, test and troubleshoot your DEAPI-based data solution, you should regularly reference the in-line documentation exposed by this metadata. Common problems may include running filters using non-indexed fields, or attempting OData change tracking capabilities on objects that do not support this functionality yet.

5.2.3. Selecting Fields to Return

A core object may contain more fields than you need for your business use case. While you could retrieve and store all of these fields with the thought that you could eventually use them, this comes with some additional expense, not only with the additional time needed to serialize and transport the data, but also the additional storage space on the receiving end. It's important to review your use case and what you plan to do with the data, and then retrieve only what you need rather than getting everything. The OData **$select** query string element can be used to retrieve specific fields of data from a core object, for example:

```
https://{environment}.csod.com/services/api/x/dataexporter/api/objects/users_
core?$select= user_id,user_ref,user_name_last,user_name_first,user_status_id
```

Best Pracice

- You should reduce your payload at the source by using $select, $filter and/or OData change tracking functionality (latter two options discussed in the following sections). Doing so will yield significant performance gains in response and processing time from the service.
- Most browsers and applications have a max URL of 2,083 characters. This is important to keep in mind as you build out your solution.

5.2.4. Filtered Queries

5.2.4.1. $Filter Overview

The OData **$filter** query string element can be used to retrieve subsets of data from a core object, for example:

```
https://{environment}.csod.com/services/api/x/dataexporter/api/objects/users_
core?$filter= _last_touched_dt_utc ge cast('2022-08-01', Edm.DateTimeOffset)
```

which is also equivalent to

```
https://{environment}.csod.com/services/api/x/dataexporter/api/objects/users_
core?$filter= _last_touched_dt_utc ge 2022-08-01T00:00:00.000Z
```

Multiple filter criteria can be combined in a single query by using the 'and' or 'or' keywords, such as:

```
https://{environment}.csod.com/services/api/x/dataexporter/api/objects/users_
core?$filter= _last_touched_dt_utc ge 2022-08-01T00:00:00.000Z and _last_touched_
dt_utc lt 2022-08-02T00:00:00.000Z
```

The table below lists the various filter operators supported by DEAPI.

URL Option	Syntax
Equals	`{baseURL}/objects/users_core?$filter=user_id eq 78`
Not Equals	`{baseURL}/objects/users_core?$filter=user_id ne 78`
Greater Than	`{baseURL}/objects/users_core?$filter=user_create_dt gt cast('2022-08-01', Edm.DateTimeOffset)`
Greater Than or Equal	`{baseURL}/objects/users_core?$filter=user_create_dt ge cast('2022-08-01', Edm.DateTimeOffset)`
Less Than	`{baseURL}/objects/users_core?$filter=user_create_dt lt cast('2022-08-01', Edm.DateTimeOffset)`
Less Than or Equal	`{baseURL}/objects/users_core?$filter=user_create_dt le cast('2022-08-01', Edm.DateTimeOffset)`
Logical 'AND'	`{baseURL}/objects/users_core?$filter=user_create_dt le cast('2022-08-01', Edm.DateTimeOffset) and user_id eq 78`
Logical 'OR'	`{baseURL}/objects/users_core?$filter=user_create_dt le cast('2022-08-01', Edm.DateTimeOffset) or user_id eq 78`
Logical 'NOT'	`{baseURL}/objects/users_core?$filter=not (user_id eq 78)`
'IN' [1,2]	`{baseURL}/objects/users_core?$filter=user_id in (1, 2, 3)`
NOT 'IN' [1,2]	`{baseURL}/objects/users_core?$filter=not user_id in (1, 2, 3)`

1. Maximum of 64 comma-separated list of primitive values can be passed per operand.
2. Combined primitive values across all operands cannot exceed 1500 characters.

Best Practices

- You must use '**and**' and not '**&**' (the ampersand symbol) when combining multiple filter criteria… if you use the '**&**' symbol, subsequent parts of the filter criteria will be interpreted by the request handler as a new query string parameter and will likely be rejected as a malformed url. A similar mistake while testing is to omit the '**$**' in front of the **$filter** keyword (or other keywords like **$select**). In this situation, the request handler will recognize a properly formed url query string and will forward the request, but '**filter**' by itself is not recognized as an OData keyword and your filter criteria will not be applied. The response will contain an unfiltered data set, which may not be readily apparent until it is inspected carefully.

5.2.4.2. DEAPI Filter Rules

Unlike RAPI, which lets you filter by any field regardless of performance impact (and many times those queries will fail if performance is bad enough), DEAPI restricts which fields can be used in a filter to ensure adequate performance and delivery to the end user. Continuing the ELT paradigm discussion in section 3.1, the goal of DEAPI is not to provide "Query as a Service", but to provide a pipeline that lets you load data into your own store where you can further query, filter and analyze that data.

In order to filter on a field, two conditions must be met:

1. The field needs to be marked as filterable in the metadata ('FieldIsFilteringAllowed = true').
2. An index with that field must exist for the core object (found under 'ExtendedMetadata. Indexes' collection of each object). If an index includes multiple fields, then a user can filter on all or any fields starting from the first indexed field.

Generally speaking, if an index contains a specific field that field should also be marked as 'FieldIsFilteringAllowed = true'. However, the opposite is not necessarily true: it is quite common for a field to be marked as 'FieldIsFilteringAllowed = true' but not be included in an index. That is because indexes were created only for fields that CSOD's reporting teams felt were most likely to be used as filters (as indexes take space and maintaining them affects performance). However, nothing prevents customers from creating additional indexes on their local stores if the business need arises... so the 'FieldIsFilteringAllowed' property can largely be considered informational for such use cases. **Indexes will be your primary informational tool to determine how you can filter DEAPI objects directly**.

Filtering index rules:

- If an index contains fields A,B,C, then you can filter on either (A), (A, B), (A, B, C) or even (C, B, A), but not (B, A), (B, C), or (C). [Note: a different field order like (C, B, A) is only allowed if ALL fields in the index are present.] Sorting (using $orderby) has a similar requirement, but it is sensitive to index field ordering and direction. If an index has A, B, C, you can sort by (A), or (A, B), or (A, B, C), but not (B, A), (B, C) or any other combination. You can sort by (A desc, B desc) but not (A desc, B) or (A, B desc).
- If an object has multiple indexes, fields from different indexes cannot be combined. All fields being requested in the query should be covered by a single index. For example, if you have two indexes with fields A,B,C and D,E a single query cannot filter on (A,B,C,D,E).

For a filtering example, see the screenshot below of some users_core indexes. Note it is not a complete list of indexes, but a point in time snapshot of two specific indexes. Upon parsing the metadata you will see that the users_core entity implements two indexes:

- IX_rpt_user__user_ref covering field user_ref
- IX_rpt_user_name_last_name_first covering fields user_name_last and user_name_first. The 'Ordinal' of the field (zero-based) represents the significance of the field within the index. Here, user_name_last is the first field within the index and should always be present if this index is to be used.

```
<LabeledElement Name="IX_rpt_user__user_ref">
    <Collection>
        <Collection>
            <LabeledElement Name="PropertyName">
                <PropertyPath>user_ref</PropertyPath>
            </LabeledElement>
            <LabeledElement Name="Ordinal">
                <Int>0</Int>
            </LabeledElement>
            <LabeledElement Name="IsAscending">
                <Bool>true</Bool>
            </LabeledElement>
        </Collection>
    </Collection>
</LabeledElement>
<LabeledElement Name="IX_rpt_user_name_last_name_first">
     <Collection>
         <Collection>
             <LabeledElement Name="PropertyName">
                 <PropertyPath>user_name_last</PropertyPath>
             </LabeledElement>
             <LabeledElement Name="Ordinal">
                 <Int>0</Int>
             </LabeledElement>
             <LabeledElement Name="IsAscending">
                 <Bool>true</Bool>
             </LabeledElement>
         </Collection>
         <Collection>
             <LabeledElement Name="PropertyName">
                 <PropertyPath>user_name_first</PropertyPath>
             </LabeledElement>
             <LabeledElement Name="Ordinal">
                 <Int>1</Int>
             </LabeledElement>
             <LabeledElement Name="IsAscending">
                 <Bool>true</Bool>
             </LabeledElement>
         </Collection>
     </Collection>
 </LabeledElement>
```

With the above indexes, a user can submit the following filter queries:

Valid:

- …/api/x/dataexporter/api/objects/users_core?$filter=user_name_last eq 'abc' and user_name_first eq 'abc'
- …/api/x/dataexporter/api/objects/users_core?$filter=user_name_first eq 'abc' and user_name_last eq 'abc'.
- …/api/x/dataexporter/api/objects/users_core?$filter=user_name_last eq 'abc'
- …/api/x/dataexporter/api/objects/users_core?$filter=user_ref eq 'abc'

Invalid:

- …/api/x/dataexporter/api/objects/users_core?$filter=user_name_first eq 'abc'.
  Reason: all fields of index should be present in filter starting with most significant field.
- …/api/x/dataexporter/api/objects/users_core?$filter=user_name_first eq 'abc' and user_ref eq 'abc'.
  Reason: No covering index found, cannot combine fields from two different indexes. Each filter query should be covered by a single index.

In the event an invalid filter request is made, an error message alerting you of the condition will be given.

5.2.5. OData Change Tracking / Delta

Cornerstone rolled out DEAPI OData 4.0 change tracking functionality for inserted, updated, and deleted data in the August 2022 release starting with the 'user_ou_core' and 'user_ou_multi_core' objects, with support for additional objects following soon thereafter. Prior to this Release, customers had to manually supply date/time filters in the URL to retrieve new and updated information (see previous section on '_last_touched_dt_utc'), and there was no way to be informed of deleted data for GDPR compliance. With OData 4.0 change tracking, users will find a delta link with an encrypted token at the end of their payload response. This token can be used within the next 15 days to receive any inserts, updates or deletes to/from the dataset represented by the original API query. After the changed data set is received, a new delta token is provided and should be used for the next data call (assuming your other filter criteria does not change).

To determine if a particular core object supports OData 4.0 change tracking, inspect the results of the $metadata endpoint. If ExtendedMetadata.ObjectSupportsChangeTracking' = 'true' for that object, then you can proceed with the steps below.

1. Insert 'Prefer: odata.track-changes' into the HTTP header to request that the endpoint return a delta link. The response header will include 'Preference-Applied: odata.track-changes' to indicate that the preference is applied. Subsequent pages will automatically reapply 'Prefer: odata.track-changes'. If an object does not support delta change tracking, you will receive a '501' (Not Implemented) error.

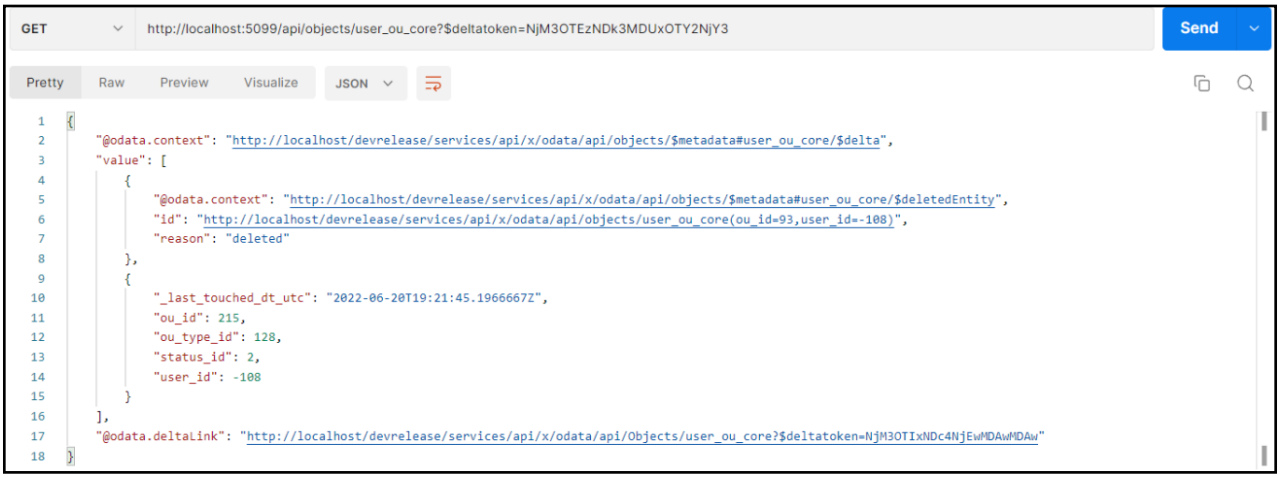
2. Process the response as normally would, using the @nextlink to page through the result set (see section 5.3 on paging). The last page of the result set will include a delta link with a delta token (e.g., "@odata.deltaLink": "user_ou_core?deltatoken=xxxx"). This delta token is an encoded representation of the original query and a timestamp. You should retain this delta link/token for your next request and only replace it when you receive the next delta link/token. If the request or response is interrupted or otherwise unsuccessful, or if you encounter an error while processing the records locally, can use the last token to request the delta result set again. You do not need to keep track of timestamps. You can repeat this step indefinitely, using the next delta link provided, as long as a delta link is used within 15 days.

3. If more than 15 days passes before you use the delta token, you will receive a '410' (Gone) error and will need to retrieve the full data set starting with step 1 again. Because 'Tombstone records' that represent deleted application data expire after 15 days and are cleaned out of the RTDW on a routine basis, any data retrieval that exceeds that time period cannot guarantee to be a true change tracking data set.

Delta result sets may include two types of records (see sample image below):

- Records deleted since the timestamp in the delta token. Record will include an 'id' field (which may consist of multiple composite key fields) and a 'reason' field with a value of 'deleted'. The 'id' field can be used to delete the corresponding record (if it exists) in the customer's local database.

- Records inserted or updated since the timestamp in the delta token. Note: the payload does not distinguish whether the record was inserted or updated… customers should continue to use their current MERGE or UPSERT process used to process Data Exporter API data.



For additional information on OData 4.0 change tracking, go to http://docs.oasis-open.org/odata/odata/v4.0/os/part1-protocol/odata-v4.0-os-part1-protocol.html#_Toc372793707.

5.3. Paging

The DEAPI implements the OData server driven paging protocol, which segments large results of data into pages. You can control the page size, i.e. the number of records per page, through the 'prefer' header in your request. For example,

```
prefer = odata.maxpagesize=10,000
```

The maximum accepted page size is 10,000. If the 'prefer' header is not included in your request, DEAPI uses the default page size of 1,000 records. When the following condition is met: total record count > page size, the service will inject into the page response a link to retrieve the next page of results.

```
{
  "@odata.context": "https://{environment}.csod.com/services/api/x/ api/x/
  dataexporter/api/objects/$metadata#transcript_core",
  "value": […],
  "@odata.nextLink": "https://{environment}.csod.com/services/api/x/
  dataexporter/api/objects/transcript_core?... "
}
```

"@odata.nextlink" should be treated as opaque, parsing it or manipulating could result in incorrect results and even errors.

5.3.1. Streaming

The DEAPI uses a streaming protocol, which means that it sends data out to the client as soon as it becomes available rather than waiting until it receives all the records from the database before sending them out. This allows the API to provide access to very large data sets very efficiently, using a relatively small memory footprint on our servers, allowing us to serve many more simultaneous requests.

5.3.2. Record Duplication

To provide you the most up to date data, the DEAPI's data is synced from the transactional database on a frequency of at most fifteen minutes. This is important to consider because it means the data the API operates on can change at any time, even while you are executing your queries.

As you retrieve records, page after page, newly synced data can cause the records positions to shift. A previously retrieved record can potentially re-appear in subsequent pages and potentially cause errors as you process it on your end.

It is highly recommended that you first stage the data you retrieve in a temporary storage, run any validation logic you deem necessary, including checking for duplicate records, before sending it to its final destination in your pipeline.

5.3.3. Record Count

It is good practice to get a count of records at the beginning of your process. The record count can also be useful to determine what to set your page size to. If the total count is relatively low but higher than the default page size, you may elect to increase the page size by setting it higher to reduce the number of pages you go through. Because the record count is a snapshot at the time of the request – it reflects the total possible records that meet the filtering criteria you specified, it is important to specify the same filtering criteria you will use when retrieving your records.

For example, using the following use case: Retrieve all completed transcript records in the last 24 hours.

1. Get the total record count.

   Request:

   ```
   Get https://{environment}.csod.com/services/api/x/dataexporter/api/
   objects/ transcript_core?$filter=user_lo_comp_dt ge cast('currentDate - 24
   hours', Edm.DataTimeOffset)&$count=true&$top=0
   ```

   It's important to specify $top=0 when retrieving the total count, so that the service only returns a count. Calculating the total count can be a very expensive operation on the system, especially when the object has a very large record set.

2. Parse the response.

   ```
   {
   "@odata.context": "https://.../services/api/x/dataexporter/api/
   objects/$metadata#transcript_core",
   "@odata.count": 31379,
   "value": []
   }
   ```

3. Proceed to retrieve the records

Request:

```
Get https://{environment}.csod.com/services/api/x/dataexporter/api/
objects/ transcript_core?$filter=user_lo_comp_dt ge cast('currentDate - 24
hours', Edm.DataTimeOffset)&$select=transc_user_id, tansc_object_id, reg_
num, user_lo_score,user_lo_comp_dt
```

Response:

```
{
  "@odata.context":
  "https://.../services/api/x/api/x/dataexporter/api/
  objects/$metadata#transcript_core",
  "value": […],
  "@odata.nextlink":"https://{environment}.csod.com/services/api/x/ /
  dataexporter/api/objects/transcript_core?...&$skiptoken=..."
}
```

Repeat call @odata.nextlink for additional records.

4. While you can compare your accumulated record count to the initial count you retrieved, they may not be identical due to additional or changed data that appears during the streaming/paging process.

5.3.4. Throttling

To ensure the best possible performance of the DataExporter API, CSOD has implemented throttle limits. The current throttle limit is **120 calls per min** in total.

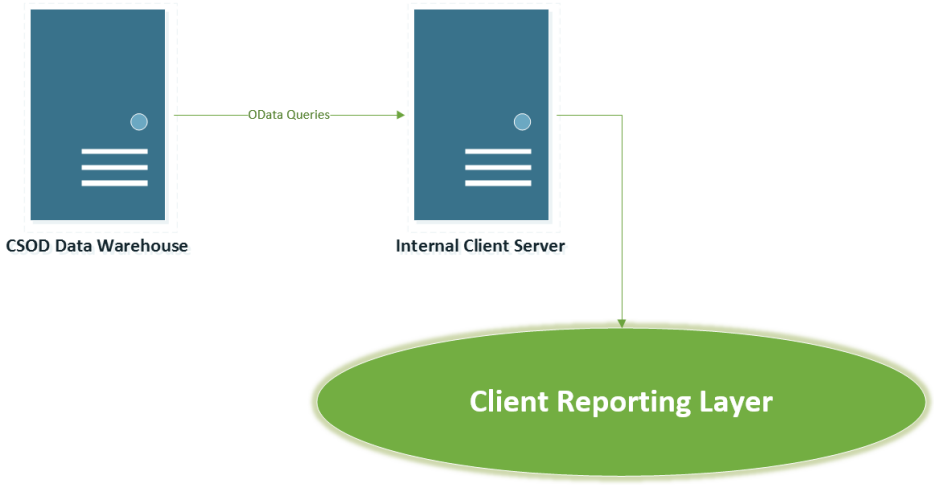
- This means you may call 2 objects 60 (2x60 =120) times in one minute or 12 objects 10 times in one minute (10x12=120).
- If you exceed the call limit, you'll receive a 429 Error. Throttle limits apply to all authenticated APIs calls. If a request is made after the throttle limit is reached, an HTTP error is thrown, which is to be handled elegantly. We recommend that interface applications queue such requests until the throttle limit duration have passed and the request can be re-sent.
- This is another area where the change tracking techniques described earlier can work to your advantage. By it's nature, change tracking results in smaller data sets, fewer pages of data, and hence fewer requests for that data.

5.4. Join Restrictions

With DEAPI, there is no option to perform any joins between objects in a single API query. Because of this, there are two recommended solutions to bring in data from various objects into a single report.

Option 1 – ELT to Internal Server (recommended)

In this approach, you can pull various core objects into a local database in which you are able to create and manage your own queries, views, and security. Using the delta and change tracking techniques described in previous sections, you can load and maintain local copies of data that are fairly close to real time. See section 6.3 for additional information on accessing relationship diagrams to see how you can join information in your local database after you have retrieved your data.



Option 2 – Iterative Loops

Within your coding environment, you can build out a looping mechanism to pull information from multiple tables. In the below example, we want to pull applicant information as well as some data on the requisition they have applied to. One down side to this approach is that it generates many small queries instead of grabbing batches of data in a related core object at once… this leads to greater potential of running into API throttling limits.

```
Endpoint =
https://{environment}.csod.com/services/api/x/dataexporter/api/objects/applicant_
core

For each job_applicant_id returned:

    Next Endpoint = https://{environment}.csod.com/services/api/x/dataexporter/
    api/objects/recruiting_core? $filter=ats_req_id eq { job_requisition_id from
    applicant }
```

5.5. Request Resiliency and Retry Guidance

The CSOD business objective for the API is to provide clients access to their growing reporting data programmatically. The goal for CSOD is to deliver a solution that not only meets the functional business needs of the clients, but one that is robust, secure and scalable. However, robust and scalable does not mean that errors will never occur. You should always code defensively, expect transient errors to occur and handle them gracefully.

In general responses with a status of 2xx are successful, however there are circumstances where this is not always the case. We will cover the most common types of errors and the special case when 200 is not actually successful.

5.5.1. Client Errors 4xx

The following errors require attention on your side, it usually involves fixing some aspects of your request before re-issuing it.

400 Bad Request / validation

Incorrect URL or parameters, such as the incorrect spelling of odata keywords $select, $filter, etc.

401 Unauthorized / Unauthenticated

Missing the right credentials or using an expired token in your request. Check the values you are sending and correct them accordingly.

403 Access Denied

The user account does not have the correct privileges to access the resource requested. If you encounter this error, you will need to login into your portal and assign the correct permissions to the user account being used to make the API calls.

404 Not Found

This error occurs when an incorrect resource is requested or the predicate for a resource does not match a record. For example

- Incorrect resource: requesting vw_rpt_usr instead of vw_rpt_user
- Incorrect predicate: requesting vw_rpt_user/1234, when a user record with that id does not exist

429 Limit Exceeded

This is an indication that you have exceeded your throttling limit for the API.

Implement a back off logic accordingly to stagger your calls around the limits.

```
{
    "status": "429",
    "timeStamp":"2018-05-15T18:06:55+0000",
    "error": {
        "errorId": "77ed61e0-7052-4fad-b265-c52679ab2cac",
        "message": "CSOD Too many requests.",
        "code":"429",
        "description": null,
        "details": null
    }
}
```

5.5.2. Server Errors 5xx

Server errors are generally the result of a malfunction that occurred on our servers. In most cases they are transient errors and the recommendation is to retry the request until it succeeds. You should cap the number of retries after which you would error out and report the error to Global Product Support (GPS).

500 Server Error

An error occurred which may be transient. You should examine the details of the error :

```
{
    "status": "500",
    "timeStamp":"2018-05-15T18:06:55+0000",
    "error": {
        "errorId": "77ed61e0-7052-4fad-b265-c52679ab2cac",
        "message": "error message",
        "code":"xxx",
        "description": null,
        "details": null
    }
}
```

In some cases, the error could be due to a timeout on the server. This could be due to a couple of reasons:

- The load on the data source resulted in an excessive time before it responded to the request
- The page size requested was too large, resulting in a timeout between the data source and the service

In both cases retrying the request could succeed, however to increase your rate of success, it is recommended you implement a back-off logic when retrying.

You might want to progressively reduce your page size. Using the count feature could be helpful in making that determination. If the count returned indicates you are dealing with a very large number of records, for example, over a million records and the records are very wide, i.e. have a large number of columns, then you should reduce the page size and exercise the optimizations suggested earlier in this document.

503 Service Unavailable

This likely to occur when the system is down for maintenance. Cornerstone sends out notifications to our clients in advance in which maintenance windows are specified. The recommendation is to pause your process until after the maintenance window has elapsed. If you receive this error outside an announced maintenance window, you should notify Global Product Support (GPS) immediately so they can investigate.

5.5.3. Special Case: 200 not a Success

As stated in section 5.4.1 of this document, the API uses a streaming protocol to allow it to support large volumes of data. The results are sent back as they become available from the database. You can think of it as a fire hose – the data flows directly from the source, rather than being accumulated entirely into a container, then shipped as a whole. Errors are handled the same way as well.

This can sometimes cause the Data Exporter API to initially send back a 200 success response, however as you continue to page through additional records, it may timeout.

How would this error manifest itself?

Let's say you issue a request with a page size of 10,000 records. The server processes the query successfully and begins to send back results. Because a number of records are successfully sent, the response status is automatically set to 200 OK.

You continue to receive valid records, however at some point an internal server error occurs due to a resource exhaustion, this typically manifests itself as a timeout. The HTTP protocol does not allow changing the status of a response once it has been sent back, and therefore the client side is unaware that an error occurred.

On the receiving end, you would continue to process the records, unfortunately the data in the stream no longer represents a valid record. Your JSON parser errors out when it encounters data which is not consistent with a record. The response may look as follows:

```
{
        "@odata.context": "https://{yourcorp}.csod.com/services/api/x/odata/api/
        views/$metadata#vw_rpt_transcript",
        "value": [
        {valid records}
        error message, could be html
```

If you are collecting the entire JSON payload before you begin parsing it, then it is likely it is not valid JSON anymore. If you are parsing the data stream on the fly then you will run into the error in real-time.

In both scenarios, the error is most likely transient, and the recommendation is to follow the same approach as a 500 service error of retry with back-off logic.

6. Post Data Retrieval

6.1. Data Processing Best Practices

There are many different environments (local storage, data warehouses, data lakes, etc.), technologies, and third party data vendors that customers can leverage after they have retrieved their data from the CSOD system. It would be foolhardy to provide rules that would adequately apply to all environments/scenarios. Nevertheless, here are some general guidelines and considerations as you process your data.

- **Store your unprocessed data in one location and copy it to another location before processing it. Or if you are going to transform the data, store the transformed data in an alternate location without modifying the original data.** This is particularly true when initially building and testing your data solution. Many times there will be an unexpected business logic error used during post-processing and the error will not be discovered until after the fact. If you retain the originally received data for a certain period of time, you can 1) verify the data in question was received in the first place, and 2) retry and troubleshoot your process. CSOD cannot vouch for the reliability of customer data after it has been delivered and processed by another entity. The alternative is to pull another full download of the relevant core objects, which defeats many of the efficiencies that the DEAPI provides.
- **Remember data is likely ordered/optimized for delivery, not necessarily for merge/upsert into the receiving system.** Received data may be ordered by _last_touched_dt_utc, particularly if you are taking advantage of change tracking capabilities. Many merge/upsert processes are most effective when the two data sets to be combined use the same index. Therefore, you may wish to reorder or index your data before merge. See section 6.2 to determine the unique keys for each core object.

Additional guidance can/will be added as they are forwarded by DEAPI customers.

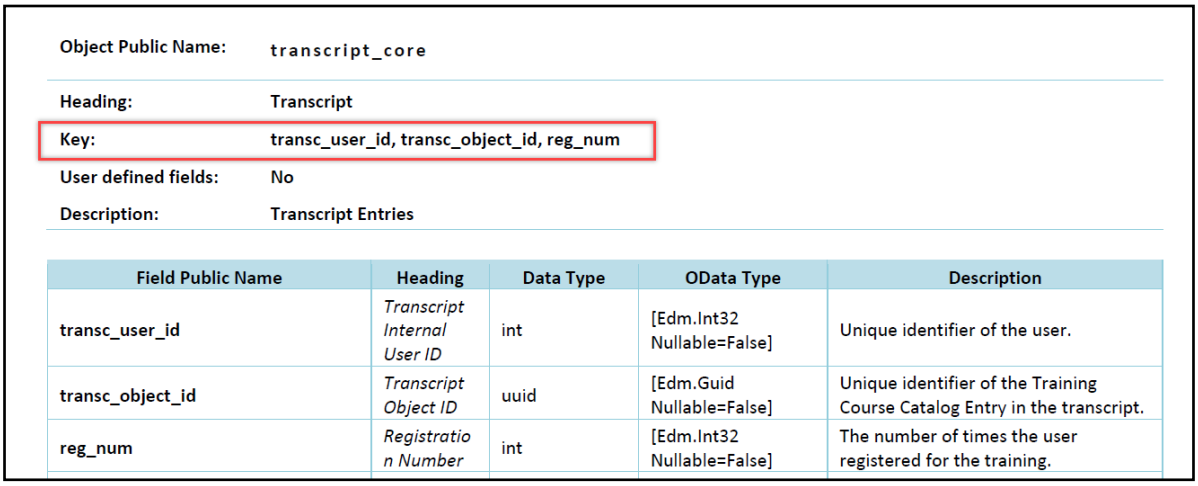
6.2. Unique Keys / IDs

Unlike RAPI views which may or may not have a unique key associated with them, all DEAPI core objects have a unique key / constraint that can be applied to the receiving data store (also useful for efficient merge/upsert operations). There are two ways to retrieve these keys:

- Via object metadata (see section 5.2.2). The Key collection will identify those fields that you can use to create your primary key / unique constraint. The Indexes collection can also identify alternate indexes, although not unique, that you may wish to add to your objects.

```
<EntityType Name="transcript_core">
    <Key>
        <PropertyRef Name="reg_num" />
        <PropertyRef Name="transc_object_id" />
        <PropertyRef Name="transc_user_id" />
    </Key>
    <Property Name="_last_touched_dt_utc" Type="Edm.DateTimeOffset"
        Nullable="false">
        <Annotation Term="ExtendedMetadata.FieldPublicName"
            String="_last_touched_dt_utc" />
```

- DEAPI documentation in the DataExporter and Reporting API community within the Cornerstone Success Center (CSC). The Key field(s) are listed above each table that describes the fields of each core object.



Object Public Name:	transcript_core
Heading:	Transcript
Key:	transc_user_id, transc_object_id, reg_num
User defined fields:	No
Description:	Transcript Entries

Field Public Name	Heading	Data Type	OData Type	Description
transc_user_id	*Transcript Internal User ID*	int	[Edm.Int32 Nullable=False]	Unique identifier of the user.
transc_object_id	*Transcript Object ID*	uuid	[Edm.Guid Nullable=False]	Unique identifier of the Training Course Catalog Entry in the transcript.
reg_num	*Registration Number*	int	[Edm.Int32 Nullable=False]	The number of times the user registered for the training.

6.3. Relationship Diagrams

Once individual objects are loaded into the customer's local data store, they can be joined together for further querying and analysis. The DEAPI's $cs_diagram endpoint can generate diagrams that show the relationship between objects in an instant, without the need to navigate lengthy text documents:

```
https://{environment}.csod.com/api/x/objects/$cs_diagram?mode=N&obj= X,Y,Z
```

Two Inputs are required:

- Mode – Choose from Basic, Detail, and Relations.
- Obj – Add the name of the core objects for which you want to see relationships, separated by a comma. Up to 32 objects may be requested per call

Each relationship is represented with an arrow which shows navigation from one object to another object. The object where the arrow originates is considered (To) and the object where the arrow points towards is considered (From) object. If the arrow points to itself, it represents that there is a self-referencing navigation relationship. Each relationship also shows the Multiplicity that relationship represents between the two entities. Multiplicity defines how object participate in a relationship and it is the number of instances of one object related to one instance of the other object.

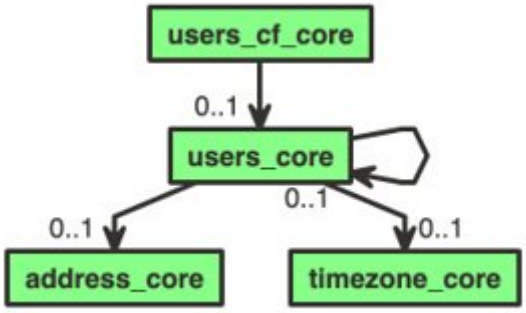
- 0..* - An object instance can have zero or many related instances in another object. for example, A user can have zero or more transcripts.
- 1 - An object instance has exactly 1 related instance in another object. For example, a user can have only one manager.
- 0..1 - An object instance can have zero or one related instance in another object. For example, a user can have zero or one indirect manager.

Relationships are ultimately derived from DEAPI metadata. Therefore, to generate a diagram the calling application should have access to the $metadata endpoint (i.e., the 'obj_metadata:read' scope when configuring application permissions). If you ever forget the relationship diagram URLs, you can inspect each object's metadata to retrieve the ObjectBasicDiagramUrl, the ObjectDetailDiagramUrl and the ObjectRelationsDiagramUrl.

6.3.1. Basic Mode

Example: `https://{environment}.csod.com/api/x/objects/$cs_diagram?`**`mode=basic`**`&obj=users_core,users_cf_core,`

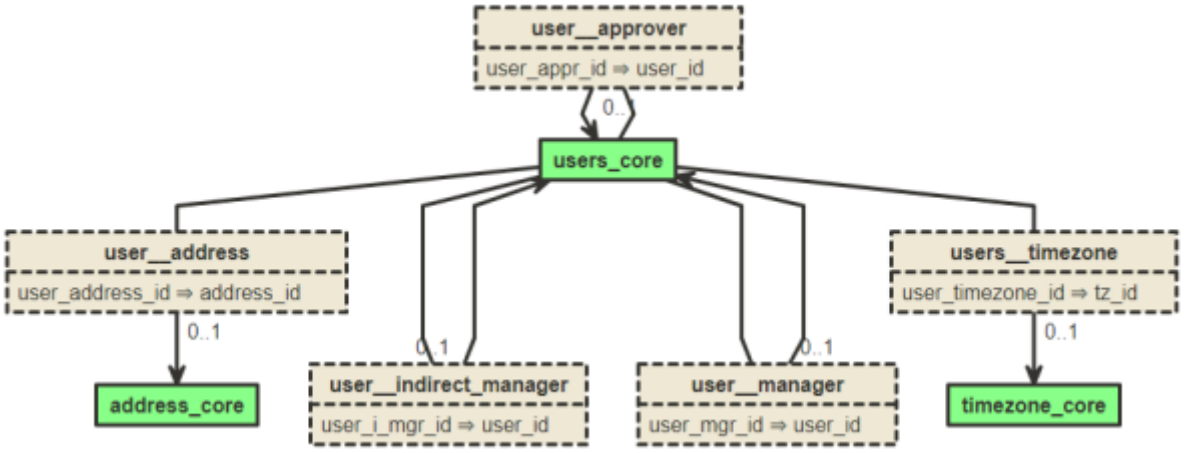
This mode simply illustrates if objects are related. The relationship is illustrated by the presence of a connecting line. Looping lines mean that there is an internal self-reference, meaning that there are fields within the object that are related. In the case of user_core, manager and user are related.



6.3.2. Detail Mode

Example: `https://{environment}.csod.com/api/x/objects/$cs_diagram?`**`mode=detail`**`&obj=users_core,users_cf_core,`

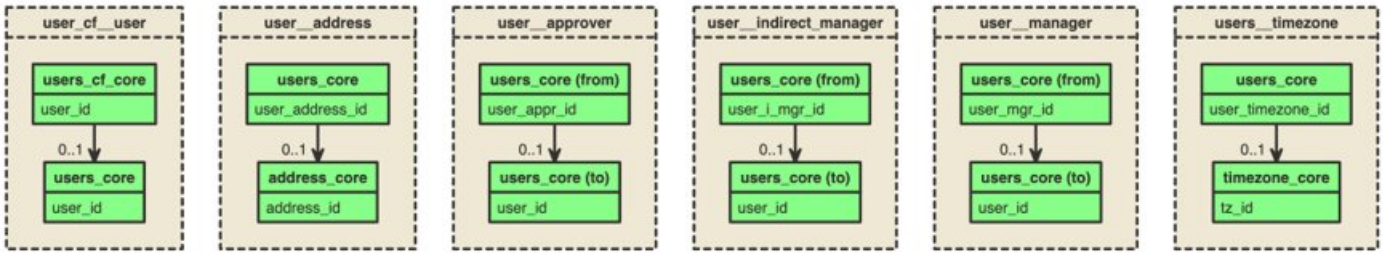
This mode shows every relationship between the selected objects, along with a more human-readable relationship name and the field names that form the relationship.



6.3.3. Relations Mode

Example: `https://{environment}.csod.com /api/x/objects/$cs_diagram?mode=relations&obj= users_core,users_cf_core,`

This mode simply illustrates if objects are related. The relationship is illustrated by the presence of a connecting line. Looping lines mean that there is an internal self-reference, meaning that there are fields within the object that are related. In the case of user_core, manager and user are related.



6.4. Custom Fields

6.4.1. Encoded Field ID

Whenever a custom field is created in the portal, it is assigned a database-generated encoded ID. **Please note this encoded ID may not be the same between environments**. The ID will never change within its respective environment except for copy downs.

The general rule for locating custom fields is by looking for '_cf' appended to the name of the object. For example, "users_cf_core" will contain all the user custom fields. **Please note that custom fields are categorized by type/entity and will not be found in a single object**. To see which objects contain user defined custom fields, please refer to the Data Exporter reference documentation. If an object contains custom fields, it will be categorized as "Yes" under the user defined fields column.

Data Exporter API Objects

Object Public Name	Heading	Description	User defined fields
user_status_core	*User Status*	User Status	No
user_status_local_core	*User Status Localization*	Localizations for user status titles	No
user_type_local_core	*User Type Localization*	Localizations for user types	No
users_cf_core	*User Custom Fields*	User custom fields	Yes
users_core	*User*	Users	No

The custom field Objects include the encoded field ID and the value selected for that field. In order get the mapping between the encoded field ID and the name of the custom field as seen in your CSOD portal, please refer to the metadata for that object. In the metadata, the

portal's common name of the field will be listed under ExtendedMetadata.FieldHeading (see image below).

```
<Property Name="field_1" Type="Edm.String" MaxLength="400">
    <Annotation Term="ExtendedMetadata.FieldPublicName" String="field_1" />
    <Annotation Term="ExtendedMetadata.FieldDescription" String="TBD" />
    <Annotation Term="ExtendedMetadata.FieldHeading" String="Dress Code" />
    <Annotation Term="ExtendedMetadata.FieldOuroborosDataType">
        <EnumMember>Ouroboros.OuroborosDataType/NVarChar</EnumMember>
```

6.4.2. Dropdowns, Multiple Checkboxes, and Branched Hierarchies

For dropdown, radio button, multiple checkbox, and branched hierarchy custom fields, the object may provide you with an ID in your return. To map the ID with the description of the record, you must map the ID to one of the local objects.

7. Appendix

7.1. Frequently Asked Questions

Please search through our repository of FAQs in Edge Answers.

7.2. Acronyms and Abbreviations

API - Application Programming Interface - set of subroutine definitions, protocols, and tools for building application software.

CSC – Client Success Center

CSM – Client Success Manager

CSOD – Cornerstone OnDemand

DEAPI – Data Exporter API

ETL – Extract, Transform, Load – refers to a process in database usage and especially in data warehousing.

GPS – Global Product Support

REST – Representational State Transfer or RESTful web services allows systems to access and manipulate textual representations of web resources using a uniform and predefined set of stateless operations.

RAPI – Reporting API

RDW – Replicated Data Warehouse

RTDW – Real-time Data Warehouse

XML – Extensible Markup Language

_cf – When seen in a view name, this notation typically outlines custom fields within a specific entity

_local – When seen in a view name, this notation typically outlines a localized view providing language translations for a given field name (typically defined by culture_id)

_id – Commonly seen in the database views. This typically refers to a database ID.

_ref – Commonly seen in the database views. This typically refers to an ID located in the portal (Ex. User_ref = User ID)

7.3. Disclaimer