Hazcheck®
DETECT

Functional Description
Hazcheck Cargo Screening Solution
Confidential

EXIS TECHNOLOGIES
NCB GROUP

Exis Technologies Limited
May 2021

CONTENTS

1. Overview **3**

1.1. Service Assumptions 3

1.2. Microservices Overview 3

2. Description **5**

2.1. Security 5

2.1.1. Authentication 5

2.1.2. Authorisation 5

2.2. Screening 5

2.2.1. Booking 5

2.2.2. Rules 5

2.2.3. Libraries 5

2.2.4. Metadata 5

2.3. Exis Libraries 6

2.3.1. Undeclared Dangerous Goods 6

2.3.2. Mis-declared Dangerous Goods 6

2.4. Sharing Rules 6

2.5. Service Hooks 6

2.6. User Interface to manage Users access, rules and libraries 7

2.7. Case Management 7

2.8. Service Diagram 9

Appendix 1 – Hazcheck Detect Query Syntax **10**

Overview 10

Terms 10

Fields 10

Search Modifiers 10

Fuzzy Searches 10

Proximity Searches 11

Logic Operators 11

AND 11

OR 11

NOT 11

Grouping 11

Nested Objects 12

Nested Collections 13

Appendix 2 - Libraries and Data **14**

Appendix 3 - Example Query Syntax Error! Bookmark not defined.

1. Overview

This document outlines the functionality of Hazcheck Detect (HCD), a Cargo Screening Service solution for the Container Industry.

1.1. Service Assumptions

- HCD is delivered as a hosted service solution, under licence, hosted and maintained by Exis Technologies.
- Container Line system calls the HCD web service, with results exposed to Container Line users only.
- Calls to the HCD web service queue a screening to run, and then return an acknowledgement. Screening will be immediately processed in real time. Screening results will be determined by the keyword/ rules entered by Container Line, Industry group or those provided by Exis.
- Libraries provided by Exis, defined in Appendix.
- Container Line handles all existing internal interfaces.
- HCD cargo screening service is the method for integration of screening results into Container Line internal systems.
- Case Management user interface to allow users to review manage screening results
- Exis is not responsible for following up the results of the screening; this would be part of the container line's own internal processes.
- Exis to provide web user interface so that users can enter and maintain data search terms/ keyword/ rules, such Container Line data is not provided by Exis.
- Where the Industry group agrees common/standard rules/queries these can be made available to all users of HCD.
- Screening input and output to be stored for up to 1 month for diagnostic purposes.

1.2. Microservices Overview

The Hazcheck Detect solution consists of a front-end web service API and backend microservices to process shipment screenings. The front-end API will include endpoints for managing libraries of search queries, for managing users and authorisation, triggering a screen, retrieving results, and configuring service hooks to receive automatic notification of screening results.

A screening is triggered by an API call to the endpoint from a downstream service belonging to the customer. The object is added to an asynchronous queue, and a response is immediately returned, acknowledging receipt of the object to be screened. The acknowledgment includes a unique screening identifier, which can be used to correlate the screen request with the results.
The backend microservices consume objects from the queue in real-time, first preparing the object for processing, then running each of the rules against the object in parallel. The results are collated, and any configured service hooks are called.

The queries will be organised into collections, known as libraries. The primary component of a query will be a piece of text, which conforms to the language syntax. A query can be as simple as a single keyword (which will be searched for in all fields) or as complex as a multi-part search utilising operators such as AND, OR, or NOT.

Regarding data tables and keywords then Exis already provide (and maintain) cargo screening data in the form of IMDG dangerous goods list data including proper shipping names, substance variants, substance index, limited set of technical names and organic peroxides / self-reacting substance data. We also provide and maintain proper shipping names published in the 49CFR US regulations.

A user interface is provided to update and manage keywords rules.

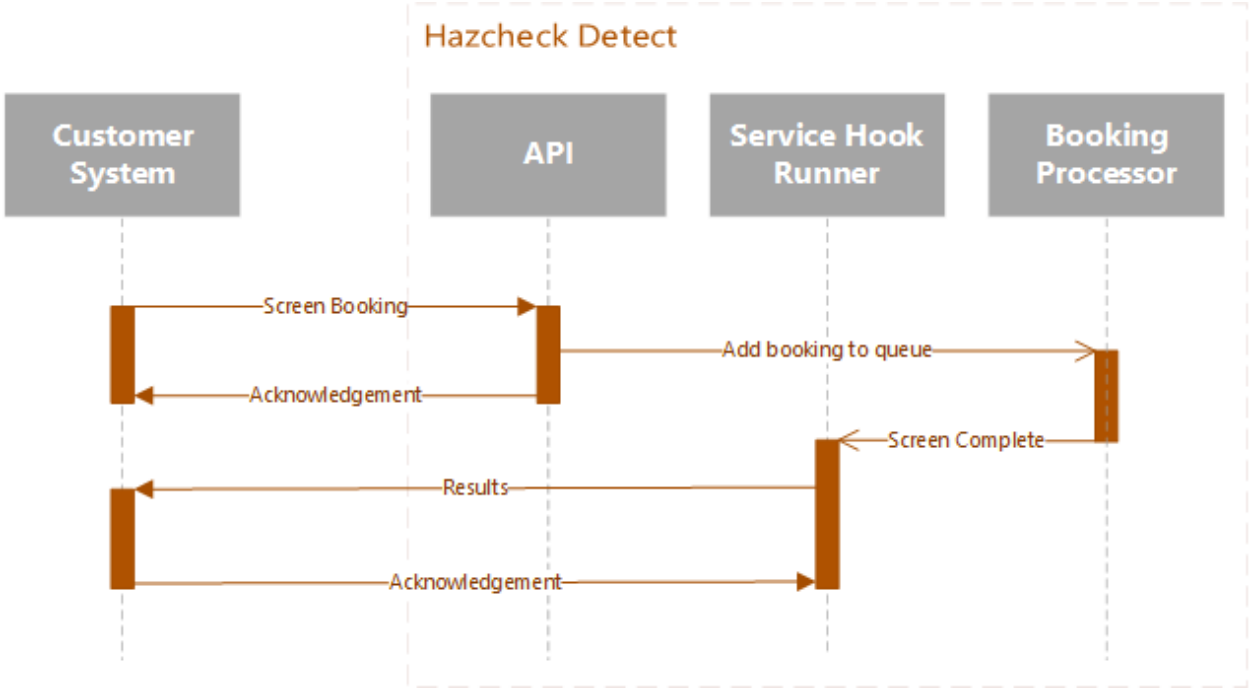


Figure 1:Hazcheck Detect screening process sequence diagram

2. Description

At its core, Hazcheck Detect is an engine for running custom rules against arbitrary objects and returning information about the rules that were triggered, and what caused them to trigger.

2.1. Security

All Hazcheck Detect endpoints are accessible via https only.

2.1.1. Authentication

Machine to machine connections are secured by oauth access tokens. User authentication will be via Single Sign On, with major identity federation protocols supported.

2.1.2. Authorisation

Hazcheck Detect associates a permission with every action. It will be possible to create custom roles with only the permissions you require and assign these roles to users. Additionally, each library will have its own set of permissions that apply only to itself and its rules. In this way you can allow users edit access to the rules of one library, read only access to another library, and no access at all to a third library.

2.2. Screening

2.2.1. Booking

Although Hazcheck Detect can screen any arbitrary object, it is useful to define a minimal structural schema for the booking object, in order that the rules provided by Exis can function. Additionally, any shared or industry wide rules would need to comply with the schema. See the following link for details example of the booking object schema and API calls.

https://api.demo.hl.hazcheckdetect.com/swagger/index.html

2.2.2. Rules

In Hazcheck Detect, rules are the core component of functionality. A rule consists of a friendly name/description and the query, written in the Hazcheck query language. For more information on the Hazcheck query language, see the Appendix for the syntax guide.

2.2.3. Libraries

The organisational unit for collections of rules is the library. It will be possible to create libraries for whatever organisational reasons are required. The name of the library will be returned in the results whenever a rule from the library has been triggered.

2.2.4. Metadata

Both libraries and rules can contain metadata – that is, custom, flexible additional information that is not used or required by Hazcheck Detect itself but would be useful as part of the downstream processing. Some examples include priority, categorisation information, etc.

A hit result includes all metadata as a single collection, regardless of whether it came from the rule itself or the library containing the rule. One exception to this is when the library and the rule contain the same metadata entry – in these cases the rule value is used.
This means you can apply a piece of metadata once, on the library, and be confident that it will be returned when any of the contained rules are triggered. An example use case for this scenario is that you want all rules in a library to all be marked as “priority: Medium”, but you don’t want to have to set this value individually on each rule.
You can also override a library metadata value by applying the same piece of metadata to an individual rule. For example, in your library from the previous example, there may be a few rules that should be “priority: High”, instead of the default “priority: Medium”.

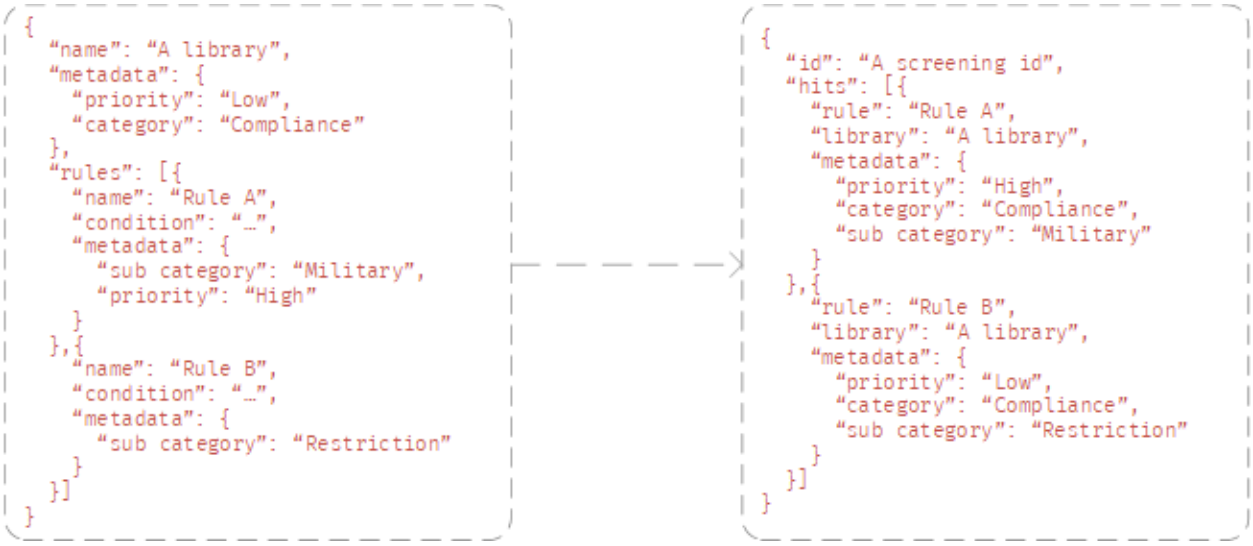


Figure 2: The mapping of metadata values from libraries and rules to results.

2.3. Industry Libraries

Exis will provide built-in rules that will be organised into the following libraries;

2.3.1. Undeclared Dangerous Goods

The undeclared DG rules would focus on items that are *not* declared as DG, trying to identify items that *should* have been declared as DG.

2.3.2. Mis-declared Dangerous Goods

The mis-declared DG rules would focus on items that *are* declared as DG, trying to identify items that have not been identified as the *correct* DG.

2.4. Sharing Rules (under development)

Hazcheck Detect will have a feature allowing Container Lines to share selected libraries of rules with other Container Lines using Hazcheck Detect (or their own screening system) and to consume libraries shared by those other Container Lines.

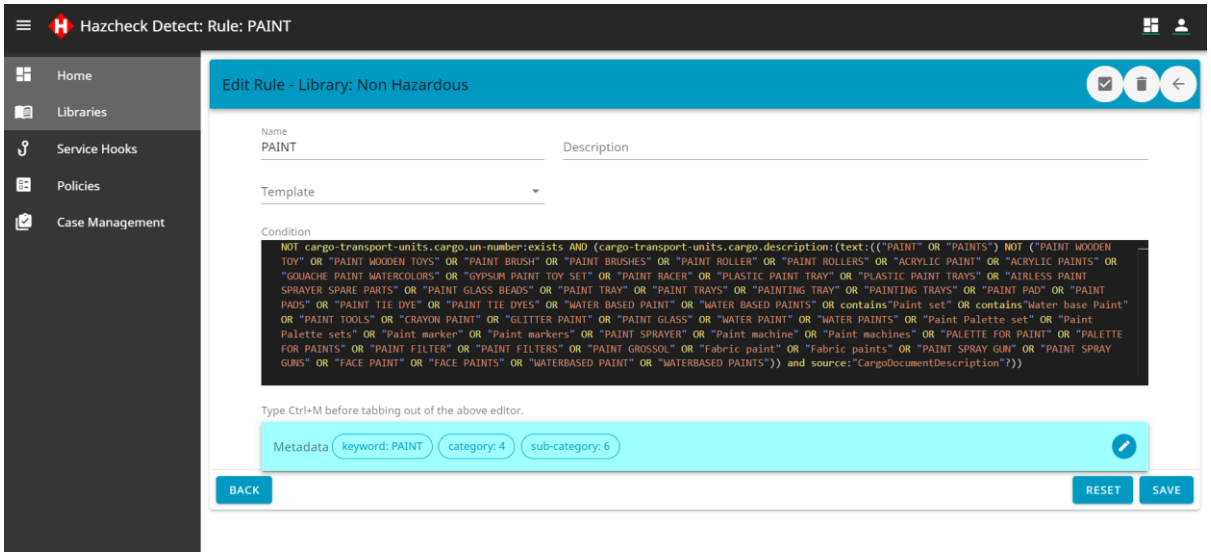
2.5. Service Hooks

Screen results will be available from an API endpoint once the cargo screen has completed, however it will also be possible to configure service hooks to notify when a cargo screen is complete. The service hook will also send the cargo screen results (hits)

(users can consume the results via the API into their own applications or view the results in the Hazcheck Detect case management user interface, see 2.7)

2.6. User Interface to manage Users access, rules and libraries

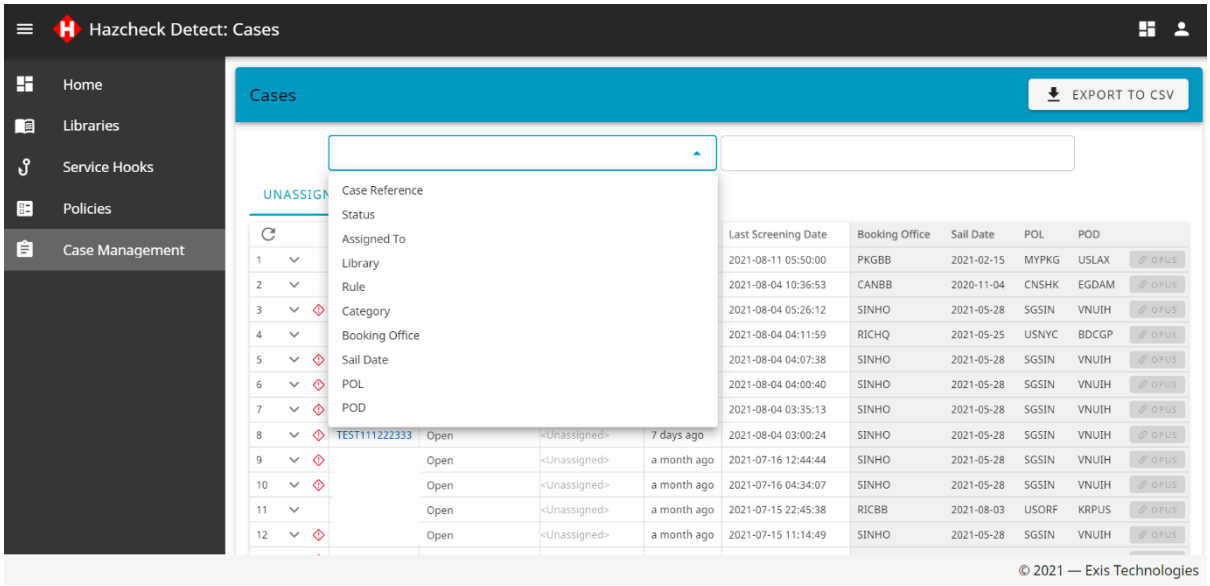
Hazcheck Detect has a user interface for managing users, rules/ libraries (although rules can also be updated via the API). The user interface enables users to create and maintain libraries, rules, and service hooks.
Rule queries can be created using a rich text editor, with autocomplete, syntax highlighting etc., or via a graphical interface utilising select boxes.

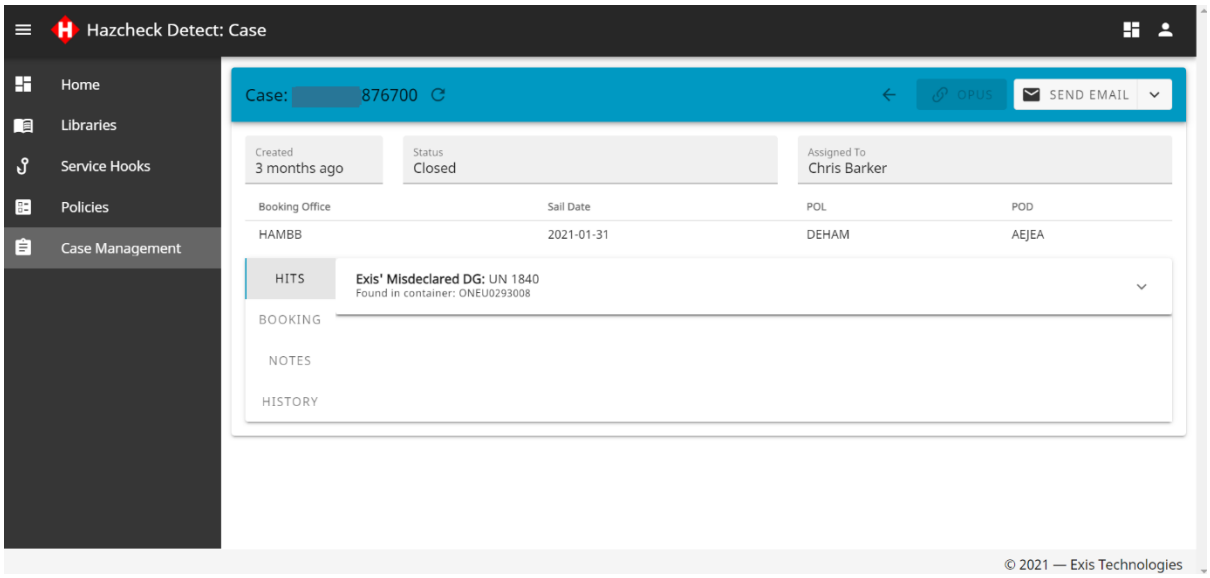


2.7. Case Management

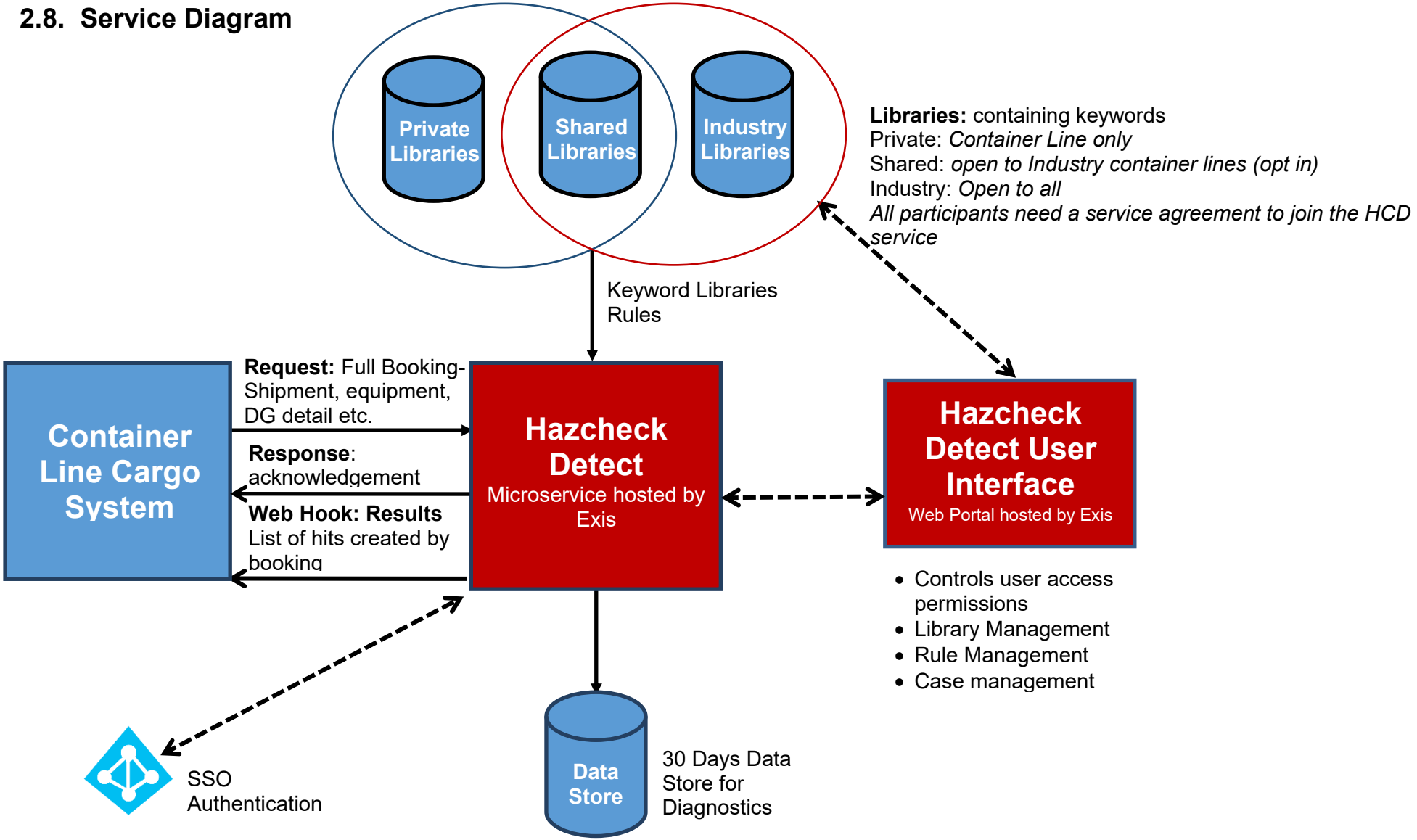
Hazcheck Detect includes a user interface (end point) for users to view and actions hits returned for booking screened by the service.
Features:,

- Filter hits by various criteria e.g. sailing date, status, user assigned, rule, port etc.
- Assign hits to appropriate user.
- Update status and record actions
- Link to Customer systems and create emails to customer service teams
- View details of hit showing search term found and location.
- View history of case and previous screening for booking.
- Export hits data to csv.


Hazcheck Detect: Cases
Home
Libraries
Service Hooks
Policies
Case Management
Cases
EXPORT TO CSV
Case Reference
Status
Assigned To
Library
Rule
Category
Booking Office
Sail Date
POL
POD
UNASSIGN
Last Screening Date
Booking Office
Sail Date
POL
POD
2021-08-11 05:50:00 PKGBB 2021-02-15 MYPKG USLAX
2021-08-04 10:36:53 CANBB 2020-11-04 CNSHK EGDAM
2021-08-04 05:26:12 SINHO 2021-05-28 SGSIN VNUIH
2021-08-04 04:11:59 RICHQ 2021-05-25 USNYC BDCGP
2021-08-04 04:07:38 SINHO 2021-05-28 SGSIN VNUIH
2021-08-04 04:00:40 SINHO 2021-05-28 SGSIN VNUIH
2021-08-04 03:35:13 SINHO 2021-05-28 SGSIN VNUIH
TEST111222333 Open <Unassigned> 7 days ago 2021-08-04 03:00:24 SINHO 2021-05-28 SGSIN VNUIH
Open <Unassigned> a month ago 2021-07-16 12:44:44 SINHO 2021-05-28 SGSIN VNUIH
Open <Unassigned> a month ago 2021-07-16 04:34:07 SINHO 2021-05-28 SGSIN VNUIH
Open <Unassigned> a month ago 2021-07-15 22:45:38 RICBB 2021-08-03 USORF KRPUS
Open <Unassigned> a month ago 2021-07-15 11:14:49 SINHO 2021-05-28 SGSIN VNUIH
© 2021 — Exis Technologies


Hazcheck Detect: Case
Home
Libraries
Service Hooks
Policies
Case Management
Case: 876700
OPUS
SEND EMAIL
Created
3 months ago
Status
Closed
Assigned To
Chris Barker
Booking Office
Sail Date
POL
POD
HAMBB
2021-01-31
DEHAM
AEJEA
HITS
Exis' Misdeclared DG: UN 1840
Found in container: ONEU0293008
BOOKING
NOTES
HISTORY
© 2021 — Exis Technologies

2.8. Service Diagram



Appendix 1 – Hazcheck Detect Query Syntax

Overview

This document outlines the syntax to use for queries/rules in the Hazcheck Detect engine.

Terms

A query is broken up into terms and operators. A term is a group of words surrounded by double quotes such as "Hazcheck detect". Multiple terms can be combined with Boolean operators to form a more complex query.

Fields

When performing a search, you can specify a field to search in. You can search any field by typing the field name followed by a colon, then the term you are looking for.
As an example, using the following object:

```
{
  name: "Hazcheck Detect",
  description: "A cargo screening engine."
}
```

If you want to return a hit for this object, you can enter:

```
name:="Hazcheck Detect"
```

or

```
description:"engine"
```

Search Modifiers

Hazcheck Detect supports several special modifiers to searches.

Fuzzy Searches

Hazcheck Detect supports fuzzy searches based on the Damerau-Levenshtein distance metric. Fuzzy searches are enabled using the ~ operator at the end of a term.
For example, to search for words *like* "Hazcheck" in the *description* field, you can use:

```
description:"Hazcheck"~
```

It is possible to specify the degree of similarity required by providing a value between 0 and 1. The closer the value is to 1, the more similar matches must be.
For example, the following could match "Hazcheck" and "Hascheck", but not "Hassschec".

```
description:"Hazcheck"~0.8
```

The default value will be 0.5.

Proximity Searches

Hazcheck Detect supports finding words that are a specific distance away. Proximity searches are enabled using the @ operator at the end of a phrase.
For example, to match the words "Hazcheck" and "Detect" when they are no more than 3 words apart, you can use:

```
description:"Hazcheck Detect"@3
```

Logic Operators

Hazcheck Detect supports the following Boolean operators.

AND

To search for hits where both terms must exist, you can use the AND operator. For example:

```
name:"Hazcheck" AND description:"Hazcheck Detect"
```

will return only hits where the *name* field contains the word "Hazcheck" *and* the *description* field contains the phrase "Hazcheck Detect".

OR

To search for hits where either term must exist, you can use the OR operator. For example:

```
name:"Hazcheck" OR description:"Hazcheck Detect"
```

will return hits where either the *name* field contains the word "Hazcheck" *or* the *description* field contains the phrase "Hazcheck Detect".

NOT

To search for hits while explicitly excluding certain hits, you can use the NOT operator. For example:

```
name:"Hazcheck" NOT description:"Hazcheck Detect"
```

will return hits where the *name* field contains the word "Hazcheck" but the *description* field does *not* contain the phrase "Hazcheck Detect".

Grouping

Hazcheck Detect supports grouping statements into subqueries using parentheses.
For example, to search for hits where the name field contains either "Hazcheck" or "Detect" and the description field also contains "engine", you can use:

```
(name:"Hazcheck” OR name:"Detect") AND description:"engine"
```

or, using parentheses around only the search terms

```
name:("Hazcheck" OR "Detect") AND description:"engine"
```

Nested Objects

Hazcheck Detect supports searching a complex object with a hierarchy of properties and children. The complex objects will be "flattened" allowing nested properties to be accessed using the common "dotted notation".
For example: an object like this:

```
{
  product: {
    name: "Hazcheck Detect",
    description: "A cargo screening engine."
  },
  creator: {
    name: "Exis Technologies"
  }
}
```

can be queried as follows:

```
product.name:"Hazcheck"
```

or

```
product.description:"cargo screening"
```

or

```
creator.name:"Exis Technologies"
```

Nested Collections

Hazcheck Detect supports searching collections of complex objects with a hierarchy of properties and children, some of which may be additional collections. Given the following object:

```
{
  stages: [
    {
      type: "Load",
      vessel: {
        code: "EX1",
        name: "Exis Trinity",
        operator: {
          code: "EXS",
          name: "Exis Shipping"
        }
      },
      port: {
        code: "GBTEE",
        name: "Teesport",
        country: {
          code: "GB",
          name: "United Kingdom"
        }
      }
    },
    {
      type: "Unload",
      vessel: {
        code: "EX1",
        name: "Exis Trinity",
        operator: {
          code: "EXS",
          name: "Exis Shipping"
        }
      },
      port: {
        code: "DKCPH",
        name: "Copenhagen",
        country: {
          code: "DK",
          name: "Denmark"
        }
      }
    }
  ]
}
```

Properties on collections will be accessible using the dotted notation used in the complex objects example.

```
stages.type:="Load"
```

or

```
stages.port.country.code:="DK"
```

However, if you wanted to query multiple fields you would run the risk of matching different objects within a collection.
For example, the following query would match the example object, because it does not specify that both matches should be on the same child collection object.

```
stages.type:="Load" AND stages.port.country.code:="DK"
```

To get around this, you can group a sub-query around the properties of a nested object, to ensure the matches are on the same child object.
The following query would not match the example object. Notice the omission of stages from the start of each field identifier, because we've already specified the stages collection.

```
stages:(type:="Load" AND port.country.code:="DK")
```

Appendix 2 - Libraries and Data

As part of the project, there are some checks that require Exis' expertise. For the most part, these will be delivered as standard query libraries, exactly as our customers could create, but some checks may need additional features.

Undeclared DG
These checks would focus on cargo that is *not* declared as DG, looking for suspicious items that perhaps *should* be declared as DG.

Misdeclared DG
These checks would focus on cargo which *is* declared as DG, but perhaps may not have been declared as the correct DG.

Complex Queries
Most queries should be viable using the query syntax, but some may have additional complexity, or may require the ability to lookup external data, such as DG info, substance indexes, etc.

Data
The following Data will be held in a standard library to be updated and maintained by Exis – all other Libraries to be updated and supplied by users (Customer Data)

- Proper shipping names
- Substance variants
- Substance index names

- Limited set of technical names organic peroxides / self-reacting substance data