U.S. Department of Transportation
Federal Aviation Administration

Software Specification

ARTIFACTS VERSIONING IN SWIM
Version 1.0.0

September 12, 2024

Table of Contents

Table of Figures

1 Scope

1.1 Scope

This specification establishes the requirements and guidelines for versioning practices within the FAA's System Wide Information Management (SWIM) framework. It addresses the need to manage changes to SWIM-enabled service components due to evolving business requirements, technological advancements, modifications to common infrastructure, or system upgrades.

The SWIM artifacts versioning defined in this specification outlines both the syntax (structure) and semantics (meaning) for a service identifier.

This document supersedes "SWIM-005, Artifacts Versioning for SWIM-enabled Services, Version 1.0.0 (December 18, 2015)"[6] and provides updated standards that reflect current technologies and methodologies.

This document is prepared in accordance with FAA-STD-067 [1].

1.2 Overview

1.2.1 Background

Initially developed in 2015, the previous specification ("SWIM-005, Artifacts Versioning for SWIM-enabled Services"[6]) required updating to reflect current industry standards and terminologies, particularly in the context of RESTful APIs and related technologies, which have become more prevalent since the previous version.

This specification aims to provide a comprehensive framework for versioning SWIM-enabled services and their associated artifacts, reflecting current technologies and methodologies. It facilitates the coexistence of multiple versions, allowing for phased rollouts, backward compatibility, and seamless transitions between versions.

1.2.2 Basic Concepts

1.2.2.1 Versioning

In the context of Service Oriented Architecture (SOA), versioning refers to managing changes to services and their associated artifacts over time. It allows different versions of a service to coexist, ensuring that existing clients are not adversely affected by modifications.

The intrinsically agile nature of SWIM as a multi-organizational service-oriented technological framework places special emphasis on the need for managing changes to services and service-related artifacts. These changes usually originate from new business requirements, constantly evolving technological solutions, or modifications and upgrades to a common infrastructure.

An important but challenging problem in a multi-organizational technological framework of reusable

and shared services such as SWIM is the coexistence of multiple releases of active services. Addressing these challenges results in the need for a robust service versioning scheme—a set of rules for assigning version identifiers to services or service-related artifacts to manage their changes effectively.

Controlled evolution of services is another significant benefit of versioning in the SWIM environment. As services evolve to incorporate new technologies and features, versioning provides a structured way to manage these changes, allowing for phased rollouts and minimizing the risk of errors or incompatibilities.

By enabling different versions to coexist, SWIM can also mitigate risks associated with implementing new changes. If issues arise with a new version, services can revert to a previous stable version without significant operational impact.

In summary, versioning is a fundamental practice in the SWIM environment, essential for maintaining the seamless and reliable operation of the NAS by supporting interoperability, backward compatibility, and the controlled evolution of services.

1.2.2.2 **Version Identifier**

The service identifier is a fundamental concept in versioning, defined as a unique value consisting of digits, alphabetic characters, or both, associated with a specific version of an artifact.

The FAA uses sequence-based service identifier schemes, such as using letters of the English alphabet following a document identifier (e.g., Order FAA 1800.66A, FAA-STD-073B) for tracking new releases of documentation, and a sequential numbering model for software releases. However, these sequence-based version identifier models have the disadvantage of not providing information about the level of significance of changes and how those changes may affect a consumer.

In contrast, this specification prescribes the use of semantic versioning (aka SemVer) [9], a version scheme [7] that encodes a version in a three-part version number (*major.minor.patch*), with each number indicating the level of significance of changes in a new version. All changes that can be made are classified into three major categories: *major*, *minor*, and *patch*.

Where:

- The *major* version number is incremented for significant changes that break backward compatibility.
- The *minor* version number is incremented for backward-compatible feature additions or enhancements.
- The *patch* version number is incremented for backward-compatible bug fixes or typographical errors.

This approach provides clear and structured information about the impact of changes on backward compatibility and helps consumers make informed decisions about upgrading to a new version.

1.2.2.3 **Key Artifacts**

Resources that are typically subjected to versioning in the SWIM environment include, but are not limited to:

- **Services and Service Interfaces:** Versioning services enables multiple versions of services and service interfaces to coexist, facilitating backward compatibility and phased rollouts of new features or bug fixes.
- **RESTful APIs:** RESTful APIs expose a set of resources and operations through HTTP methods and URIs. Versioning RESTful APIs involves managing changes to resource representations, request/response formats, and API behaviors. Versioning RESTful APIs ensures compatibility and allows for the evolution of the API without disrupting existing clients.
- **Data Format Specification:** A document that defines the structure and format of data exchanged between services and other components (e.g., XML/JSON schemas, exchange data models). Versioning of data format specifications ensures compatibility between service versions and client implementations.
- **Message Headers:** A message header contains a structured collection of key-value pairs allowing services and consumer agents to convey essential information about exchanged messages, such as content type, routing, and caching instructions, etc. Versioning message headers ensures proper interpretation and routing of messages across service versions. This versioning applies to the encompassing artifacts the message header describes, such as services or APIs, depending on the technological solution.
- **Service Descriptions:** A service description is an authoritative source of information on how to use a service. It can be human-readable or machine-processable, generally based on FAA, industry, and international standards. Versioning service descriptions communicates changes to services, service interfaces, or other client-facing aspects.
- **URIs/URLs:** A version identifier embedded in a URI/URL often serves as part of a resource identifier and an endpoint for accessing a service or API on the web. Versioning URIs/URLs ensures proper routing and resource identification across service versions.
- **Other Documentation:** Detailed documentation, such as service requirements, design and implementation documents, Service Level Agreement (SLA), and etc. are versioned to reflect changes in service behavior, usage, or requirements.

1.3 Intended Audience

This document is intended for:

- Developers and architects designing and implementing SWIM services and interfaces;
- SWIM implementers developing service documentation;
- SWIM analysts and systems architects overseeing SWIM architectures.

A basic understanding of SOA principles, interactions among SWIM-enabled service components, REST APIs, and HTTP is assumed.

1.4 Notational Conventions

The keywords "SHALL", "SHALL 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 [12]. 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.

artifact: Any resource or asset that is subject to versioning in the SWIM environment, such as services, interfaces, data contracts, message headers, service descriptions, URIs/URLs, and documentation.

backward compatibility: The ability of a system or component to interoperate with older versions of itself or other systems or components.

dereferenceable URI: A URI is called dereferenceable when it can be used to obtain a representation of the resource (e.g., a system or a document) it identifies using the Internet protocol designated by the URI scheme.

Java Message Service (JMS): A Java-based application programming interface (API) that provides a common way for Java programs to create, send, receive, and read an enterprise messaging system’s message [8].

NAS Enterprise Messaging Service (NEMS): A NAS-based implementation of message-oriented middleware (MOM) that is used by FAA to distribute messages among information consumers and providers, as well as providing administrative functionality that includes (but is not limited to) fault tolerance, load balancing, mediation and orchestration support [8].

NAS Service Registry/Repository (NSRR): An FAA SWIM-supported capability for making services visible, accessible, and understandable across the NAS. NSRR supports a flexible

mechanism for service discovery, an automated policies-based way to manage services throughout the services lifecycle, and a catalog for relevant artifacts [8].

RESTful API: An API that adheres to the principles of REST (Representational State Transfer) and allows interaction with RESTful web services.

SWIM-enabled service: A service that is integrated into SWIM framework. It is designed to facilitate the exchange of information within the NAS by adhering to SWIM standards, protocols, and infrastructure.

Uniform Resource Identifier (URI): A sequence of characters designed for the unambiguous identification of resources via the URI syntax rules and naming schemes prescribed by RFC 3986. [13]

versioning: The practice of managing changes to services and their associated artifacts over time, allowing different versions of a service to coexist [14].

version identifier: A unique name or number that denotes a particular version of a service or service-related artifact.[6]

1.6 Order of Precedence

In the event of a conflict between the text of this document and the references cited herein, the text of this document takes precedence. Nothing in this document, however, supersedes applicable laws and regulations unless a specific exemption has been obtained.

2 Applicable Documents

2.1 Government Documents

[1] FAA Standard Practice, Preparation of Specifications, FAA-STD-067, December 4, 2009, https://www.tc.faa.gov/its/worldpac/standards/faa-std-067.pdf

[2] System-Wide Information Management (SWIM) Governance Policies, Version 2.0, March 2014, http://www.faa.gov/nextgen/programs/swim/governance/standards/media/Governance-Policies-v20.html

[3] FAA-STD-065 Rev. B, Preparation of Web Service Description Documents, FAA; July 15, 2019 https://www.faa.gov/sites/faa.gov/files/air_traffic/technology/swim/governance/FAA-STD-065B%207_15_2019.pdf

[4] FAA-STD-073A, Preparation of Java Messaging Service Description Documents (JMSDD); FAA; December 9, 2019
https://www.faa.gov/sites/faa.gov/files/air_traffic/technology/swim/governance/FAA-STD-073A%20FINAL%2012_9_19.pdf

[5] FAA-STD-075, Creating Service Identifiers; FAA; June 29, 2021
https://www.faa.gov/sites/faa.gov/files/air_traffic/technology/swim/governance/FAA-STD-075%20FINAL%206-29-2021.pdf

[6] SWIM-005, Artifacts Versioning for SWIM-enabled Services, Software Specification, Version 1.0.0; FAA SWIM; December 18, 2015
https://www.faa.gov/sites/faa.gov/files/air_traffic/technology/swim/governance/SWIM%20Service%20Versioning%20Spec.pdf

2.2 Non-Government Publications

[7] Best Practices for Artifact Versioning in Service-Oriented Systems; TECHNICAL NOTE; Software Engineering Institute Marc Novakouski, Grace Lewis, William Anderson, Jeff Davenport; January 2012;
http://www.sei.cmu.edu

[8] SWIM Controlled Vocabulary (CV), V 1.0.0; SWIM FAA; 2019-03-25
https://semantics.aero/pages/swim-vocabulary.html#toc

[9] Semantic Versioning 2.0.0; Tom Preston-Werner; Creative Commons; 2013;
http://semver.org/spec/v2.0.0.html

[10] Web Services Description Language (WSDL) Version 2.0 Part 0: Primer; W3C Recommendation 26 June 2007
https://www.w3.org/TR/2007/REC-wsdl20-primer-20070626

[11] OpenAPI Specification v3.1.0, Version 3.1.0; 15 February 2021
https://spec.openapis.org/oas/v3.1.0

[12] 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

[13] RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*, Network Working Group, January 2005 http://www.rfc-editor.org/rfc/rfc3986.txt

[14] IEEE 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology; IEEE Computer Society; 1990-12-31
https://standards.ieee.org/ieee/610.12/855/

3 General Requirements

3.1 General Requirements

This section specifies the requirements that are common to all SWIM-enabled services, interfaces/APIs and their associated artifacts.

3.1.1 Every release of a SWIM service interface or service-related artifact SHALL be versioned according to this specification.

3.1.2 A version identifier SHALL be formatted as three dot-separated positive integers without leading zeroes (e.g., “2.2.1”):

- The first digit represents a major change.
- The second digit represents a minor change.
- The third digit may represent a patch.

Note: This structure ensures clear communication of changes. For example, a service initially released as version 1.0.0 might undergo minor updates (1.1.0, 1.2.0) that do not disrupt existing clients. Major changes would necessitate an update to version 2.0.0, requiring clients to adapt.

Figure 1 presents a scenario wherein a service consumer agent was designed to use version 1.0.0 of some service. After minor changes, which were communicated through the incremented second and third digits in the version identifier (1.1.0 and 1.1.1 respectively), the consumer agent could continue to use each subsequent version of this service. However, after major changes were made (and communicated via the incremented first digit), the consumer agent required modification in order to use version 2.0.0 of the artifact.

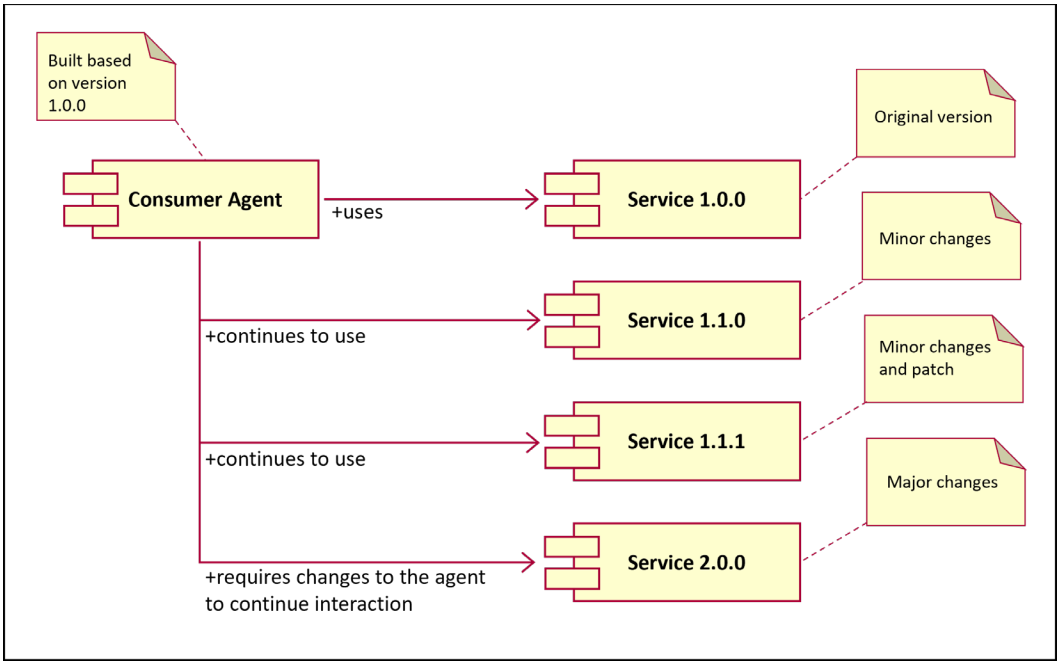

Figure 1 Example of applying described versioning approach

3.1.3 The version identifier SHALL always include a major version number as a positive integer without leading zeroes (e.g., “1”).

Note: The major version number signals significant changes that break backward compatibility, aiding in effective version management and update decisions.

3.1.4 The version identifier SHOULD include a minor version number as a dot-separated positive integer without leading zeroes following the major version number (e.g., “1.1”). However, it may be omitted if only major changes are communicated.

Note: The minor version number indicates backward-compatible feature additions or enhancements. While recommended for clarity, its inclusion is not mandatory when focusing solely on major updates.

3.1.5 The version identifier MAY include a patch version number as a dot-separated positive integer without leading zeroes following the minor version number (e.g., “1.1.1”).

Note: The patch version number identifies backward-compatible bug fixes or fixing typographical typos in case of a document. Including this number is optional and should be decided based on the context of the updates.

3.1.6 A version identifier SHALL be presented as part of the versioned artifact, independent of any additional tooling.

Note: Versioning data should be autonomous, ensuring consistent identification without

reliance on external tools or repositories.

4 Detailed Requirements

4.1 Detailed Requirements

4.1.1 Services and Service Interfaces

4.1.1.1 All SWIM-enabled services SHALL be versioned; that is, a service identifier SHALL be assigned as prescribed in section 3.1 of this specification.

4.1.1.2 The service version identifier SHALL be included in service metadata stored and presented by the NSRR.
Note: SWIM Governance Policies [2] mandate registering all SWIM services in the NSRR.

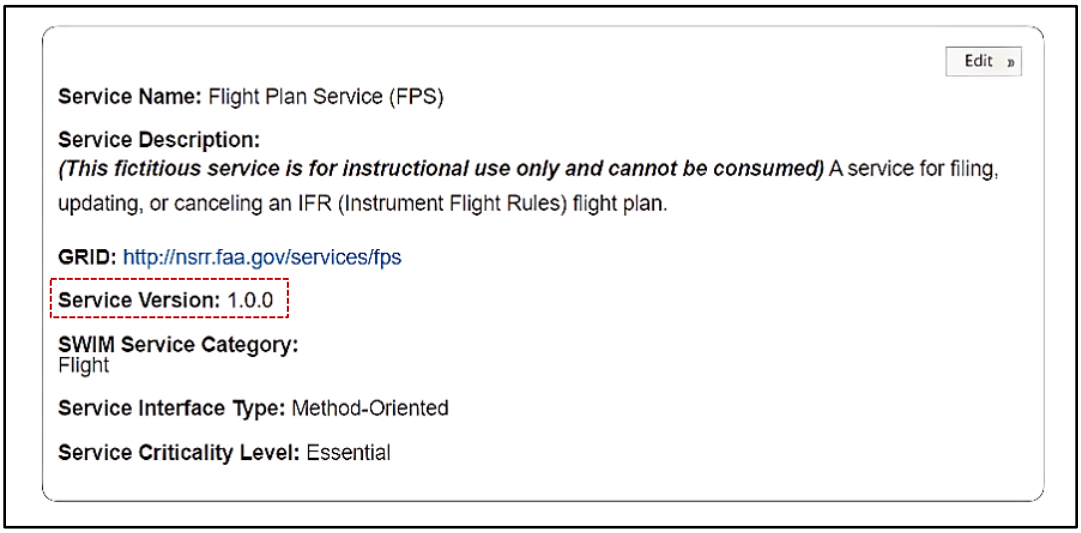

Figure 2 Example of versioning a service in the NSRR.

4.1.1.2.1 In the event of a conflict between the service version presented in the NSRR and other service-related artifacts, the version shown in the NSRR SHALL take precedence.

4.1.2 RESTful APIs

4.1.2.1 Each REST API SHALL include the version identifier in the URI path (e.g., `http://swim.faa.org/services/fps/2.3`).

4.1.2.2 REST APIs SHOULD support versioning through a custom header.

4.1.2.3 REST APIs SHOULD use the "API-Version" HTTP header to identify the version of the API being requested (e.g., `API-Version: 2.3`).

4.1.2.4 REST APIs MAY include a version identifier as a query parameter (e.g., `http://example.org/api/flights?version=2.3`)

4.1.3 Data Exchange Schemas

This section specifies requirements that should be applied during the development of SWIM XML and JSON schemas. These requirements are not applicable to the schema models developed by external (non-FAA) organizations, which, from the perspective of this specification, are immutable.

4.1.3.1 XML Schema

4.1.3.1.1.1 XML schemas SHALL define a unique namespace URI that includes the version identifier.

4.1.3.1.1.2 Each XML schema SHOULD include a `version` attribute in the root element to specify the version.

4.1.3.1.1.3 XML schemas MAY include documentation within the schema file that specifies the version including brief description of the version.

```
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://faa.gov/fps/1.0"
    version="1.0">
    <xs:annotation>
        <xs:documentation>
            Version 1.0 - Initial version.
        </xs:documentation>
    </xs:annotation>
       ...
</xs:schema>
```

Figure 3 Example of versioning of an XML schema

4.1.3.2 JSON Schema

4.1.3.2.1.1 Each JSON schema SHALL include a `$id` attribute formatted as a URI of the schema that includes a version identifier.

4.1.3.2.1.2 The `$id` attribute in a JSON schema MAY be presented as a URL dereferenceable to the schema location.

4.1.3.2.1.3 Each JSON schema SHALL include a `version` property within the schema to specify the version.

```
{
 "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "http://faa.gov/fps/1.0.0/flight-plan.json",
  "version": "1.0.0",
}
```

Figure 4 Example of versioning a JSON schema

4.1.4 Service Description

Service descriptions are essential components of the SWIM framework, providing crucial information about services. These descriptions can be categorized into two types: *machine-processable* and *human-readable*. Machine-processable descriptions, such as Web Services Description Language (WSDL) document and OpenAPI Specification (OAS), are designed for automated processing and integration. Human-readable descriptions are intended for manual review and understanding, typically documented in formats like PDFs or Word documents.

4.1.4.1 Machine-processable service descriptions

4.1.4.1.1 WSDL

This section specifies requirements that should be applied during the development of Web Service Description Language (WSDL) [10] documents for SWIM Web services. WSDL documents are used to describe the functionality offered by a web service, providing a machine-processable format that allows for automated integration and interaction. These documents are based on the WSDL specification, which defines how to describe web services in a standardized way, ensuring interoperability between different systems.

4.1.4.1.1.1 Each WSDL file SHALL include a `version` attribute in the definitions element to specify the version.

4.1.4.1.1.2 WSDL files SHALL define a unique namespace URI that includes the version identifier.

4.1.4.1.1.3 WSDL files SHALL include documentation within the file that specifies the version and provides a summary of changes.

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  targetNamespace="http://faa.gov/swim/fps/wsdl/1.0"
                  version="1.0">
    <wsdl:documentation>
        Version 1.0 - Initial version.
    </wsdl:documentation>
       …
```

Figure 5 Example of versioning a WSDL document

4.1.4.1.2 OpenAPI

This section specifies requirements that should be applied during the development of service descriptions for SWIM RESTful APIs. The OpenAPI Specification (OAS) version 3.1.0 [11], formerly known as the Swagger Specification, is used for presenting these requirements. OAS is generally considered the most widely adopted and supported standard for defining RESTful APIs.
Note, this specification does not prescribe the usage of OAS as a SWIM standard.

4.1.4.1.2.1.1 Each OpenAPI document SHALL include an `id` attribute that is formed as URI which includes the version identifier.

4.1.4.1.2.1.2 Each OpenAPI document SHALL include a `version` attribute in the `info` object to specify the version.

```
{
  "openapi": "3.1.0",
  "info": {
    "title": "Flight Plan Service API",
    "version": "1.0.0"
  },
       ...
}
```

Figure 6 Example of versioning an OpenAPI document

4.1.4.2 Human-Readable Service Descriptions

According to the SWIM Governance Policies document, all SWIM services are required to provide one of two service description documents. Those documents SHALL comply with either FAA standard FAA-STD-065 Rev. B, "FAA-STD-065 Rev. B, Preparation of Web Service Description Documents (WSDD)" [3] or FAA standard FAA-STD-073A, "Preparation of Java Messaging Service Description Documents (JMSDD)"[4] depending on the service's technological solution.

4.1.4.2.1.1.1 All service descriptions SHALL be versioned; that is, a service identifier SHALL be assigned as prescribed in section 3.1 of this specification.

4.1.4.2.1.1.2 The service identifier SHALL be displayed on the title page of a service description document.

4.1.4.2.1.1.3 The service identifier SHOULD be included in the headers of the service description document.

4.1.5 Message Headers

A message is a basic unit of communication from one software agent to another, sent in a single logical transmission. Each message includes a message header, which precedes the message body and contains a structured collection of key-value pairs that convey essential information about the message, such as content type, content length, and other metadata.

Currently, the NAS Enterprise Messaging Service (NEMS), a NAS-based implementation of message-oriented middleware, is the most prevalent method for distributing messages among information consumers and providers within the SWIM ecosystem.

Additionally, SWIM is exploring the potential use of RESTful Web Services or APIs as an alternative solution for delivering SWIM data to consumers. Note, the requirements outlined in this section are forward-compatible, developed in anticipation of the future deployment of RESTful APIs within the SWIM ecosystem.

4.1.5.1 **Java Message Service (JMS)**

This section outlines the requirements that should be applied during the development and deployment of a JMS message header. JMS is a Java-based API employed by NEMS to create, send, receive, and read messages within the context of the SWIM ecosystem.

4.1.5.1.1 Each JMS message header SHALL include a version identifier to specify the version of the SWIM service that originates the message.

Note: the following example is non-normative.

```
// The service version as a JMS message property may be included in the message header like
this:
message.setStringProperty("serviceVersion", "2.1.1");


// Assuming 'message' is the received JMS message
String serviceVersion = message.getStringProperty("serviceVersion");
System.out.println("Service Version: " + serviceVersion);


When the above code is executed, it would print the value of the serviceVersion property to
the console or log, for example:
Service Version: 2.1.1
```

Figure 7 Example of including version identifier in JMS message header

4.1.5.2 **Hypertext Transfer Protocol (HTTP)**

This section outlines the requirements that should be applied during the development and deployment of a HTTP message headers. RESTful APIs are built on top of HTTP, meaning that every RESTful API utilizes HTTP for communication. These APIs leverage HTTP messages, headers, and methods to perform operations on resources. Consequently, the constructs of all RESTful API

messages—and their respective headers—are inherently defined by Hypertext Transfer Protocol, version 1.1.

4.1.5.2.1 The HTTP message header SHALL include a version identifier to specify the version of the API being used (e.g., `API-Version: 1.0.0`)

Note: Also, refer to section 4.1.6.1, which requires the inclusion of the version number in the URI path.

4.1.6 Uniform Resource Identifier (URI)

A Uniform Resource Identifier (URI) is a string of characters used to identify a resource on the internet [13]. In the SWIM framework, URIs support versioning for most artifacts described in this specification. A new version of an artifact generally requires the creation of a new URI or the modification of an existing one, such as an API endpoint, the namespace in an XML schema, or a service identifier in the NSRR.

According to FAA standard FAA-STD-75 [5], a URI shall be declared as an HTTP-based URI. This means it mandates rendering a URI used for identifying services and/or service-related artifacts with the default value for the scheme element "`http`." This approach allows the URI to serve as a globally unique identifier and to obtain the location of a resource (system or document).

4.1.6.1 Each URI SHALL include the version number in the path to distinguish different versions of the artifact (e.g., `http://faa.gov/swim/api/fps/1.0/flights`).

4.1.6.2 The value of the URI that identifies artifacts described in this specification SHOULD be dereferenceable.

4.1.7 Other Documentation

This section outlines the versioning procedure for all documents or artifacts not mentioned in sections 4.1.2 through 4.1.6. Documents that fall into the category of "Other Documentation" may include, but are not limited to, requirements, design documents, Service Level Agreements (SLAs), policies, and other related artifacts.

4.1.7.1 Each document SHALL include a version identifier formatted as prescribed in section 3.1.

4.1.7.2 This identifier SHOULD be displayed on the title page of the document.

4.1.7.3 The version identifier SHOULD be included in the header of each page of the document to ensure that the version is clear and easily identifiable throughout the document.

4.1.7.4 The use of the versioning scheme as described in this specification does not preclude the use of additional versioning or naming schemes for internal development or tracking purposes. However, the version identifier format prescribed in this document SHALL be used for all externally facing documents.

5 Notes

This specification makes no assertions about a correlation between versioning of a service or an interface and versioning of the documents that are used to describe or represent the service.

No retroactive application of these guidelines is envisioned for currently published services' versions. Existing version identifiers will be exempted.

The versioning scheme described in this specification is intended to be applied consistently across all SWIM-enabled services and artifacts. However, it is recognized that there may be cases where deviations from this scheme are necessary due to specific requirements or constraints. In such cases, the rationale for the deviation should be clearly documented and communicated to all relevant stakeholders.

6 Quality Assurance Provisions

All artifacts described in this specification are registered or intended to be registered or uploaded in the NSRR.

The artifacts presented for uploading to the NSRR will be subject to verification that they conform to the requirements prescribed by this specification.

- Verification will be performed by inspection by the members of the SWIM Governance team using the following criteria.
  Conformance to versioning guidelines.
- Consistency of version identifiers with documented changes.
- Completeness and accuracy of the provided artifacts.

Regular reviews and updates of the versioning process will be conducted based on feedback and evolving requirements.

7 Preparations for Delivery

Delivery of versioned artifacts within the SWIM ecosystem will be systematic, secure, and transparent, ensuring that all stakeholders have access to the latest versions and related information.

Artifacts will be stored in a secure digital repository, such as the NAS Service Registry/Repository (NSRR), ensuring they are accessible to authorized users.

Versioned artifacts will be uploaded to the NSRR for official registration and archival as prescribed by SWIM Governance Policies [2].

Re-versioned artifacts will be presented to relevant stakeholders through appropriate change boards or forums.