Document title
Service Discovery Register HTTP/TLS/JSON
Date
2021-01-15
Author
Szvetlin Tanyi
Contact
szvetlin@aitia.ai

Document type
IDD
Version
4.3.0
Status
RELEASE
Page
1 (9)

Service Discovery Register HTTP/TLS/JSON

Interface Design Description

Service ID: *"register"*

Abstract

This document describes a HTTP/TLS/JSON variant of the Service Discovery Register service.

ARTEMIS Innovation Pilot Project: Arrowhead
THEME [SP1-JTI-ARTEMIS-2012-AIPP4 SP1-JTI-ARTEMIS-2012-AIPP6]
[Production and Energy System Automation Intelligent-Built environment and urban infrastructure for sustainable and friendly cities]

Contents

1 Overview

This document describes the HTTP/TLS/JSON variant of the Service Discovery Register Eclipse Arrowhead service, which is enables autonomous service registration by systems. Examples of this interaction is a system that has the capability to provide some kind of service. To enable other systems to use, to consume it, this service needs to be offered in the Service Registry.

This document exists as a complement to the *Service Discovery 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 Discovery Register service using HTTP [1], TLS [2] and JSON [3], both in terms of its functions (Section 2) and its information model (Section 3).

2 Service Functions

This section lists the functions that must be exposed by the Service Discovery Register service in alphabetical order. In particular, each subsection first names the HTTP method and path used to call the function, after which it names an abstract function from the Service Discovery Register SD document, as well as input and output types. All functions in this section respond with the HTTP 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.

2.1 POST /serviceregistry/register

Interface: Register
Input: ServiceRegistryRequest

Called to register a service offered by the caller system, as exemplified in Listing 1.

```
POST /serviceregistry/register HTTP/1.1

{
  "endOfValidity": "2020-12-05 12:00:00",
  "interfaces": [
    "HTTP-SECURE-JSON"
  ],
  "metadata": {
    "unit": "celsius"
  },
  "providerSystem": {
    "address": "192.168.0.101",
    "authenticationInfo": "public key of the client certificate",
    "port": 8080,
    "systemName": "exampleprovider"
  },
  "secure": "TOKEN",
  "serviceDefinition": "temperature",
  "serviceUri": "/",
  "version": 1
}
```

Listing 1: A Register invocation.

```
{
  "id": 14,
  "serviceDefinition": {
    "id": 13,
    "serviceDefinition": "temperature",
    "createdAt": "2020-12-01 11:59:10",
    "updatedAt": "2020-12-01 11:59:10"
  },
  "provider": {
    "id": 4,
    "systemName": "exampleprovider",
    "address": "192.168.0.101",
    "port": 8080,
    "authenticationInfo": "public key of the client certificate",
    "createdAt": "2020-12-01 11:59:10",
    "updatedAt": "2020-12-01 11:59:10"
  },
  "serviceUri": "/",
  "endOfValidity": "2020-12-05 12:00:00",
  "secure": "TOKEN",
  "metadata": {
    "unit": "celsius"
  },
  "version": 1,
  "interfaces": [
    {
      "id": 1,
      "interfaceName": "HTTP-SECURE-JSON",
      "createdAt": "2019-10-07 04:29:51",
      "updatedAt": "2019-10-07 04:29:51"
    }
  ],
  "createdAt": "2020-12-01 11:59:10",
  "updatedAt": "2020-12-01 11:59:10"
}
```

Listing 2: A Register response. Every Object contains an id.

3 Information Model

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 3.3, 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 Metadata

A JSON Object which maps String key-value pairs.

3.3 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.

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 *+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 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.

3.3.1 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"

3.3.2 alias id = Number

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

3.3.3 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".

3.3.4 alias Name = String

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

3.3.5 alias SecureType = String

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

3.3.6 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".

3.3.7 alias Version = Number

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

4 References

[1] R. Fielding and J. Reschke, “Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing,” RFC 7230, 2018, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC7230

[2] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” RFC 8446, 2018, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC8446

[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	1.0.0		Szvetlin Tanyi

5.2 Quality Assurance

No.	Date	Version	Approved by
1			