SWIM Discovery Service (SDS) Implementation Specification, Version 2.0.0, Working Draft

U.S. Federal Aviation Administration (FAA),
System Wide Information Management (SWIM)

March 2024

Notice

Executive Summary

This document, “SWIM Discovery Service (SDS) Implementation Specification, Version 2.0.0, Working Draft,” is a technical specification developed by the United States Federal Aviation Administration (FAA) SWIM program in collaboration with the APAC SWIM SDS Project Team. It establishes guidelines and technical principles for the implementation of SWIM Discovery Services (SDS) that facilitate the federated discovery of aviation services within the global SWIM ecosystem. The specification aims to promote interoperability, standardization, and seamless service metadata exchange across various SWIM initiatives and stakeholders.

The initial version of this specification, developed in 2020 by the United States FAA SWIM program in collaboration with the Korea Airports Corporation (KAC) of the Republic of Korea (ROK), has been implemented by various SWIM initiatives across the United States, ROK, Japan, and the People's Republic of China (PRC). Feedback and operational insights gathered from these international implementations have prompted the development of SDS Implementation Specification version 2.0.0.

This updated version introduces several significant advancements and refinements to the SDS framework, including:

- **Streamlined Data Exchange Pattern**: Version 2.0 simplifies the data exchange processes, enhancing efficiency and reducing complexity in service metadata exchange.
- **Usage of HATEOAS**: Hypermedia as the Engine of Application State (HATEOAS) principles are now integrated, providing dynamic navigability and discoverability within the SDS API responses, thus enriching client interactions with service metadata.
- **Alignment with SDCM 3.0**: The specification aligns with the latest Service Description Conceptual Model (SDCM) version 3.0, ensuring consistency and compatibility in service metadata representation and management.

- **Enhanced Security Considerations**: An increased focus on security led to the adoption of OAuth 2.0 as the recommended security solution for the SDS environment, addressing authentication and authorization in a more robust and flexible manner.
- **API Versioning Requirements**: Clear guidelines for API versioning have been established to facilitate future updates and backward compatibility, ensuring seamless evolution of the SDS APIs.

These enhancements collectively aim to address the evolving needs of the global aviation community, promoting seamless service metadata exchange, maximizing interoperability, and ensuring the discoverability and accessibility of services within SWIM.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

1.1 Purpose

The purpose of this specification is to establish guidelines and general technical principles for the development of SWIM Discovery Services (SDS). It presents the enabling technologies and practices that support federated service discovery among independently developed and autonomously managed SDS implementations.

1.2 Overview

Service discoverability is a critical component in the development of SWIM ecosystems. Service discoverability refers to the ability of a technology architecture to locate a service using a uniformly interpretable set of service metadata. This metadata is accessed by a service consumer through mechanisms like discovery services or service registries.

In recent years, the notion of "global" SWIM has introduced new challenges to SWIM developers in the area of service discoverability.

One such challenge is enabling SWIM stakeholders to retrieve service metadata not only from registries they are affiliated with and have direct access to but also from other SWIM registries [7].

A notable issue for implementing such a solution is that today's SWIM service registries are designed to interact explicitly with a human and do not support machine-to-machine communication; in other words, registries cannot interoperate ("talk") with each other.

The solution defined by this specification is implementing a service called SWIM Discovery Service (SDS), which aims to provide service metadata that may be retrieved from a service registry or other SDS.

1.3 Intended Audience

This document is designed for use by:

- Developers and architects involved in designing and implementing SWIM Discovery Services.
- SWIM implementers involved in the development of REST API services and applications.
- SWIM analysts and systems architects responsible for conceptualizing and overseeing SWIM architectures.
- Groups and individuals responsible for setting SWIM regional and global standards.

A basic understanding of HTTP and REST principles is assumed.

1.4 Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD

NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted with the significance attributed to them within RFC-2119 [10]. These key words are capitalized when used to unambiguously specify requirements. When these words are not capitalized, they are meant in their natural-language sense.

All parts of this document that are labeled as examples should be considered "non-normative" unless explicitly stated otherwise. Such non-normative examples are intended to enhance understanding and facilitate the practical application of the specification's principles but do not constitute mandatory requirements for compliance.

Throughout this document, `instances of text encased in shaded and bordered boxes` are utilized to denote literal values, terms, or code snippets that are directly applicable in implementations.

1.5 Terms and Definitions

For the purpose of this specification, the following terms and definitions apply:

application programming interface (API)*:* A set of rules that define how applications or devices can connect to and communicate with each other.

discovery service (DS): A service that is compliant with the SDS specification and with which users directly interact for service discovery.

information service (IS): A service that offers capabilities for generating, making available, storing, managing, and analyzing information.

> **Note:** The term IS encompasses a broad category of services whose descriptions are stored in SWIM registries. While SWIM registries may contain data on a diverse range of services (e.g., Core Services [3]) in this document, any service that is not a Discovery Service (DS) will be collectively referred to as an Information Service (IS).

principal discovery service: A DS that a user initially contacts to initiate service discovery in SDS environment.

> **Note:** The *principal discovery service* is understood as the first point of contact or the main gateway for a user within the SDS environment.

resource path: A relative URI (Uniform Resource Identifiers) to an individual endpoint.

REST API: An API that conforms to the constraints of the REST architectural style and allows for interaction with RESTful Web Services.

RESTful Web Service: A web service that follows the REST architectural style.

service registry: An application designed to support service discovery through a formal registration process for storing, cataloging, and managing collection of service metadata [4].

peer service (PS): Any discovery service other than the one initially accessed by a user.

resource: A representation of an entity (e.g., a service or service description); it is identified by a URI and can be manipulated via a uniform interface and exchangeable representations.

user: A person or system that interacts with a discovery service.

2 Architecture

2.1 Conceptual Vision

Following the SOA principles [9] as well as established SWIM practices, an SDS is designed to be interoperable, loosely-coupled, autonomous, and self-describing where:

- ***interoperable:***
  The service uses standardized communication protocols, data formats, and well-defined APIs (Application Programming Interfaces) that specify how it can interact with its peers.
- ***loosely-coupled:***
  The service is designed as a self-contained module with a well-defined function, reducing dependencies on other services; each service needs minimal knowledge about the workings of other services.
- ***autonomous:***
  The service manages its state and resources without external intervention; it encapsulates its business logic, ensuring that it can operate effectively on its own.
- ***self-describing:***
  The service provides metadata about itself, identifying the service (ID, name, provider, etc.) and references to accessible documentation, which are crucial for developers who need to integrate the service into their applications.

In a broader context, each SDS instance functions as a node within a decentralized network, where each node operates independently and interacts with other SDS instances (aka., peers) to exchange information about the contents of their respective registries.

Figure 1 illustrates a typical SDS scenario where two SWIM implementations exchange service metadata. It depicts how a User affiliated with SWIM A (User A) may access SWIM Registry A to input or query service metadata through a Graphical User Interface (GUI). To access information from another SWIM Registry (Registry B), the User utilizes an instance of the SWIM Discovery Service A, that is, a service defined by this specification. This setup enables SDS A to communicate with SDS B via a RESTful API, facilitating the exchange and retrieval of service metadata by User A from Registry B. Importantly, this scenario is bidirectional, allowing similar operations from SWIM B towards SWIM A or between any other SWIMs with an established SDS instance.

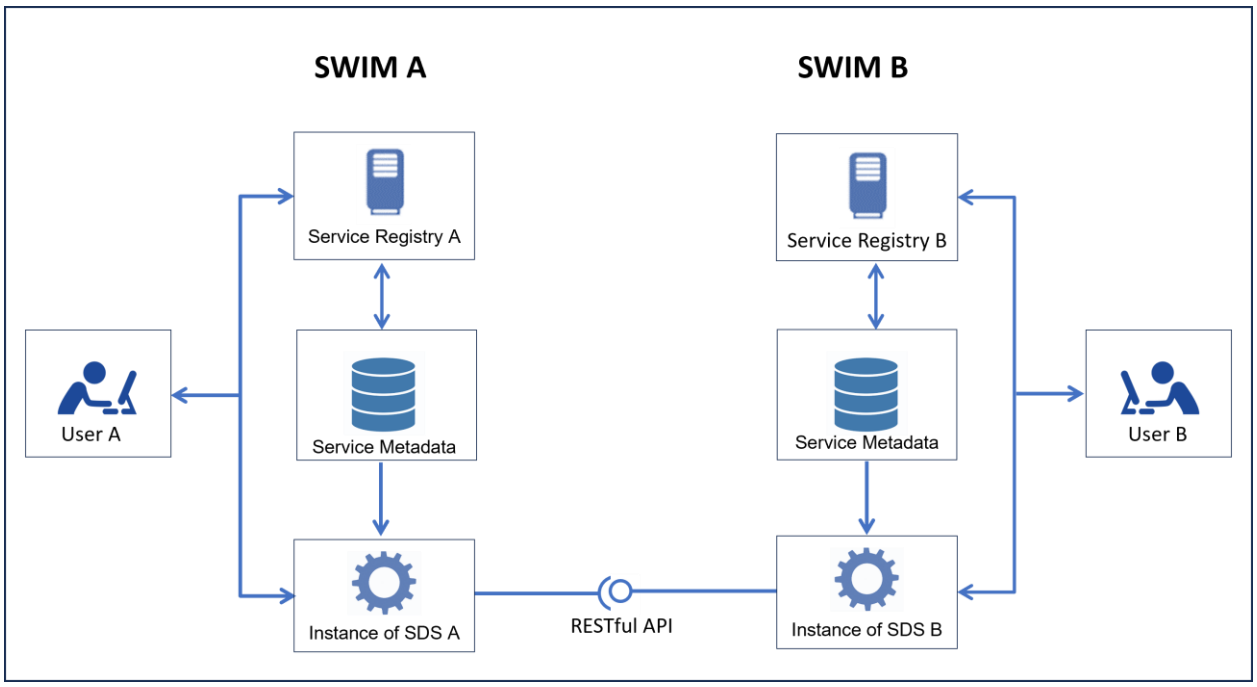

Figure 1. Notional example of SDS environment

At its core, the SWIM Discovery Service (SDS) is a specialized web service designed to facilitate the retrieval of service metadata from diverse SWIM registries. It is a vital tool in transcending geographical and organizational divides, automating the discovery process in the global SWIM ecosystem.

2.2 Technological Solutions

In order to meet these requirements, the SDS adopts two architectural styles: Representational State Transfer (REST) [2] and Peer-to-Peer (P2P) Discovery [9].

2.2.1 REST and RESTful API

A REST architecture, often implemented through RESTful Web Services (RESTful WS) [2], is the most common approach for building Application Programming Interfaces (APIs) in distributed systems. RESTful APIs use standard HTTP methods to interact with resources, which are identified via Uniform Resource Identifiers (URIs). RESTful APIs are characterized by their statelessness, meaning each request from a client to a server must contain all the information needed to understand and process the request. This approach ensures that RESTful APIs are scalable, reliable, and independent.

The following factors have led to the selection of this architectural solution:

- ***interoperability****:* RESTful APIs are built on standard HTTP methods, ensuring broad compatibility and facilitating seamless interactions between disparate systems in a federated network.
- ***uniform Interface****:* A consistent interface simplifies the overall architecture and decouples clients from servers, allowing for the independent evolution of their respective discovery systems.

- ***ease of use:*** The widespread familiarity with HTTP [1] facilitates easy adoption and use of RESTful APIs in the development of discovery services.

2.2.2 Peer-to-Peer (P2P) Discovery

A Peer-to-Peer (P2P) network, as an architectural solution, operates on the principle of decentralized distribution and control among the network's nodes, or peers. In a P2P network, each peer functions both as a client and a server, unlike traditional client-server models where roles are distinctly separated. This means that every node can initiate requests (as a client) and also respond to requests (as a server) from other nodes. Such a network architecture eliminates the need for centralized coordination, allowing for a more resilient and scalable system.

The following factors have led to the selection of this architectural solution:

- ***decentralization:*** The decentralized topology of P2P networks evenly distributes the network load, reducing single points of failure and performance bottlenecks typical of client-server models.
- ***absence of a central registry:*** A lack of a central registry across existing SWIM implementations necessitates a model where data concerning SDS nodes are disseminated through each member relaying information to their peers.
- ***dynamic scalability:*** The addition or removal of nodes in the SDS network can be achieved seamlessly without impacting the performance or availability of other nodes within the network.

The combined deployment of RESTful APIs and P2P-based communication within the SDS architecture predicates on delivering a resilient, scalable, and interoperable discovery mechanism suited to the expansive and evolving nature of SWIM global service discovery.

3 General Requirements

The following requirements capture the essential functionalities and attributes that the SDS is expected to exhibit.

1. SDS SHALL enable the discovery of SWIM-enabled services across organizational and geographical divides.
2. SDS SHALL provide complete service descriptions consistent with data exchange models adopted by SWIM.
3. SDS SHALL adhere to open standards and protocols to ensure compatibility with diverse SWIM participants and service providers.
4. SDS SHALL provide fast and responsive service discovery, minimizing latency and maximizing throughput.
5. SDS SHALL implement robust security measures to protect service metadata and prevent unauthorized access.

6. SDS SHALL be extensible to accommodate future requirements and evolving SWIM architecture.
7. SDS SHALL operate within a decentralized network of discovery services, enabling peer-to-peer (P2P) communication and collaboration.
8. SDS SHALL maintain and provide information about its peers within the network, including their location and endpoints.

4 Use Cases

This section outlines various scenarios and interactions that the SWIM Discovery Service (SDS) is designed to handle by defining a set of use cases.

Use cases provide a clear and detailed description of how users interact with the system. They help ensure that the architectural design aligns with the practical needs and expectations of users.

The SDS is designed to support the following use cases:

Use Case 1: Retrieving a Service Description of a principal DS

Precondition: The user is aware of the endpoint of a principal DS.

Scenario:

a) The user sends a request to the DS for its service description.

b) The DS responds with its service description, which outlines its usage. The response includes

- **a.** A Uniform Resource Identifier (URI) [11] that uniquely references and locates the primary DS.
- **b.** The full name of the service and any commonly used acronyms.
- **c.** The current version or release number of the primary DS.
- **d.** Service Provider Details:
    a. The name of the organization responsible for providing the DS.
    b. A brief outline of the organization, highlighting its role and significance in the context of the DS.
    c. The official website or web page of the service provider.
    d. Contact information for a designated person or group within the organization, facilitating human communication for inquiries, support, or other purposes.
- **e.** Direct references to pertinent documentation that consumers may require to interact effectively with the DS. This could include user guides, API documentation, integration manuals, and other relevant materials.

Effect: The user utilizes this information to build a client for further service discovery.

Use Case 2: Discovering Peer Services through Principal DS

Precondition: The user has access and is able to deploy the principal DS.

Scenario:

a) The user wishes to retrieve information about services, which may be cataloged in other SWIM registries and accessible via other SDS instances. To this end, the user requests a list of PSs known to the principal DS.

b) The principal DS responds with endpoints of zero or more PSs that it has records of.

c) The information about each PS includes:

i. A URI that uniquely references each PS [11].

ii. Descriptive names and acronyms are provided for each PS.

iii. The current version of the PS.

iv. This endpoint is where the PS can be accessed, allowing users to request a self-description of the peer service.

Effect: The User employs these endpoints to access the PSs, as outlined in Use Case 1. Beyond this point, all use cases apply equally to all SDS instances: DS and PSs.

Use Case 3: Retrieving a List of Information Services (IS) from a PSs

Precondition: The user is able to interact with a DS (either principal or peers).

Scenario:

a) The user seeks information about services potentially cataloged in other SWIM registries and accessible through their respective SDS instances.

b) The DS responds with a list of ISs, each including

i. A URI that uniquely references and locates each IS [11].

ii. The full name of the IS and any commonly used acronyms.

iii. The current version of the IS.

iv. A brief textual summary of the IS.

Effect: The user gains insights into services accessible through the DS's registry and may further explore any service of interest.

Use Case 4: Obtaining Full Detailed Service Description from the List of ISs

Precondition: The user has compiled a list of services as per Use Case 3.

Scenario:

a) The user selects a service from this list and requests its comprehensive description, as defined in the SDCM.

b) The DS responds with the full service description of the chosen service, i.e., with an instance of service description as it defined in the SDCM [8].

Effect: The user acquires comprehensive details about a specific service, facilitating informed decision-making for service usage.

Figure 2 combines all use cases to depict the entire discovery process starting from initiation and ended with retrieving a specific detailed service description.

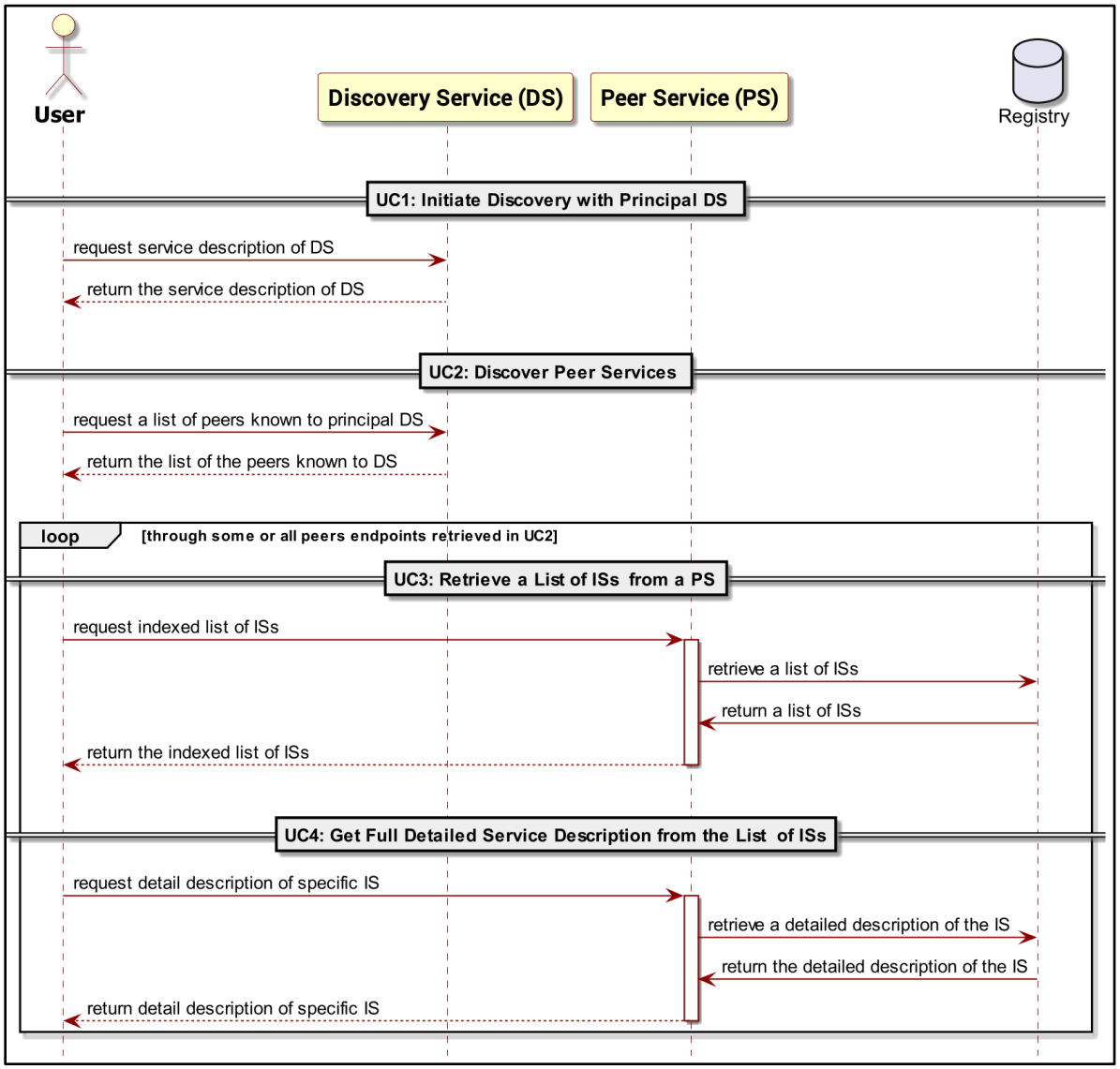

Figure 2. SDS Use Cases

5 Resource Model

The Resource Model outlined in this section provides a comprehensive overview of the different resources exposed by the SDS API, including their structures, relationships, and functionalities.

In addition to adhering to the commonly recognized practices of REST API, the SDS Resource Model encapsulates the following specific requirements.

1. Each resource within the SDS SHALL be uniquely identified. This identification is achieved by combining a resource path (e.g., `/discovery-service`) with an HTTP method (e.g., GET) used for accessing or manipulating the targeted resource.
2. The root resource of the SDS API SHALL be `/discovery-service`. The endpoint, formed by the combination of a host URL and the path `/discovery-service`, will serve as the primary entry point of the SDS API.
3. Resources within the SDS API may incorporate zero or more parameters. Each parameter SHALL be distinctly defined by a combination of a name and its location within the resource structure.
4. All resources in the SDS, upon retrieval, SHALL return a representation in the JavaScript Object Notation (JSON) data interchange format.

5.1 Resource: /discovery-service

path:	/discovery-service
HTTP method:	GET
description:	Returns information about a principal discovery service.

5.1.1 Parameters

None.

5.1.2 Representation

Compliance with the schema presented in Figure 3 is REQUIRED for the representation of the `/discovery-service` resource.

This schema enforces specific linking requirements per RFC 8288 Web Linking standard [12]. These requirements are outlined as follows:

1. Exactly one link with `"rel": "service"` SHALL be present. This link points to the endpoint of the described discovery service.

There SHALL be one or more links with `"rel": "describedby"`. These links refer to resources that provide descriptions in the context of the URI, such as API documentation, SDS specifications, or other pertinent materials, allowing for a comprehensive understanding of the service.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "description": "Information that describes an instance of SWIM Discovery Services (SDS).",
  "properties": {
    "id": {
      "type": "string","format": "uri"
    },
    "name": {
      "type": "string"
    },
    "version": {
      "type": "string"
    },
    "provider": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string"
        },
        "description": {
          "type": "string"
        },
        "website": {
          "type": "string"
        },
        "point-of-contact": {
          "type": "array",
          "items": [
            {
              "type": "object",
              "properties": {
                "name": {
                  "type": "string"
                },
                "function": {
                  "type": "string"
                },
                "email": {
                  "type": "string"
```

```
                }
              },
              "required": ["name", “function”,” email"]
            }
          ]
        }
      },
      "required": ["id", "name", "description", "website”, “point-of-contact"]
    },
"references": {
      "type": "object",
      "properties": {
        "links": {
          "type": "array",
          "minItems": 2,
          "items": {
            "type": "object",
            "properties": {
              "rel": {
                "type": "string",
                "enum": ["service", "describedby"]
              },
              "href": {
                "type": "string"
              },
              "title": {
                "type": "string"
              },
              "type": {
                "type": "string"
              },
              "language": {
                "type": "string"
              }
            },
            "required": ["rel", "title", "href", "type"]
          }
        }
      },
      "required": ["links"]
```

```
    }
  },
  "required": ["id", "name", "version", "provider", "references"]
}
```

Figure 3 Schema of representation of the */discovery-service* resource

An example of data following the `/discovery-service` resource schema can be found in Appendix A.

5.2 Resource: /discovery-service/peers

path: /discovery-service/peers
HTTP method: GET
description: Returns a collection of peers' services known to the discovery service.

5.2.1 Parameters

None.

5.2.2 Representation

Compliance with the schema presented in Figure 4 is REQUIRED for the representation of the `/discovery-service/peers` resource.

Each peer service entry SHALL include a link with `"rel": "service"`, signifying the endpoint through which the PS can be accessed.

```
{
 "$schema": "http://json-schema.org/draft-07/schema#",
 "type": "object",
 "properties": {
  "peers": {
   "type": "array",
   "items": {
    "type": "object",
    "properties": {
     "service-id": {
      "type": "string",
      "format": "uri"
     },
     "name": {
      "type": "string"
     },
```

```
     "version": {
      "type": "string"
     },
     "description": {
      "type": "string"
     },
     "links": {
      "type": "array",
      "minItems": 1,
      "items": {
       "type": "object",
       "properties": {
        "rel": {
         "type": "string",
         "enum": ["service" ],
         "description": "A link relation type that represents the endpoint of the peer
service."
        },
        "title": {
         "type": "string"
        },
        "href": {
         "type": "string",
         "format": "uri",
         "description": "The URL to the endpoint of the peer service."
        }
       },
       "required": ["rel","title","href"]
      }
     }
    },
    "required": ["service-id","name","version","description","links"]
   }
  }
 },
 "required": ["peers"]
}
```

Figure 4 Schema of the representation of the */discovery-service/peers* resource

An example of data following the `/discovery-service/peers` resource schema can be found in Appendix B.

5.3 Resource: /discovery-service/services

path: /discovery-service/services

HTTP method: GET

description: Returns a collection of ISs known to the targeted DS.

5.3.1 Parameters

The `/discovery-service/services` resource supports the retrieval of specific items from the service collection through a filtering mechanism based on name-value query parameters. Filters are applied by appending parameter assertions to the URL, with multiple assertions combined using the "&" symbol to form an "AND" relationship:

`/{resource}?{parameterName1}={parameterValue1}&{parameterName2}={parameterValue2}`

For instance, to fetch services categorized as "Flight" that are currently operational, the following URI format should be utilized:

```
/discovery-service/services?service-category=flight&availability-status=operational
```

To encode "OR" logic within a single parameter, separate the values with a semicolon (`;`) using the following expression:

`/{resource}?{parameterName}={parameterValue1};{parameterValue2}`

An example to retrieve services that are either "operational" or "prospective" would look like:

```
/discovery-service/services?availability-status=operational;prospective
```

Note, this behavior SHALL be explicitly supported by the API. Query parameters SHALL be properly URL-encoded to handle special characters and spaces. For example, spaces should be encoded as `%20` or replaced with `+`.

name: service-category
permissible value: values defined in http://semantics.aero/service-category taxonomy
type: query
required: false

name: availability-status
permissible value: values defined in http://semantics.aero/availability-status taxonomy
type: query
required: false

name:	interface-type
permissible value:	values defined in http://semantics.aero/interface-type taxonomy
type:	query
required:	false

5.3.2 Representation

Compliance with the schema presented in Figure 5 is REQUIRED for the representation of the `/discovery-service/services` resource.

This schema ensures that each service entry complies with a standardized format, including essential details such as service identification, version information, description, and categorized links that adhere to the Hypermedia as the Engine of Application State (HATEOAS) principle. By utilizing links compliant with RFC 8288 Web Linking standard [12], the document facilitates the dynamic discovery and integration of services based on taxonomies hosted by "discovery.aero". This approach not only enhances the interoperability and scalability of the SDS but also allows for the seamless integration of updates and modifications to service definitions and classifications from external sources.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "services": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "service-id": {
            "type": "string",
            "format": "uri"
          },
          "name": {
            "type": "string"
          },
          "version": {
            "type": "string"
          },
          "description": {
            "type": "string"
          },
          "categories": {
            "type": "array",
            "items": {
```

```
            "type": "object",
            "properties": {
              "category": {
                "type": "string"
              },
              "links": {
                "type": "array",
                "items": {
                  "type": "object",
                  "properties": {
                    "rel": {
                      "type": "string",
                      "enum": ["describedby", "code"]
                    },
                    "href": {
                      "type": "string",
                      "format": "uri"
                    },
                    "title": {
                      "type": "string"
                    },
                    "type": {
                      "type": "string",
                      "enum": ["text/html"]
                    },
                    "language": {
                      "type": "string",
                      "enum": ["en"]
                    }
                  },
                  "required": ["rel", "href", "title", "type", "language"]
                },
                "minItems": 1
              }
            },
            "required": ["category", "links"]
          }
        }
      },
      "required": ["service-id", "name", "version", "description", "categories"]
```

```
      }
    }
  },
  "required": ["services"]
}
```

Figure 5 Schema of the representation of the *`/discovery-service/services`* resource

An example of data following the `/discovery-service/services` resource schema can be found in Appendix C.

5.4 Resource: /discovery-service/services/service-description

path: /discovery-service/services/service-description

HTTP method: GET

description: Returns a comprehensive description of the selected service or services.

5.4.1 Parameters

The `/discovery-service/services/service-description` resource allows clients to retrieve detailed descriptions for one or more Information Services (IS) based on their identifiers (`service-id`). This is achieved by appending the `service-id` parameter to the query string of the request URL. The parameter supports both single and multiple values, enabling the retrieval of descriptions for specific services as needed.

For example, to fetch the description of a single service, its service-id should be included in the query parameter as follows:

```
/discovery-service/services/service-description?service-id=https://nsrr.faa.gov/services/fps
```

To retrieve descriptions for multiple services, list their service-id values separated by commas within the same query parameter:

```
/discovery-service/services/service-description?service-
id=https://nsrr.faa.gov/services/fps,http://swim.faa.gov/services/stdds-adp
```

For more detailed explanations on forming query strings and applying filters, refer to Section 3.3.1. This section includes comprehensive guidance on constructing query parameters for filtering, including the logical grouping of parameters and the use of URL encoding to handle special characters within parameter values.

name: service-id

description: The query parameter service-id is designed to accept one or more identifiers of the services for which the detailed descriptions are requested.
Note: This parameter must contain at least one valid service identifier. If a request is made without this parameter or with an empty value, the service should respond with the status code "400 Bad Request," indicating that the request cannot be processed due to the absence of required parameter values.
type: query
required: true

5.4.2 Representation

Compliance with the schema presented in Figure 5 is REQUIRED for the representation of the `/discovery-service/services/service-description` resource.

This schema facilitates the representation of services by including key identifying information such as service-id, name, version, and description, alongside detailed categorizations and references to additional specifications for `profile`, `model`, and `grounding`. These references are made to external JSON schema documents, enabling a modular and extensible approach to describing services in a comprehensive manner.

```
{
 "$schema": "http://json-schema.org/draft-07/schema#",
 "type": "object",
 "properties": {
  "services-description": {
   "type": "array",
   "items": {
    "$ref": "#/definitions/service-description"
   }
  }
 },
 "definitions": {
  "service-description": {
   "type": "object",
   "properties": {
    "service-id": {
     "type": "string",
     "format": "uri"
    },
    "name": {
     "type": "string"
    },
    "version": {
```

```
     "type": "string"
    },
    "description": {
     "type": "string"
    },
    "categories": {
     "type": "array",
     "items": {
      "type": "object",
      "properties": {
       "category": {
        "type": "string"
       },
       "links": {
        "type": "array",
        "items": {
         "type": "object",
         "properties": {
          "rel": {
           "type": "string",
           "enum": ["describedby","code"]
          },
          "href": {
           "type": "string",
           "format": "uri"
          },
          "title": {
           "type": "string"
          },
          "type": {
           "type": "string",
           "enum": ["text/html", “text/pdf”]
          },
          "language": {
           "type": "string",
           "enum": ["en"]
          }
         },
         "required": ["rel", "href", "title", "type", "language"]
        },
```

```
        "minItems": 1
       },
       "required": ["category", "links"]
      }
     }
    },
    "profile": {
     "$ref": "https://swim.aero/sdm-j/1.0.1/profile.json#/definitions/profile"
    },
    "model": {
     "$ref": "https://swim.aero/sdm-j/1.0.1/model.json#/definitions/model"
    },
    "grounding": {
     "$ref": "https://swim.aero/sdm-j/1.0.1/grounding.json#/definitions/grounding"
    }
   },
   "required": ["service-id","name","version","description","categories"]
  }
 },
 "required": ["services-description"]
}
```

Figure 6 Schema of the representation of the *`/discovery-service/services/ service-description`* resource

An example of data following the `/discovery-service/services/service-description` resource schema can be found in Appendix D.

6 Interface

In the context of RESTful APIs -- including the Service Discovery Service (SDS) -- the Interface is conceptualized as the definitive contract that outlines the interaction between the service provider and the client. This contract specifies the operations available to the client, each defined by its unique purpose and associated HTTP method. Operations facilitate the exchange of request and response messages between the client and service, encapsulating the communication's data.

6.1 Operations

An operation represents a discrete action that a client can perform on a resource. It is characterized by an HTTP method, which indicates the action's nature, such as GET, for retrieving resources. Each Operation initiates a sequence of request and response messages, enabling structured interactions with the API.

The SDS Interface is designed to support the discovery and description of services within a system. It consists of four primary operations, with each of these operations corresponding to a specific Use Case

previously described (see section 2.3) and retrieves a particular resource outlined in the Resource Model (see section 3).

Table 1 demonstrates the relationships between the Use Cases, Operations, and the Resources that the operations retrieve. This structure underscores the logical organization of the SDS interface, emphasizing how each operation serves a distinct business purpose in the service discovery process by interacting with specific resources.

Operation	***Use Case***	***Resource***
GetDiscoveryService	Use Case 1: Retrieving a Service Description of a principal DS	/discovery-service
GetPeers	Use Case 2: Discovering Peer Services (PS) through Principal DS	/discovery-service/peers
GetServices	Use Case 3: Retrieving a List of Information Services (IS) from a PSs	/discovery-service/services
GetServiceDescription	Use Case 4: Obtaining Full Detailed Service Description from the List of ISs	/services/services/service-description

Table 1 Operations supported by SDS.

6.2 Messages

Messages are the vehicles of data exchange in a RESTful architecture [2]. All messages, along with their headers and bodies, SHALL comply with the standards set forth in RFC 2616, Hypertext Transfer Protocol - HTTP/1.1 [1].

A Message is categorized into:

- Request Message: Initiated by the client, a request message calls upon the service to perform a specified operation.
- Response Message: Issued by the service, a response message contains the outcome of the requested operation (e.g., a representation of a resource).

Both request and response messages comprise a header and an optional body:

- Message Header: Contains a structured collection of key-value pairs that convey essential information about the message, such as content type, content length, and other metadata. Headers play a crucial role in HTTP communications, ensuring that messages are correctly understood and processed by both the client and the service.
- Message Body: Present when relevant, the message body carries the data pertinent to the operation. For instance, in a GET request, the body would include the retrieved resource information.

6.2.1 Header

The following requirements capture the essential HTTP header fields [1] and attributes that the SDS is expected to exhibit.

6.2.1.1 Request Header

a. The client SHALL use the `Accept` header field to specify the preferred media type(s) for the response.
b. The client SHALL use the `Accept-Language` header field to specify the preferred natural language(s) for the response.
c. The default expected language is English (`en`), which SHALL be assumed if no specific language is requested.

6.2.1.2 Response Header

a. The server SHALL use the `Content-Type` header field to specify the media type of the response body.
b. The server SHALL support `application/json` as the default media type.
c. When a client specifies preferred languages using the `Accept-Language` header, the server SHALL return the response in the specified locale whenever possible. If the specified language is not supported, the response SHALL be provided in English (`en`) by default.
d. The server MAY use the `Content-Location` header field to provide the direct URL of the resource represented in the response. This is especially valuable when the response body is derived from a location different from the requested URL, such as when data is sourced from an external registry.
e. When appropriate, response bodies MAY be compressed to reduce the size of the transmitted data. Supported compression schemes (e.g., `gzip`) SHOULD be indicated in the `Accept-Encoding` request header, and the server's choice of compression, if any, SHALL be declared in the `Content-Encoding` response header. This approach optimizes network usage and improves the efficiency of data transfers.
f. For responses representing a collection of items, such as those returned by the `GetServices` operation, the server MAY include an `X-Total-Count` header field to convey the total number of items in the collection. This information aids clients in understanding the scope of the returned data and facilitating pagination.

6.2.2 Body

a. For all Service Discovery Service (SDS) operations, the request body SHALL be empty. This stipulation ensures that operations are driven solely by parameters encoded in the request's URI.
b. For each SDS operation, the response body SHALL include a representation of the resource(s) associated with the operation, as outlined in Table 1 of the SDS specification.

6.2.3 Status Codes

HTTP status codes act as standardized indicators that summarize the outcome of the server's attempt to fulfill a client's request. Embedded within every HTTP response message, these status codes provide immediate feedback to the client about the success, failure, or required next steps in the interaction with the web service.

Each status code is a three-digit integer where the first digit categorizes the response, and the next two digits specify the outcome within that category. Accompanying each status code is a reason phrase, a textual representation that provides a brief explanation of the status code, making it more understandable for humans. These codes and their interpretations are standardized in RFC 2616, ensuring consistency across the web.

Specific status codes and their meanings relevant to SDS operations are detailed in Table 2 of this specification. The selection of these codes is tailored to the types of interactions expected in service discovery scenarios, ensuring that clients can effectively interpret responses and act accordingly.

Status Code	***Reason Phrase***	***Requirement***
200	OK	SHOULD be used to indicate nonspecific success. The response body contains the data requested by the client.
204	No Content	SHOULD be used when the response body is intentionally empty, e.g., a request to retrieve services based on specific criteria returned zero responses.
301	Moved Permanently	SHALL be used to indicate that the requested resource has been permanently moved to a new URL. The response SHOULD include a Location header field indicating the URI to which the resource has been moved.
307	Temporary Redirect	SHALL be used to indicate that the requested resource resides temporarily under a different URI. The client SHOULD repeat the request with the URI provided in the Location header field.
400	Bad Request	SHALL be used when the request cannot be processed due to malformed syntax, invalid query parameters, or a bad request structure.

401	Unauthorized	SHALL be used when the request lacks valid authentication credentials for the target resource.
403	Forbidden	SHALL be used when the server understood the request but refuses to authorize it.
404	Not Found	SHALL be used when a specified resource or endpoint could not be found.
405	Method Not Allowed	SHALL be used when the HTTP method used is not supported by the resource.
406	Not Acceptable	SHALL be used when the requested resource is capable of generating only content not acceptable according to the Accept headers sent in the request.
408	Request Timeout	SHOULD be used when the server times out waiting for the request.
500	Internal Server Error	SHALL be used when an unexpected condition was encountered and no more specific message is suitable.
501	Not Implemented	SHALL be used when the server does not support the functionality required to fulfill the request.
503	Service Unavailable	SHOULD be used when the server is currently unable to handle the request due to temporary overloading or maintenance. This is a temporary state.

Table 2 HTTP Status Coded supported by SDS

6.2.4 Caching

This specification aims to promote effective caching practices within the SDS environment. Properly applied caching is essential for optimizing efficiency and performance, reducing latency, and minimizing network traffic. Caching the response data is especially useful when the data is updated only periodically or rarely, as it is common to registry information.

a. The operation GetDiscoveryService SHALL be cached at all times. This requirement mandates that responses from the GetDiscoveryService operation be stored in cache to optimize performance and reduce server load.

b. The operation GetPeers SHALL always be cached. This ensures that information retrieved by the GetPeers operation is readily available in cache, facilitating efficient access to data about peer services.

c. The operation GetServices SHOULD be cached. While not as strictly required as the GetDiscoveryService and GetPeers operations, caching the responses from GetServices is recommended to enhance service discovery efficiency.

d. The operation GetServiceDescription SHOULD be cached. Similar to GetServices, caching the results of GetServiceDescription is advised to improve responsiveness and decrease the need for repeated requests to the server for the same information.

The `Cache-Control` HTTP general-header field is used to specify directives for caching mechanisms in both requests and responses. Its primary purpose is to help control how and for how long network responses can be cached.

In some cases, a requester may discourage caching. In these cases, `Cache-Control` can be added with the values `no-cache`, `no-store`, and `must-revalidate`. However, it should be noted that directives that prevent a service from using a cached response SHOULD NOT be used unless absolutely necessary or infeasible.

6.2.5 Authentication

This specification supports Basic Authentication as a method for user-server authentication. It also allows for a Token-Based Authentication to enhance security and flexibility in authentication processes. Both methods utilize the Authorization request header field in a request message, enabling the server to verify the user's identity efficiently. Refer to section 7, "Security Considerations," for additional authentication and authorization mechanisms.

6.2.5.1 Basic Authentication

a. The client SHOULD include this Authorization header in each request requiring authentication, allowing the accessed service to verify the user's identity.

b. The value of the Authorization field SHALL consist of the word Basic, followed by a space, and then a base64-encoded string credentials. These credentials SHALL be the user's identifier (such as a username) and password concatenated with a colon (:) character: `Authorization:Basic`<username:password>.

c. When constructing a request, the client SHALL encode the user's identifier and password into a base64-encoded string and prepend it with Basic to construct the complete `Authorization` header value. For example: `Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==`

d. If the credentials are invalid, the server MUST respond with a `401 Unauthorized` status code.

6.2.5.2 Token-Based Authentication

e. For Token-Based Authentication, the `Authorization` field's value SHALL consist of the word "Bearer," followed by a space, and then the token string: `Authorization: Bearer` <token>.

f. Upon successful initial authentication using Basic Authentication or another method, the server MAY issue a token to the client. The client SHALL include this token in the Authorization header of subsequent requests to authenticate themselves without sending their credentials again.

g. The issued token SHOULD be a secure, signed token that uniquely identifies the user and MAY include additional claims or permissions relevant to the user's session.

h. The server SHALL validate the token included in the Authorization header of each request and respond appropriately based on the validity of the token.

i. If the token is invalid or expired, the server MUST respond with a 401 Unauthorized status code.

7 Security Considerations

Basic and Token-Based authentication, while widely used for their simplicity and effectiveness in many scenarios, may not be the most suitable security solutions for the SDS environment, particularly when declared via the Authorization header field. There are several reasons for this:

Security Vulnerabilities: Basic Authentication involves sending base64-encoded credentials with each request, which, if intercepted (especially over non-HTTPS connections), can compromise security. Similarly, tokens, if intercepted, can provide unauthorized access to services. While tokens are generally more secure than Basic Authentication credentials (especially when using HTTPS), the risk of token theft and replay attacks still exists.

Statelessness and Scalability Concerns: SDS environments, particularly those designed to support a federated model with numerous interconnected instances and services, require a stateless authentication mechanism that scales efficiently. Basic Authentication, by its nature, requires the server to verify credentials with every request, which can be resource-intensive and less scalable. Token-Based Authentication is more scalable and stateless, but managing and validating tokens across multiple services and instances can introduce complexity and overhead, potentially impacting performance.

Delegated Access Limitations: SDS environments often require the ability to delegate access to resources in a controlled manner, such as allowing one SDS instance to access resources in another on behalf of a user. Basic and simple Token-Based Authentication methods do not natively support delegated access scenarios.

Given the outlined deficiencies of Basic and Token-Based Authentication for the SDS environment, it becomes clear that a more effective approach is required to meet its unique demands. The OAuth 2.0 protocol [13] emerges as a fitting solution, offering a comprehensive framework designed to address the security challenges inherent in modern distributed systems like SDS.

The SDS environment, characterized by its federated architecture and the need for secure, cross-domain service metadata exchange, presents unique security and operational challenges. OAuth 2.0's comprehensive approach to authorization and security directly addresses these challenges by providing:

Interoperability: As a widely adopted standard, OAuth 2.0 facilitates interoperability among different systems and services. By aligning with OAuth 2.0, the SDS can more easily integrate with other services and platforms, promoting a seamless user experience across the federated landscape.

Enhanced Security: OAuth 2.0 introduces a higher level of security by abstracting the user's credentials from the actual request. Tokens issued by an Authorization Server, which can be tightly controlled and monitored, replace the need to send credentials with each request, significantly reducing the risk of credential theft.

Delegated Authorization: Delegated authorization is one of the most important characteristics of OAuth 2.0. It allows users to grant access to their protected resources to a third-party application without sharing their credentials.

Building on the advantages of integrating OAuth 2.0 into the Service Discovery Service (SDS) environment, it's essential to understand how OAuth 2.0's components align with and support the architectural and operational framework of SDS. The OAuth 2.0 framework comprises several key components, each playing a crucial role in the authentication and authorization process. Here's how these components fit into the SDS ecosystem, supporting its architecture and use cases:

resource owner - An entity capable of granting access to a protected resource. In the context of SDS 2.0, a principal DS has the authority to grant access to service metadata or to delegate the authority to access metadata from a peer DS.

client - An application making protected resource requests on behalf of the resource owner and with its authorization. In the SDS 2.0, a user that makes requests to access service metadata or perform discovery operations within the SDS.

resource server - The server hosting the protected resources is capable of accepting and responding to protected resource requests using access tokens. In SDS 2.0, the DS Instance (typically, a peer DS) that hosts the service metadata acts as the *resource server*. It is capable of handling requests for protected resources (service metadata) using access tokens.

authorization server - A server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization. In the time of writing, the authorization server has not been specified in SDS.

Use cases for the implementation of OAuth2 in the context of SDS may look like the following.

1. A User (Resource Owner) needing to access data from a Peer DS (Resource Server) initiates the process through a principal DS (Client).
2. The Client requests access to the Peer's SDS (Resource Server), presenting credentials or tokens obtained from an Authorization Server. This step verifies that the Client is authorized to access the data on behalf of the Principal.
3. Upon successful authentication and authorization, the Peer's DS (Resource Server) provides the requested data to the Client, which then relays this information back to the User (Resource Owner).

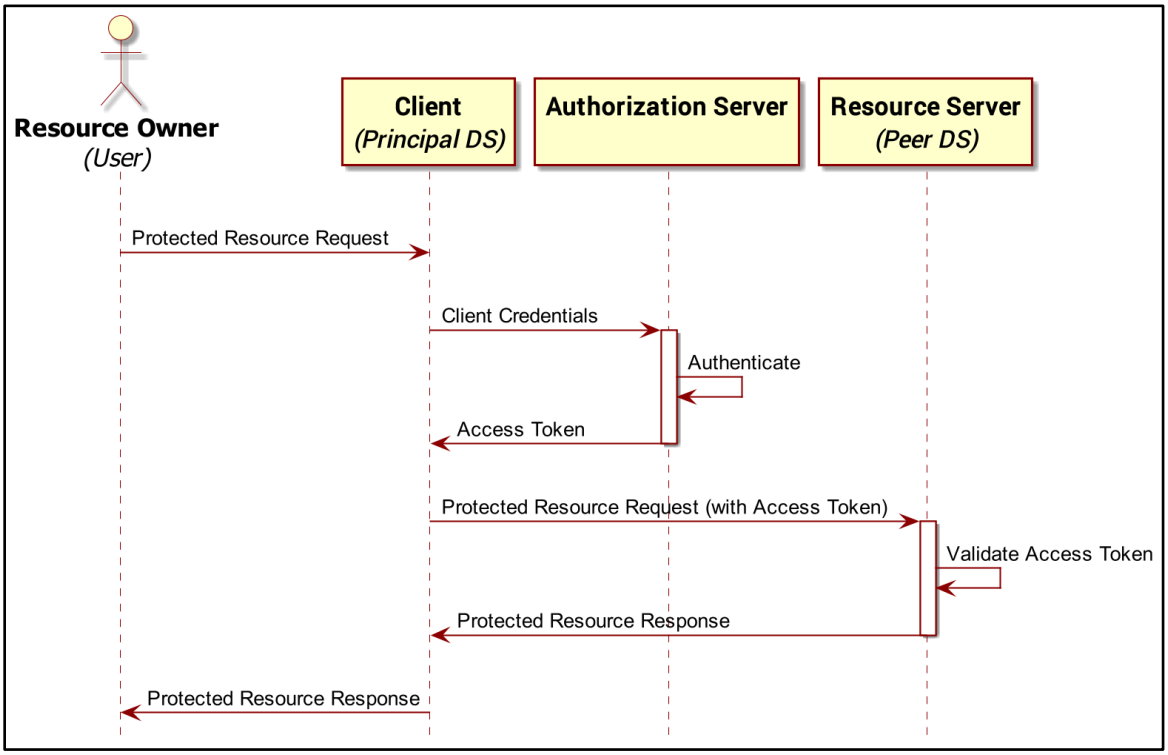

Figure 7 OAuth 2 in the SDS context

Note: This specification asserts that the identification of specific security protocols and regulations falls outside its purview. Instead, it offers recommendations with the understanding that the responsibility for specifying and implementing SDS security measures should reside with organizations charged with establishing SWIM security standards, both regionally and globally. This delineation of responsibilities aims to:

- Promote uniform security practices across various SDS implementations and geographical regions, fostering a secure and interoperable SWIM ecosystem.
- Facilitate compliance with the aviation sector's regulatory and policy frameworks, ensuring that SDS security measures meet the stringent requirements of international information exchange.
- Draw on the specialized knowledge and expertise of security professionals in the aviation context, enabling the adoption of robust and effective security solutions tailored to the unique challenges of SWIM.
- Encourage simultaneous progress in service discovery technologies and the security frameworks governing RESTful API services, ensuring both areas evolve cohesively to support the broader objectives of SWIM.

It is further anticipated that SDS developers will thoroughly review and incorporate any security measures recommended by these specialized entities into subsequent updates of this specification.

8 Versioning

a. The versioning of the SDS API SHALL comply with the established guidelines as detailed in the "Guidance for Versioning SWIM Services" [5] document.

b. The version of the SDS API SHALL be explicitly indicated in the service URL to ensure clarity. This practice allows clients to specifically target different versions of the API without ambiguity.

c. The version of the SDS API SHALL be explicitly indicated in the HTTP headers. This approach keeps the API endpoint clean and the URLs consistent across versions, allowing for easier future changes and version management. For example:

```
GET /discovery-service HTTP/1.1
Host: ansp-example.org
Accept: application/json
API-Version: 2.0
```

d. Services following the initial version of the specification and already in operation MAY continue to use unversioned URLs, such as `http://example.com/discovery-service`.

e. For an SDS instance based on version 2.0 of this specification the service URL SHALL explicitly include the version number, e.g., `http://example.com/v2/discovery-service`.

9 References

[1]	RFC 2616, Hypertext Transfer Protocol - HTTP/1.1; IETF; R. Fielding et al; June 2016 https://datatracker.ietf.org/doc/html/rfc2616
[2]	Architectural Styles and the Design of Network-based Software Architectures, Dissertation; Roy Thomas Fielding, 2000 https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
[3]	WP/10 SWIM Service Category Taxonomy; The Fourth Meeting of System Wide Information Management Task Force (SWIM TF/4); November 2020 https://www.icao.int/APAC/Meetings/2020%20SWIM%20TF4/WP10_USA%20AI.5d_Task1-5%20-%20SWIM%20Service%20Category%20Taxonomy.pdf
[4]	SWIM Controlled Vocabulary V. 1.0.0; FAA SWIM; 2019-03-25 https://semantics.aero/pages/swim-vocabulary.html
[5]	WP/04 Guidance for Versioning SWIM Services; The Third Meeting of System Wide Information Management Task Force (SWIM TF/3); Bangkok, Thailand, 07 – 10 May 2019 https://www.icao.int/APAC/Meetings/2019SWIMTF3/WP04_USA%20AI3d%20-%20Task%201-4_SWIM%20Service%20Versioning.pdf

[6]	Web Services Glossary; W3C Working Group Note; 11 February 2004 https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/
[7]	Concept of Operations for the SWIM Inter-Registry Framework (SIRF), Version 1.0.0; U.S. FAA SWIM, October 11, 2018 https://www.faa.gov/air_traffic/technology/swim/governance/international_collaboration/media/conops-sirf-1.0.0.pdf
[8]	Service Description Conceptual Model (SDCM) 3.0, FAA, September 2023 https://discovery.swim.aero/sdcm/3.0.0/SDCM_v3.0.0_092223.pdf
[9]	Web Services Architecture; W3C Working Group Note 11 February 2004 https://www.w3.org/TR/ws-arch/
[10]	RFC 2119, Key words for Use in RFCs to Indicate Requirement Levels, Network Working Group, March 1997 http://www.rfc-editor.org/rfc/rfc2119.txt
[11]	Guidance for Creating SWIM Service Identifiers; ICAO APAC SWIM-TF3; April 28, 2019 https://www.icao.int/APAC/Meetings/2019SWIMTF3/WP03_USA%20AI3d%20-%20Task%201-4_SWIM%20Service%20Identifier.pdf
[12]	Request for Comments: 8288, Web Linking; Internet Engineering Task Force (IETF), M. Nottingham; October 2017 https://https.cio.gov/
[13]	The OAuth 2.0 Authorization Framework, RFC6749; IETF; October 2012 https://tools.ietf.org/html/rfc6749

Appendixes

Appendix A. Resource /discovery-service – example of data


```
{
 "id": "http://swim.faa.gov/smxs",
 "name": "SWIM Metadata Exchange Service (SMXS)",
 "version": "2.0.0",
 "provider": {
  "name": "FAA SWIM Program",
  "description": "The SWIM is a program established by FAA to develop, provision, and operate distributed, collaborative, and reusable services by leveraging the principles of Service-Oriented Architecture (SOA).",
  "website": "https://www.faa.gov/air_traffic/technology/swim",
  "point-of-contact": [
   {
```

```
    "name": "John Doe",
    "function": "Program Manager",
    "email": "john.doe@faa.gov"
   }
  ]
 },
 "references": {
  "links": [
   {
    "rel": "service",
    "href": "http://nsrr.faa.gov/services/api/fps/discovery-service",
    "title": "SWIM Metadata Exchange Service (SMXS)"
   },
   {
    "rel": " describedby",
    "href": "http://nsrr.faa.gov/documents/smsx-wsdd.pdf",
    "title": "SMXS Web Service Description Document (WSDD)",
    "type": "application/pdf",
    "language": "en"
   },
   {
    "rel": "describedby",
    "title": "SWIM Discovery Service (SDS) Implementation Specification, Version 2.0.0",
    "href": "https://discovery.swim.aero/sds/1.0.0/SDS%20Specification%20v.1.0.0.pdf",
    "type": "application/pdf"
   }
  ]
 }
```


Appendix B. Resource /discovery-service/peers – example of data


```
{
 "peers": [
  {
   "service-id": "http://swim.airport.co.kr/sdmx",
   "name": "Korea Airports Corporation (KAC) Discovery Service",
   "version": "1.0.0",
   "description": "Managed by KAC, this service enables discovery of SWIM services across
Korea's aviation sector.",
   "links": [
```

```
    {
     "rel": "service",
     "title": "The Endpoint of KAC SDMX",
     "href": "https://api.otherservice.com/api/sdmx"
    }
   ]
  },
  {
   "service-id": "https://www.enri.go.jp/swim/sdmx",
   "name": "Electronic Navigation Research Institute Service Metadata Exchange Service (ENRI
SMXS)",
   "version": "1.0.0",
   "description": "Operated by ENRI, this service facilitates the discovery of JCAB-managed
SWIM services in Japan.",
   "links": [
    {
     "rel": "service",
     "title": "The Endpoint of ENRI SDMX",
     "href": "https://www.enri.go.jp/swim/api/v.2/sdmx"
    }
   ]
  }
 ]
}
```


Appendix C. Resource /discovery-service/services – example of data

The following is an example of the representation returned by the instance of SDS of the resource /discovery-service/services. In this example, a user limited the search to services that are categorized as flight services, in either prospective or operational lifecycle stages, and offered a method-oriented type of interface. (For the sake of brevity, only one service is shown. However, the array of services may have any number of items.)


```
{
 "services": [
  {
   "service-id": "https://nsrr.faa.gov/services/fps",
   "name": "Flight Plan Service (FPS)",
   "version": "1.0.0",
   "description": "A service for filing, updating, or canceling a flight plan.",
   "categories": [
    {
```

```
    "category": "SWIM Service Category",
    "links": [
     {
      "rel": "describedby",
      "href": "http://semantics.aero/service-category",
      "title": "SWIM Service Category",
      "type": "text/html",
      "language": "en"
     },
     {
      "rel": "code",
      "href": "http://semantics.aero/service-category#flight",
      "title": "Flight",
      "type": "text/html"
     }
    ]
   },
   {
    "category": "Service Availability Status",
    "links": [
     {
      "rel": "describedby",
      "href": "http://semantics.aero/availability-status",
      "title": "Service Availability Status",
      "type": "text/html",
      "language": "en"
     },
     {
      "rel": "code",
      "href": "http://semantics.aero/availability-status#prospective",
      "title": "Prospective",
      "type": "text/html"
     },
     {
      "rel": "code",
      "href": "http://semantics.aero/availability-status#operational",
      "title": "operational",
      "type": "text/html"
     }
    ]
```

```
    },
    {
     "category": "Service Interface Type",
     "links": [
      {
       "rel": "describedby",
       "href": "http://semantics.aero/interface-type",
       "title": "Service Interface Type",
       "type": "text/html",
       "language": "en"
      },
      {
       "rel": "code",
       "href": "http://semantics.aero/interface-type#method-oriented",
       "title": "Method-Oriented",
       "type": "text/html"
      }
     ]
    }
   ]
  }
 ]
}
```

Appendix D. Resource /discovery-service/services/service-description – example of data

```
{
 "services-description": [
  {
   "service-id": "https://nsrr.faa.gov/services/fps",
   "name": "Flight Plan Service (FPS)",
   "version": "1.0.0",
   "description": "A service for filing, updating, or canceling a flight plan.",
   "categories": [
    {
     "category": "SWIM Service Category",
     "links": [
      {
       "rel": "describedby",
       "href": "http://semantics.aero/service-category",
```

```
      "title": "SWIM Service Category",
      "type": "text/html",
      "language": "en"
     },
     {
      "rel": "code",
      "href": "http://semantics.aero/service-category#flight",
      "title": "Flight",
      "type": "text/html"
     }
    ]
   },
   {
    "category": "Service Availability Status",
    "links": [
     {
      "rel": "describedby",
      "href": "http://semantics.aero/availability-status",
      "title": "Service Availability Status",
      "type": "text/html",
      "language": "en"
     },
     {
      "rel": "code",
      "href": "http://semantics.aero/availability-status#prospective",
      "title": "Prospective",
      "type": "text/html"
     },
     {
      "rel": "code",
      "href": "http://semantics.aero/availability-status#operational",
      "title": "Operational",
      "type": "text/html"
     }
    ]
   },
   {
    "category": "Service Interface Type",
    "links": [
     {
```

```
       "rel": "describedby",
       "href": "http://semantics.aero/interface-type",
       "title": "Service Interface Type",
       "type": "text/html",
       "language": "en"
      },
      {
       "rel": "code",
       "href": "http://semantics.aero/interface-type#method-oriented",
       "title": "Method-Oriented",
       "type": "text/html"
      }
     ]
    }
   ],
   "profile": {// Data for the profile object is here.},
   "model": {// Data for the profile object is here.},
   "grounding": {// Data for the profile object is here.}
  }
       // Additional service-descriptions could be added here.
 ]
}
```