query-all HTTP/TLS/JSON

Interface Design Description

Abstract
This document describes a HTTP protocol with TLS payload security and JSON payload encoding variant of the **query-all** service.

Contents

1 Overview

This document describes the **query-all** service interface, which is enables the dedicated core systems to get all the service instance data. It's implemented using protocol, encoding as stated in the following table:

Profile ype	Type	Version
Transfer protocol	HTTP	1.1
Data encryption	TLS	1.3
Encoding	JSON	RFC 8259 [1]
Compression	N/A	-

Table 1: Communication and sematics details used for the **query-all** service interface

This document provides the Interface Design Description IDD to the *query-all – Service Description* document. For further details about how this service is meant to be used, please consult that document.

The rest of this document describes how to realize the query-all service HTTP/TLS/JSON interface in details.

2 Interface Description

The service responses with the status code `200 Ok` if called successfully. The error codes are, `400 Bad Request` if request is malformed, `401 Unauthorized` if improper client side certificate is provided, `500 Internal Server Error` if Service Registry is unavailable.

```
GET /serviceregistry/query/all HTTP/1.1
```

Listing 1: A query-all invocation.

```
{
  "data": [
    {
     "id": 0,
     "serviceDefinition": {
       "id": 0,
       "serviceDefinition": "string",
       "createdAt": "string",
       "updatedAt": "string"
     },
     "provider": {
       "id": 0,
       "systemName": "string",
       "address": "string",
       "port": 0,
       "authenticationInfo": "string",
       "createdAt": "string",
       "updatedAt": "string"
     },
     "serviceUri": "string",
     "endOfValidity": "string",
     "secure": "CERTIFICATE",
     "metadata": {
       "additionalProp1": "string",
       "additionalProp2": "string",
       "additionalProp3": "string"
     },
     "version": 0,
     "interfaces": [
       {
         "id": 0,
         "interfaceName": "string",
         "createdAt": "string",
         "updatedAt": "string"
       }
     ],
     "createdAt": "string",
     "updatedAt": "string"
    }
  ],
  "count": 0
}
```

Listing 2: A query-all response.

3 Data Models

Here, all data objects that can be part of the service calls associated with this service are listed in alphabetic order. Note that each subsection, which describes one type of object, begins with the *struct* keyword, which is meant to denote a JSON Object that must contain certain fields, or names, with values conforming to explicitly named types. As a complement to the primary types defined in this section, there is also a list of secondary types in Section 3.7, which are used to represent things like hashes, identifiers and texts.

3.1 struct ServiceRegistryList

Field	Type	Description
data	List<ServiceRegistryResponse>	List of service instances
count	Number	Size of the result list.

3.2 struct ServiceRegistryResponse

Field	Type	Description
createdAt	DateTime	Service instance record was created at this UTC timestamp.
endOfValidity	DateTime	Service is available until this UTC timestamp.
id	Number	Identifier of the service instance.
interfaces	List<InterfaceRecord>	List of interfaces the service supports.
provider	SystemRecord	Descriptor of the provider system record.
secure	SecureType	Type of security the service uses.
serviceDefinitionResponse	ServiceDefinitionRecord	Descriptor of the serviceDefinition record.
serviceUri	String	Path of the service on the provider.
metadata	Metadata	Service metadata.
updatedAt	DateTime	Service instance record was modified at this UTC timestamp.
version	Version	Version of the service.

3.3 struct Metadata

An Object which maps String key-value pairs.

3.4 struct InterfaceRecord

Field	Type	Description
createdAt	DateTime	Interface instance record was created at this UTC timestamp.
id	Number	Identifier of the interface instance.
interfaceName	Interface	Specified name of the interface.
updatedAt	DateTime	Interface instance record was modified at this UTC timestamp.

3.5 struct SystemRecord

Field	Type	Description
address	Address	Network address of the system.
authenticationInfo	String	X.509 public key of the system.
createdAt	DateTime	System instance record was created at this UTC timestamp.
id	Number	Identifier of the system instance.
metadata	Metadata	Additional information about the system.
port	PortNumber	Port of the system.
systemName	Name	Name of the system.
updatedAt	DateTime	System instance record was modified at this UTC timestamp.

3.6 struct ServiceDefinitionRecord

Field	Type	Description
createdAt	DateTime	Service definition instance record was created at this UTC timestamp.
id	Number	Identifier of the service definition instance.
serviceDefinition	Name	Name of the service definition.
updatedAt	DateTime	Service definition instance record was modified at this UTC timestamp.

3.7 Primitives

As all messages are encoded using the JSON format [2], the following primitive constructs, part of that standard, become available. Note that the official standard is defined in terms of parsing rules, while this list only concerns syntactic information. Furthermore, the Object and Array types are given optional generic type parameters, which are used in this document to signify when pair values or elements are expected to conform to certain types.

JSON Type	Description
Value	Any out of Object, Array, String, Number, Boolean or Null.
Object <A>	An unordered collection of [String: Value] pairs, where each Value conforms to type A.
Array <A>	An ordered collection of Value elements, where each element conforms to type A.
String	An arbitrary UTF-8 string.
Number	Any IEEE 754 binary64 floating point number [3], except for *+Inf*, *-Inf* and *NaN*.
Boolean	One out of `true` or `false`.
Null	Must be `null`.

With these primitives now available, we proceed to define all the types specified in the **query-all** SD document without a direct equivalent among the JSON types. Concretely, we define the **query-all** SD primitives either as *aliases* or *structs*. An *alias* is a renaming of an existing type, but with some further details about how it is intended to be used. Structs are described in the beginning of the parent section. The types are listed by name in alphabetical order.

3.7.1 alias Address = String

A string representation of a network address. An address can be a version 4 IP address (RFC 791), a version 6 IP address (RFC 2460) or a DNS name (RFC 1034).

3.7.2 alias DateTime = String

Pinpoints a moment in time in the format of ISO8601 standard ”yyyy-mm-ddThh:mm:ss”, where ”yyy” denotes year (4 digits), ”mm” denotes month starting from 01, ”dd” denotes day starting from 01, ”T” is the separator between date and time part, ”hh” denotes hour in the 24-hour format (00-23), ”MM” denotes minute (00-59), ”SS” denotes second (00-59). ” ” is used as separator between the date and the time. An example of a valid date/time string is ”2020-12-05T12:00:00”

3.7.3 alias List <A> = Array<A>

There is no difference.

3.7.4 alias id = Number

An identifier generated for each Object that enables to distinguish them and later to refer to a specific Object.

3.7.5 alias Interface = String

A String that describes an interface in *Protocol-SecurityType-MimeType* format. *SecurityType* can be SECURE or INSECURE. *Protocol* and *MimeType* can be anything. An example of a valid interface is: ”HTTPS-SECURE-JSON” or ”HTTP-INSECURE-SENML”.

3.7.6 alias Name = String

A String indentifier that is intended to be both human and machine-readable.

3.7.7 alias PortNumber = Number

Decimal Number in the range of 0-65535.

3.7.8 alias SecureType = String

A String that describes an the security type. Possible values are *NOT_SECURE* or *CERTIFICATE* or *TOKEN*.

3.7.9 alias Version = Number

A Number that represents the version of the service. And example of a valid version is: 1.

4 References

[1] T. Bray, “The JavaScript Object Notation (JSON) Data Interchange Format,” RFC 8259, Dec. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8259.txt

[2] ——, “The JavaScript Object Notation (JSON) Data Interchange Format,” RFC 7159, 2014, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC7159

[3] M. Cowlishaw, “IEEE Standard for Floating-Point Arithmetic,” *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, July 2019. [Online]. Available: https://doi.org/10.1109/IEEESTD.2019.8766229

5 Revision History

5.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1	YYYY-MM-DD	4.6.0		Xxx Yyy

5.2 Quality Assurance

No.	Date	Version	Approved by
1	YYYY-MM-DD	4.6.0	Xxx Yyy