

# service-register HTTP/TLS/JSON

## Interface Design Description

### Abstract

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

## Contents

<b>1 Overview</b>	<b>3</b>
<b>2 Interface Description</b>	<b>4</b>
<b>3 Data Models</b>	<b>6</b>
3.1 struct <i>ServiceRegistryRequest</i> . . . . .	6
3.2 struct <i>ServiceRegistryResponse</i> . . . . .	6
3.3 struct <i>Metadata</i> . . . . .	6
3.4 Primitives . . . . .	6
<b>4 References</b>	<b>8</b>
<b>5 Revision History</b>	<b>9</b>
5.1 Amendments . . . . .	9
5.2 Quality Assurance . . . . .	9

# 1 Overview

This document describes the **service-register** service interface, which enables autonomous service registration. It's implemented using protocol, encoding as stated in below table:

Profile type	Type	Version
Transfer protocol	HTTP	1.1
Data encryption	TLS	1.3
Encoding	e.g. JSON	RFC 8259 [1]
Compression	N/A	-
Semantics	e.g. SensML	RFC 9100 [2]
Ontology	N/A	-

Table 1: Communication and semantics details used for the **service-register** service interface

This document provides the Interface Design Description IDD to the *service-register – 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 service-register service interface in detail.

## 2 Interface Description

The service responds with the status code 201 Created 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.

```
1 POST /serviceregistry/register HTTP/1.1
2
3 {
4   "endOfValidity": "2020-12-05 12:00:00",
5   "interfaces": [
6     "HTTP-SECURE-JSON"
7   ],
8   "metadata": {
9     "unit": "celsius"
10  },
11  "providerSystem": {
12    "address": "192.168.0.101",
13    "authenticationInfo": "public key of the client certificate",
14    "port": 8080,
15    "systemName": "exampleprovider"
16  },
17  "secure": "TOKEN",
18  "serviceDefinition": "temperature",
19  "serviceUri": "/",
20  "version": 1
21 }
```

Listing 1: A [service-register](#) invocation.

```
1 {
2   "id": 14,
3   "serviceDefinition": {
4     "id": 13,
5     "serviceDefinition": "temperature",
6     "createdAt": "2020-12-01 11:59:10",
7     "updatedAt": "2020-12-01 11:59:10"
8   },
9   "provider": {
10    "id": 4,
11    "systemName": "exampleprovider",
12    "address": "192.168.0.101",
13    "port": 8080,
14    "authenticationInfo": "public key of the client certificate",
15    "createdAt": "2020-12-01 11:59:10",
16    "updatedAt": "2020-12-01 11:59:10"
17  },
18  "serviceUri": "/",
19  "endOfValidity": "2020-12-05 12:00:00",
20  "secure": "TOKEN",
21  "metadata": {
22    "unit": "celsius"
23  },
24  "version": 1,
25  "interfaces": [
26    {
27      "id": 1,
28      "interfaceName": "HTTP-SECURE-JSON",
29      "createdAt": "2019-10-07 04:29:51",
30      "updatedAt": "2019-10-07 04:29:51"
31    }
32  ],
33  "createdAt": "2020-12-01 11:59:10",
34  "updatedAt": "2020-12-01 11:59:10"
35 }
```

---

Listing 2: A [service-register](#) response. Every Object contains an id.

## 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 3.4, which are used to represent things like hashes, identifiers and texts.

### 3.1 struct **ServiceRegistryRequest**

This structure is used to register a service offering into the Service Registry.

Object Field	Value Type	Description
"endofValidity"	DateTime	Service is available until this UTC timestamp.
"interfaces"	Array<Interface>	List of interfaces the service supports.
"metadata"	Metadata	Metadata
"providerSystem"	Name	Name of the provider system.
"secure"	SecureType	Type of security the service uses.
"serviceDefinition"	Name	Service Definition.
"serviceUri"	URI	URI of the service.
"version"	Version	Version of the service.

### 3.2 struct **ServiceRegistryResponse**

TODO

### 3.3 struct **Metadata**

A JSON Object which maps String key-value pairs.

### 3.4 Primitives

As all messages are encoded using the JSON format [3], 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.

With these primitives now available, we proceed to define all the types specified in the Service Discovery Register SD document without a direct equivalent among the JSON types. Concretely, we define the Service Discovery Register 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.

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 [4], except for <i>+Inf</i> , <i>-Inf</i> and <i>NaN</i> .
Boolean	One out of <code>true</code> or <code>false</code> .
Null	Must be <code>null</code> .

### 3.4.1 alias DateTime = String

Pinpoints a moment in time in the format of "YYYY-MM-DD HH:mm:ss", where "YYYY" denotes year (4 digits), "MM" denotes month starting from 01, "DD" denotes day starting from 01, "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 "2020-12-05 12:00:00"

### 3.4.2 alias id = Number

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

### 3.4.3 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".

### 3.4.4 alias Name = String

A String that is meant to be short (less than a few tens of characters) and both human and machine-readable.

### 3.4.5 alias SecureType = String

A String that describes an the security type. Possible values are *NOT\_SECURE* or *CERTIFICATE* or *TOKEN*.

### 3.4.6 alias URI = String

A String that represents the URL subpath where the offered service is reachable, starting with a slash ("/"). An example of a valid URI is `"/temperature"`.

### 3.4.7 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] C. Bormann, "Sensor Measurement Lists (SenML) Features and Versions," RFC 9100, Aug. 2021. [Online]. Available: <https://rfc-editor.org/rfc/rfc9100.txt>
- [3] T. Bray, "The JavaScript Object Notation (JSON) Data Interchange Format," RFC 7159, 2014, RFC Editor. [Online]. Available: <https://doi.org/10.17487/RFC7159>
- [4] 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	2020-12-05	4.4.0		Szvetlin Tanyi
2	2021-07-14	4.4.0	Minor edits	Jerker Delsing
3	2022-01-17	4.4.0	Minor edits	Jerker Delsing

### 5.2 Quality Assurance

No.	Date	Version	Approved by
1	2022-01-17	4.4.0	Jerker Delsing