CHARM REST Integration API Introduction

CARM Solutions

Exported on Apr 29, 2022 10:11 AM

Table of Contents

(DRAFT)

The **CHARM Integration API** is the primary integration point for external usage of the CHARM Platform. The REST API exposes methods for authenticating clients and users, creating risk scoring, performing price calculations, generation reports and so on.

The following core functions are exposed by the API;

- Functionality related to authenticating Clients and Users using a token-based authentication scheme.
- The ability to generate and retrieve **CHARM Scorecards** for a given entity.
- Functionality for generating, storing and downloading various **CHARM Reports**.
- Functionality for reading **Metering data** showing aggregated API usage statistics.
- Functionality for retrieving a full **Audit log** that shows a full audit trail of calls being made through the API.
- Functionality for creating, updating and managing **API User and Client accounts.**
- Functionality for working with multiple tenants in a **Multi-tenancy scenario**.
- Functionality for managing **calculation and decision rule settings** for one or more tenants.

Each CHARM deployment consist of two fully separated environments;

- A **Staging** environment that is used for development and staging, as well as a staging area for new production releases.
- A ***Production*** environment that is used to used production versions of data, models and applications.

Note that different CHARM deployments support more or less of the functionality above, for example certain deployments will have the Reports module enabled, but not the Score Card module. In these case related functionality is simply not enabled in the API.

1 Getting started

In order to start using the CARM Integration API the following is needed;

- An **API Base URL** , specific for each CHARM platform deployment. *In this document the API Sandbox URL is used as an example.*
- A valid **CHARM client id and secret** for the deployment, to be used when creating access tokens as described below. U

1.1 The API Base URL

The API Base URL is the full domain name of the CARM deployment for which to access the Integration API, for example **https://charm-sandbox-api.charmsolutions.ai.** This base URL should be used together with the desired API endpoint URL when passing requests to the API.

1.2 Swagger documentation

The API is fully documented using Swagger, which can also be used to test the API.

The Sandbox environment Swagger is reachable at;

https://charm-sandbox-api.charmsolutions.ai/api/v1/ui

2 Authentication

The CHARM Integration API uses *Bearer Token Authentication* (https://tools.ietf.org/html/rfc6750) where all API requests need to be accompanied with a bearer token, an encrypted string of data, in the request Authorization header.

The Charm API supports two different credential schemes;

- *Client-based credentials*, where a general client id and a client secret is used to create a JWT token. This corresponds to the OAuth 2.0 **client_credentials** grant type.
- *User-based credentials* , where a personal user id and password in addition to the client and id is used to create a JWT token. This corresponds to the OAuth 2.0 **password** grant type.

Client-based authentication would be used in cases where the system integrating towards the CHARM API does not require user-level tracking or audit logging of interactions with the CHARM API. Here a token is created for a provided *client id* and *client secret* and then used for all subsequent API calls. The Client id and Client secret credentials thus represents a *calling application.*

User-based authentication would be used in cases where interactive applications such as GUI portals with login functionality should be built on top of the API, and where user-level audit logging is required. Here a token is created for a provided *user email* and *password* and then used for all subsequent API calls. User-based authentication is fully optional.

The two credential schemes can be mixed and matched as fits the integration use case. Corpia will provide Client credentials, but User credentials will be created directly by the API user if needed.

2.1 Retrieving a token

Tokens are retrieved from the only un-protected API endpoint, the **/api/v1/public/auth** endpoint.

Towards this endpoint a POST request is passed with the credentials data sent in the body.

For example, POST:ing;

```
{
 "client_id" : "39ba0351-ad2e-43aa-8090-164fa760b872",
 "client_secret" : "super_secret"
}
```

Will generate an API token for the calling client system, that should then be used in all subsequent API calls.

If a the resulting token should instead represent a user the following body can be sent;

```
{
```

```
"client_id" : "39ba0351-ad2e-43aa-8090-164fa760b872",
 "client_secret" : "super_secret"
 "user_mail" : "bob@acme.com",
 "user_password" : "super_secret_password"
}
```

The resulting token will now represent a user instead of a general client application.

The token generation call will, if successful, return with a HTTP 201 and a body like;

```
{
"status": "success",
"message": "Created token successfully",
"access_token": "<token data>,
"token_type": "bearer",
"expires_in": 1800
}
```

The token can be read from the **access_token** property of the returned json object.

2.2 Passing the token in API requests

A valid bearer token should be passed in the **Authorization header** of all CHARM Integration API calls using the following format;

```
Authorization: Bearer <token>
```

2.3 Token lifetime and refresh

Any token returned from the CHARM Integration API has a lifetime of **30 minutes,** after this it cannot be used to authenticate API requests anymore, if an invalid token is passed the API will return a **403 Forbidden** HTTP status code and an error message.

A token can be refreshed in one of two ways;

1. A new token can be requested again from the **/api/v1/public/auth** endpoint. This can be done whenever.
2. All successful, authenticated API calls automatically return a new, refreshed API token in the HTTP response header **X-token**. This token has new lifetime of 30 minutes and should thus replace the token for the next API call.

Method 2 is useful when developing a conversing API interaction within a user session, such as when a graphical web interface accesses the CHARM integration API. In this scenario an initial token would be created using the api/auth API upon Web interface login, and all subsequent calls would keep using API-refreshed tokens. The 30 minute timeout will then represent Web interface session timeout on inactivity.

3 Creating and retrieving CHARM Scorecards

All operations related to Scoring of entities is collected under the **/api/v1/scores** endpoint. Here methods can be found for creating, listing, monitoring, deleting and retrieving scores.

All CHARM Scores are created using an asynchronous calling scheme;

1. A Score is created using POST call to the **/api/v1/scores** endpoint, passing data needed for the CHARM platform to properly calculated the score. The returned Score object will have a status property value of **CALC**, meaning that is it created, but not yet calculated.
2. The calling application should now poll Score completion status using GET calls to **/api/scores/<score id>** using the internal id retrieved from the previous step. Polling about every second is a good frequency. When Score calculation has successfully completed the resulting Score will have status **DONE**. If the report generation failed the status will instead be **FAIL**, with information about why the generation failed readable thought the **Score.error_info** property.
3. Once the Score is in status **DONE** it can be retrieved via a GET call to the **/api/v1/scores/<report_id>/** resource

3.1 Creating a Score

When creating a Score details basic details on the scored entity and credit is provided, together with any deployment-specific data needed for the CHARM backend in report creation. All information is provided in the POST body in a call to **/api/v1/score.** For example, POST:ing the following;

```
{
 "entity_id":"Acme Inc",
 "application_id":"Credit 2301-2",
 "loan":
 {
 "amount":300000,
 "tenor":24
 }

}
```

Will create a Score for a named entity, for a specific application. In this no extra data was needed. In a deployment where data is not available to the CHARM Platform backend this must instead be passed in with the Score generation call, in the data parameter, for example;

```
{
  "entity_id":"Acme Inc",
  "application_id":"Credit 2301-2",
  "loan":
  {
    "amount":300000,
    "tenor":24
  },

data:{<optional data needed for the Score calculation>}

}
```

Returned from the creation call is a new Score object, for example;

```
{
 "id": "666286f2-f73d-4bb5-8b02-1e92e1451170",
 "entity_id": "Acme Ltd",
 "entity_name": null,
 "created_by": {
 "client_id": "5b89929d-6d8e-43cb-a6b8-3e10f3786370",
 "is_admin": true,
 "user_id": null,
 "user_email": null,
 "user_name": null
 },
"status": "CALC",
"request": {...},
"core_response": {...},
"created": "2021-11-30 18:34:40.640148",
"updated": "2021-11-30 18:34:40.640155",
"error_info": null
}
```

Note here the status of **CALC**. The Score will have this status until it is either **DONE** or **FAIL:ed**. Note also that until the Score is in status **DONE** the detailed response returned in `Score.core_response` will not be available.

The following data can be provided in the POST call;

Parameter	Mandatory	Description
entity_id	Yes	The id of the entity to be scored. This can be anything, for example an internal id, a tax id or some other form of entity identifier. This is typically used to collect Score for the same legal entity together in the CHARM platform.
application_id	Yes	The id of a credit/loan application in the calling organization. This can be anything, and is used to connect a CHARM score to a specific credit/loan application.
loan.amount	Yes	The credit/loan amount
loan.tenor	No	The credit/loan tenor in months

Parameter	Mandatory	Description
data	No	Optional, deployment-specific data needed by the CHARM platform to calculate the Score.

3.2 Retrieving a Score

Retrieving a single Score object, typically when polling for Score calculation status, is done by a GET call towards the **/api/v1/scores** endpoint passing the internal id of a Report object, for example;

```
GET <API base URL>/api/v1/scores/b91865cb-06bf-486f-a719-21c916be45c0
```

3.3 Listing already created Scores

All generated Scores for a deployment can be listed via a GET call to the **/api/v1/scores** endpoint, passing pagination information in the **p_p** and **p_n** parameters, for example;

```
GET <API base URL>/api/vi/scores/?p_p=1,p_n=20
```

Would retrieve the first 20 Scores.

Reports are always returned sorted ascending by the **Score.updated** property.

Score pagination is controlled by two mandatory query parameters;

- **p_n** that sets the limit of returned records for the current page
- **p_p** that sets the page number to start reading page numbers from.

For example, **p_n=1** and **p_p=30** would retrieve the first 30 Scores, and **p_n=3** and **p_p=30** would retrieve Scores number 60-89. If no more Scores are available a truncated or empty result is simply returned.

3.4 Drawing a plot image for a Score

It is possible to generated and download a plot image for a Score in status DONE via a GET call to the **/api/scores/<score id>/plot/<plot type>** endpoint, passing a supported image format, for example;

```
GET <API base URL>/api/v1/scores/666286f2-f73d-4bb5-8b02-
1e92e1451170/plot/insolvency_variable_importance?format=png
```

This will download the plot image as a binary object with relevant HTTP headers for the desired image format.

The following plot types are supported;

Plot type	Description
insolvency_variable_importance	Bar chart showing the Score importance of each selected variable of the Insolvency module

The following headers are set on the returned data;

Image format	content-type header	content-disposition header
png	img/png	attachment ; filename=<name>.png
svg	image/svg+xml	attachment ; filename=<name>.svg

3.5 Deleting a Score

Scores can be removed from the CHARM platform database by making a DELETE call towards the **/api/v1/scores** endpoint, passing the internal id of the Report object to delete, for example;

```
DELETE <API base URL>/api/v1/scores/b91865cb-06bf-486f-a719-21c916be45c0
```

This will permanently delete the Score object.

Note that a token representing a Client or a User of type ADMIN is needed to delete Scores.

4 Rendering and downloading Reports

When working with reports the **/api/v1/reports** endpoint is used, using methods provided reports can be created, listed, downloaded and deleted.

All CHARM Reports are created using an asynchronous calling scheme;

1. A report is created using POST call to the **/api/v1/reports** endpoint, passing the type of report to create and any needed parameters. The returned Report object will have a status property value of **CALC**, meaning that is it created, but not yet ready for downloading as a PDF or Excel.
2. The calling application should now poll report completion status using GET calls to **/api/reports/<report id>** using the internal id retrieved from the previous step. Polling about every second is a good frequency. When report generation has successfully completed the resulting Report will have status **DONE**. If the report generation failed the status will instead be **FAIL**, with information about why the generation failed readable thought the **Report.error_info** property.
3. Once the Report is in status **DONE** is can be downloaded via a GET call to the **/api/v1/reports/<report_id>/download** URL, passing the desired download format.

4.1 Creating a report

When creating a report details on which report, and any needed parameters are passed in the POST body in a call to **/api/v1/reports.** For example, POST:ing the following;

```
{"report_type": "SCORECARD_PDF_OUTPUT",
"report_params": [
 { "name": "score_id", value": "238ef935-66ef-4b89-bd8b-
36415dc591c0"},
 { "name": "nace_code","value": "47"}
]}
```

Will create a Scorecard PDF output report for a specific Scorecard.

Returned from the creation call is a new Report object, for example;

```
{ "id": "b91865cb-06bf-486f-a719-21c916be45c0",
 "status": "CALC",
 "type": "SCORECARD_PDF_OUTPUT",
 "request": "<passed request as json>",
 "created_by": "{'client_id': '39ba0351-ad2e-43aa-8090-164fa760b872',
'is_admin': True}",
“created": "2021-11-04 11:12:16.944674",
 "updated": "2021-11-04 11:12:16.944674" }
```

Note here the status of **CALC**. The Report will have this status until it is either **DONE** or **FAIL:ed**.

4.2 Retrieving a report

Retrieving a single Report object, typically when polling for Report generation status, is done by a GET call towards the **/api/v1/report** endpoint passing the internal id of a Report object, for example;

```
GET <API base URL>/api/vi/report/b91865cb-06bf-486f-a719-21c916be45c0
```

4.3 Listing already created reports

All generated Reports for a deployment can be listed via a GET call to the /api/v1/reports endpoint, passing pagination information in the p_p and p_n parameters, for example;

```
GET <API base URL>/api/vi/reports?p_p=1,p_n=20
```

Would retrieve the first 20 reports.

Reports are always returned sorted ascending by the **Report.updated** property.

Report pagination is controlled by two mandatory query parameters;

- **p_n** that sets the limit of returned records for the current page
- **p_p** that sets the page number to start reading page numbers from.

For example, p_n=1 and p_p=30 would retrieve the first 30 Reports, and p_n=3 and p_p=30 would retrieve Reports number 60-89. If no more Reports are available a truncated or empty result is simply returned.

4.4 Downloading completed reports

Once a Report object has status DONE it can be downloaded via a GET call to the **/api/reports/<report id>/download** endpoint, passing a format supported by the Report, for example;

```
GET <API base URL>/api/vi/report/b91865cb-06bf-486f-a719-
21c916be45c0/download?format=pdf
```

This will download the Report as a binary object with relevant HTTP headers for the desired file type.

The following headers are set on the returned data;

Format	content-type header	content-disposition header
pdf	application/pdf	attachment ; filename=<name>.pdf
excel	application/vnd.openxmlformats-officedocument.spreadsheetml.sheet	attachment ; filename=<name>.xlsx

4.5 Deleting a Report

Reports can be removed by making a DELETE call towards the **/api/v1/report** endpoint, passing the internal id of the Report object to delete, for example;

```
DELETE <API base URL>/api/vi/report/b91865cb-06bf-486f-a719-21c916be45c0
```

This will permanently delete the Report object and any generated report document files.

Note that a token representing a Client or a User of type ADMIN is needed to delete Reports.

5 Working with User accounts

TODO

6 Working with multiple Tenants

TODO

7 Changing calculation settings for a Tenant.

TODO