OGC® DOCUMENT: 21-039R1

External identifier of this OGC® document: http://www.opengis.net/doc/PER/t17-D002

Open Geospatial Consortium

TESTBED-17: AVIATION API ER

ENGINEERING REPORT

PUBLISHED

Submission Date: 2021-11-19
Approval Date: 2021-12-17
Publication Date: 2022-01-21
Editor: Sergio Taleisnik

License Agreement

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD. THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications. This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Copyright notice

Note

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

I ABSTRACT

This Testbed-17 (TB-17) Aviation API Engineering Report (ER) summarizes the implementations, findings and recommendations that emerged from the efforts of building a definition for an Aviation API compliant with the requirements of the OGC Standards Program, and the exploration of the potential of aviation data fusion.

This ER describes the nine façades built to interface SWIM services and serve aviation data through OGC APIs, the two services built to consume SWIM data and fuse it to generate richer datasets while serving the fused data through OGC APIs, the client application built to display data retrieved from the façades and fusion services, and the development client built to focus on functionality and experimentation.

Finally, this ER discusses the potential of OGC APIs to help standardize the access to aviation data within the context of the System Wide Information Management (SWIM) program.

II EXECUTIVE SUMMARY

The System Wide Information Management (SWIM) Data Services of the Federal Aviation Administration (FAA) currently produce data from the National Airspace System (NAS) to consumers using various protocols and service offerings in both synchronous and asynchronous messaging formats. These services are documented and described in the FAA's NAS Service Registry/Repository (NSRR). Consumers can access the NSRR to obtain this information for each SWIM service offering in order to develop their business applications. Often, business applications require multiple SWIM data; therefore, efforts have been made to standardize the SWIM data models for their domains (i.e. weather, aeronautical, and flight) based on XML standards such as Weather Information Exchange Model (WXXM), Flight Information Exchange Model (FIXM), and Aeronautical Information Exchange Model (AIXM).

Previous OGC work has addressed the challenges of increasing interoperability between aviation data services. Recently, the OGC community has developed a new family of standardized OpenAPI-based Web APIs for the various geospatial resource types with the potential to enhance the way in which consumers can access geospatial data from various sources. Testbed-16 brought together previous work on the development of OGC APIs, the use of semantics to enrich data, and SWIM data processing, and demonstrated an OpenAPI-based API serving SWIM data. Testbed-17 took many lessons learned and recommendations from Testbed-16 and focused on further testing the value of standards-based APIs within the SWIM program.

The research questions for the Aviation API Task were:

- What are the benefits and downsides of:
    - Transitioning current APIs of SWIM services into APIs compliant with current and emerging OGC API Standards?
    - Creating a new OGC API standard for aviation data?
    - Implementing existing OGC API standards to create APIs serving aviation data?
- Do these benefits and downsides change between:
    - Serving raw aviation data and serving fused aviation data?
    - Working with a client from a domain expert perspective and working with a client from a developer perspective?
    - Working with aeronautical data and working with flight data?
- What lessons can be learned from the usage of Linked Data or different encodings?

To answer these questions, the Aviation API Task was organized into the development and testing of a system of seven interconnected components, as seen on Figure 1:

- **Façades for SWIM services**, retrieving aviation data from multiple SWIM services (or static sources) and serving these data through OGC APIs. Three Façades comprising nine API components were built.
    - The Aeronautical Data APIs (identified collectively as *D104*): Four APIs built to serve NOTAMs, Airport Layouts and Airspaces
    - The Flight Positions Data APIs (identified collectively as *D105*): Three APIs built to serve flight positions from the STDDS, SFDPS, and TFMS Services
    - The International Flight Data APIs (identified collectively as *D107a* and *D107b*): Two APIs built to serve flight plans from the SFDPS (FAA) and NMB2B (EUROCONTROL) Services
- **Components that fuse raw aviation data**, and serve the fused data through OGC APIs. Two fusion components were built, each one featuring an API.
    - The Flight Restrictions Data Fusion Service (identified as *D106*): Built to identify flight restrictions from past or future flights.
    - The International Flight Data Fusion Service (identified as *D107*): Built to fuse flight plans from a same flight spanning across the US and European airspaces which are commonly served separately.

- **Client components which present aviation data to end users**. Two clients were built: One meant to serve an aviation domain expert, and the other to serve a developer of aviation software applications.
    - The Aviation Domain Expert Client (identified as *D108*): A client built to display aviation data to domain experts in a 3D map environment.
    - The Aviation Developer Client (identified as *D109*): A client built to explore the provision of data for the sake of helping developers access API information.

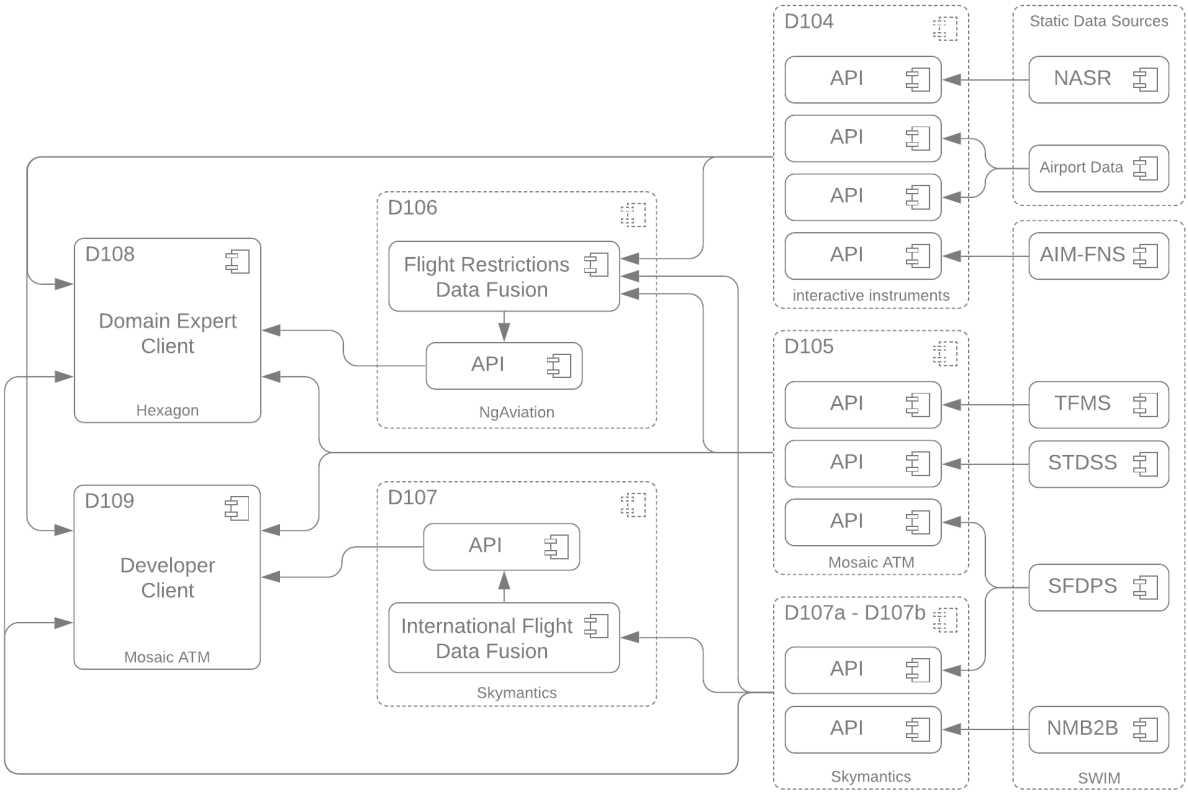


Figure 1 — Component Diagram for the Aviation API Task

All components were successfully developed and tested. Eleven APIs were built implementing several OGC API Standards. The lessons learned throughout the Testbed are captured in this ER and help respond the research questions. The following is a set of recommendations for future work:

- **Continue Evaluating the Potential of OGC API Standards in the Exchange of SWIM Data:** Future work should be coordinated with the Aviation API SWG to continue evaluating OGC API Standards in the exchange of aviation SWIM data. APIs could also be explored for more aviation data types (such as weather), or data with diverse levels of volume or update frequency.

- **Explore Additional Aviation Data Fusion Scenarios:** Future work should explore additional data fusion scenarios within the aviation domain to continue gathering lessons learned for further development of API standards, for proposing new aviation fusion services, and for defining decision-making rules for the construction of fusion services.
- **Continue Exploring JSON-FG Within the Aviation Domain:** Future work should be coordinated with the Features and Geometries JSON SWG to continue exploring the uses of JSON-FG with aviation data to ensure the emerging standard is able to support the entire scope of aviation use cases, as well as more advanced geometries.
- **Develop Aviation Vocabularies, Schemas and Ontologies:** Future work should expand current aviation vocabularies, schemas and ontologies through the exploration of use cases and scenarios that require them.
- **Demonstrating the Value of Linked Data in the Aviation Domain:** Future work should demonstrate the value of linked data through the development of aviation clients and fusion scenarios that make use of linked data.

III KEYWORDS

The following are keywords to be used by search engines and document catalogues.

ogcdoc, OGC document, Aviation, API, SWIM

IV PREFACE

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

V SECURITY CONSIDERATIONS

No security considerations have been made for this document.

VI SUBMITTING ORGANIZATIONS

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

- Skymantics, LLC

VII SUBMITTERS

All questions regarding this document should be directed to the editor or the contributors:

NAME	ORGANIZATION	ROLE
Sergio Taleisnik	Skymantics, LLC	Editor
Clemens Portele	interactive instruments GmbH	Contributor
Johannes Echterhoff	interactive instruments GmbH	Contributor
Jarrod Lichty	Mosaic ATM	Contributor
Frantisek Albert	NG Aviation SE	Contributor
Ignacio “Nacho” Correas	Skymantics, LLC	Contributor
Felipe Carrillo Romero	Hexagon	Contributor

1 SCOPE

1 SCOPE

This OGC Engineering Report (ER) presents deliverable D002 of the OGC Testbed-17 (TB-17) initiative performed under the OGC Innovation Program.

The ER reports on the activities performed throughout the testbed and provides discussions on the lessons learned. The ER discusses the business value of APIs based on OGC API standards in the purpose of serving aviation data through the SWIM system. This ER also explores the value of fusing raw aviation data into richer aviation datasets, and the consumption of raw and fused aviation data from the perspectives of both an aviation domain expert and a developer of aviation applications.

1.1. What Does This ER Mean for the Working Group and the OGC

The Aviation Domain Working Group (DWG) was tasked with reviewing this ER. The Aviation DWG works on the subjects of aviation-related data interoperability and access.

The Aviation API Standard Working Group (SWG), proposed by the Aviation DWG, was constituted concurrently with TB-17. This approach provided a framework for leveraging the lessons learned and recommendations presented by this Testbed's Aviation participants in order to work on the development and maintenance of an OGC Aviation API Standard.

Despite not being the focus for TB-17, the work on Aviation Linked Data could also be of interest to the Geosemantics DWG, which has been actively working on this subject for several years. This Geosemantics DWG reviewed the Testbed-16 (TB-16) Aviation Task ER due to the strong focus of TB-16 on semantics.

2

TERMS, DEFINITIONS AND ABBREVIATED TERMS

2 TERMS, DEFINITIONS AND ABBREVIATED TERMS

This document uses the terms defined in OGC Policy Directive 49, which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this document and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2.

This document also uses terms defined in the OGC Standard for Modular specifications (OGC 08-131r3), also known as the ‘ModSpec’. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec.

For the purposes of this document, the following additional terms and definitions apply.

2.1. Terms and definitions

2.1.1. **Application Programming Interface (API)**

an interface that is defined in terms of a set of functions and procedures, and enables a program to gain access to facilities within an application [1]

2.1.2. **Web API**

an API using an architectural style that is founded on the technologies of the Web [2]

Note 1 to entry: Best Practice 24: Use Web Standards as the foundation of APIs in the W3C Data on the Web Best Practices [2] provides more detail.

Note 2 to entry: A Web API is basically an API based on the HTTP standard(s).

2.1.3. Standards-based API

an API that conforms to one or more conformance classes specified in one or more standards

Note 1 to entry: Since almost all APIs will conform to some standard, the term is usually used in the context of a specific standard or a specific family of standards. This ER considers Web APIs with a specific focus on the OGC API standards. Therefore, whenever the term is used in this ER, it is meant as an alias for an API that conforms to one or more conformance classes as defined in the OGC API standards.

2.1.4. Standardized API

an API that is intended to be deployed by multiple API providers with the same API definition

Note 1 to entry: The only difference between the API definitions will be the URL(s) of the API deployment. All other aspects are identical (resources, content schemas, content constraints and business rules, content representations, parameters, etc.) so that any client that can use one deployment of the standardized API definition can also use all other deployments, too.

Note 2 to entry: If the API provides access to data, different deployments of the API will typically share different content.

2.1.5. Façade Service

A Façade Service is a component that fetches data from a specific data source and makes it available through its own interface [3]. The main reason for building this type of service is the difficulty or inability to modify the original data source with the intent of modifying:

- The underlying API structure
- The format in which the data is made available.

2.1.6. SWIM Data

Any data provided through the SWIM System.

2.1.7. **SWIM Data Service**

A service within the SWIM System that provides data. [4]

2.2. Abbreviated terms

AIXM	Aeronautical Information Exchange Model
API	Application Programming Interface
ARTCC	Air Route Traffic Control Center
CRS	Coordinate Reference System
ER	Engineering Report
EUROCONTROL	European Organisation for the Safety of Air Navigation
FAA	Federal Aviation Administration
FIXM	Flight Information Exchange Model
FNS	Federal NOTAM System
GUFI	Globally Unique Flight Identifier
JMS	Java Messaging Service
NAS	National Airspace System
NOTAM	Notice to Airmen
OGC	Open Geospatial Consortium
SCDS	SWIM Cloud Distribution Service
SFDPS	SWIM Flight Data Publication Service
STDDS	SWIM Terminal Data Distribution System
SWIM	System Wide Information Management
TB	Testbed
TFMS	Traffic Flow Management System
TFR	Temporary Flight Restriction

TIE	Technology Integration Experiment
WFS	Web Feature Service
WMS	Web Map Service
WMTS	Web Map Tile Service
WXXM	Weather Information Exchange Model

3

INTRODUCTION

3 INTRODUCTION

3.1. Background

3.1.1. SWIM

The System-Wide Information Management (SWIM) initiative supports the sharing of aeronautical, air traffic and weather information by providing communications infrastructure and architectural solutions for identifying, developing, provisioning, and operating a network of highly-distributed, interoperable, and reusable services.

As part of the SWIM architecture, data providers create services for consumers to access their data. Each service is designed to be stand-alone. However, the value of data increases when it is combined with other data. Real-world situations are often not related to data from one but from several SWIM feeds. Having consumers retrieving data from several SWIM services raises the need of interoperability between those services.

3.1.2. OGC API Standards

For several years, the OGC members have worked on developing a family of Web API standards for the various geospatial resource types. These APIs are defined using OpenAPI. As the OGC API standards keep evolving, are approved by the OGC and are implemented by the community, the aviation industry can subsequently experiment and implement them.

The following OGC API Standards and Draft Specifications were used for the development of APIs during this Testbed:

OGC API – Features: a multi-part standard that defines the capability to create, modify, and query vector feature data on the Web and specifies requirements and recommendations for APIs that want to follow a standard way of accessing and sharing feature data. It currently consists of four parts:

- OGC API — Features — Part 1: Core. Approved September 2019, this standard defines discovery and query operations. [5]
- OGC API — Features — Part 2: Coordinate Reference Systems by Reference. Approved October 2020, extends the core capabilities specified in Part 1: Core with the ability to use coordinate reference system identifiers other than the defaults defined in the core. [6]

- Draft OGC API — Features — Part 3: Filtering. Part 3 specifies an extension to the OGC API — Features — Part 1: Core standard that defines the behavior of a server that supports enhanced filtering capabilities. [7]
- Draft OGC API — Features — Part 4: Create, Replace, Update and Delete. Part 4 specifies an extension that defines the behavior of a server that supports operations to add, replace, modify or delete individual resources from a collection. [8]

A specification for version 2 of the Common Query Language (CQL2) is being developed together with Part 3 to standardize a language that is recommended for filter expressions. [9]

OGC API – Processes: An approved OGC API standard, published December 2021, that specifies a Web API that enables the execution of computing processes and the retrieval of metadata describing their purpose and functionality. Typically, these processes combine raster, vector, coverage and/or point cloud data with well-defined algorithms to produce new information. [10]

Draft OGC API – Tiles: This draft API defines how to discover which resources offered by the Web API can be retrieved as tiles, retrieve metadata about the tile set (including the supported tile matrix sets, the limits of the tiled set inside the tile matrix set) and how to request a tile. [11]

Draft OGC API – Styles: This draft API specifies building blocks for OGC Web APIs that enables map servers and clients as well as visual style editors to manage and fetch styles. [12]

3.1.3. Exploration of OGC API Standards by SWIM

For several years, the FAA and the OGC have jointly explored making SWIM data more easily accessible and more valuable. As part of these past efforts, Testbed-16 brought together previous work on the development of OGC APIs, the use of semantics to enrich data, and SWIM data processing. The objectives were to deliver the first demonstration of an OpenAPI-based API serving SWIM data, a component generating aviation Linked Data, and two client applications querying and displaying that data [8].

Two of the TB-16 recommendations were to integrate OGC APIs within SWIM Data Services and to demonstrate interoperability between diverse Aviation APIs [8]. In order to advance these recommendations, TB-17 focused on the development of eleven APIs based on OGC API standards, and the completion of Technology Integration Experiments (TIEs) between these APIs.

During TB-16, the development of the API serving aviation data resulted in numerous lessons learned and recommendations [8]. TB-16 saw the development of one aviation-related API based on an OGC API Standard (OGC API — Features). The APIs developed during TB-17 addressed many of those lessons learned and implemented additional OGC API standards which have been maturing since; this process is reflected in Figure 2.

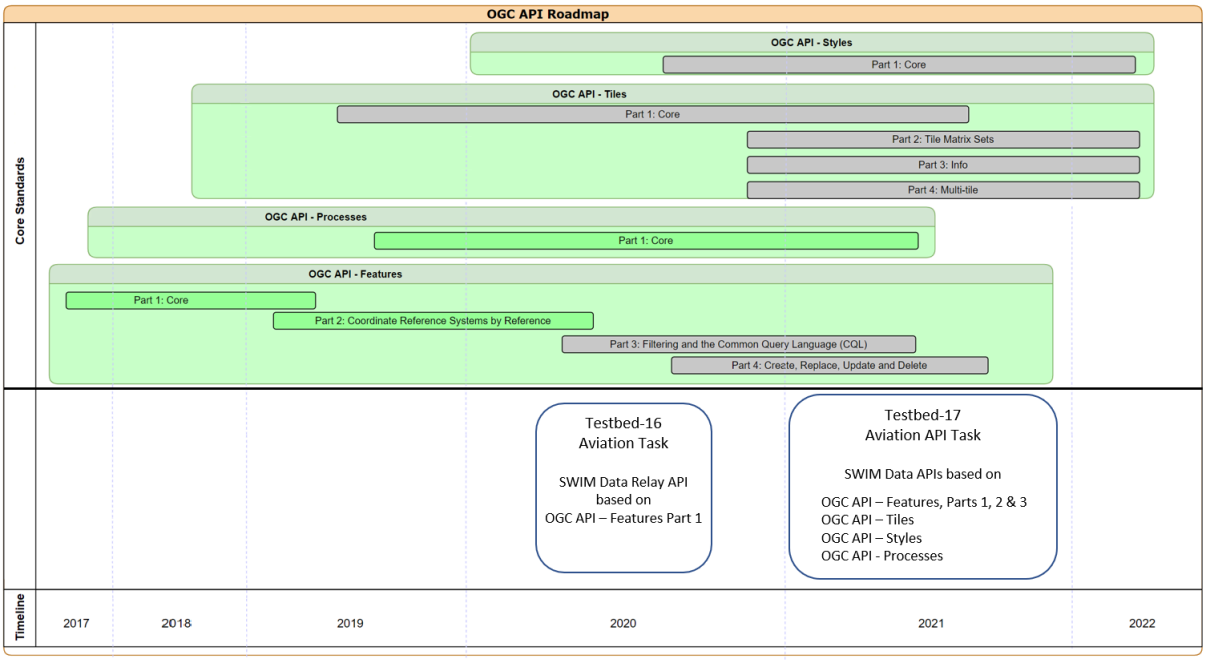


Figure 2 — Timeline of OGC API Standards and its Aviation Demonstrations

3.2. Requirements Statement

Testbed-17 required investigating the potential of OGC API Standards in the context of the SWIM Program.

The original goals of the TB-17 Aviation API Task were the following:

- Explore the potential of OGC Web APIs in the context of SWIM.
- Develop an Aviation API, based on OGC Web API building blocks, that enables convenient access to existing SWIM services.
- Demonstrate the capabilities of the new Aviation API in a data fusion scenario, and to serve aviation raw data.
- Develop two clients: one with a focus on ease-of-use for end-users, and the other with a focus on functionality for developers.
- Build on and stress-test previous Testbed results for semantic interoperability.

As noted, the original requirement for the Aviation task was the development of a draft OGC API specification specific for the exchange of Aviation information. Within the first few meetings after the Testbed began, participants agreed that this might not be a feasible option and therefore engaged into the alternative of using existing approved and draft OGC API standards to create the OGC APIs for the TB-17 components.

This change in the task focus provided a whole new set of questions to be answered. The original idea of developing a specific OGC API standard for Aviation was kept throughout the Testbed in order to contrast it with the usage of existing OGC API standards in aviation. The updated research questions are summarized as follows:

- What are the benefits and downsides of:
    - Transitioning current APIs of SWIM services into APIs compliant with current and emerging OGC API Standards?
    - Creating a new OGC API standard for aviation data?
    - Implementing existing OGC API standards to create APIs serving aviation data?
- Do these benefits and downsides change between:
    - Serving aviation raw data and serving aviation fused data?
    - Working with a client from a domain expert perspective and working with a client from a developer perspective?
    - Working with aeronautical data and working with flight data?
- What lessons can be learned from the usage of different encodings, CRSs, and Linked Data?

The TB-17 Features and Geometries Task worked on the development of potential solutions to problems with JSON and GeoJSON when serving specific sets of data, including aviation data. Many participants from the Aviation Task also participated in the Features and Geometries Task.

3.3. Functional Overview

As shown in Figure 3, the Aviation Task architecture was organized into a system of seven interconnected components. All seven components were developed simultaneously throughout the Testbed, with permanent communication and cooperation among participant organizations.

The components can be divided into three groups:

- Façades for SWIM services, retrieving aviation data from multiple SWIM services (or static sources) and serving these data through OGC APIs. Three Façades comprising nine API components were built.
    - The Aeronautical Data APIs (identified collectively as *D104*): Four APIs built to serve NOTAMs, Airport Layouts and Airspaces
    - The Flight Positions Data APIs (identified collectively as *D105*): Three APIs built to serve flight positions from the STDDS, SFDPS, and TFMS Services
    - The International Flight Data APIs (identified collectively as *D107a* and *D107b*): Two APIs built to serve flight plans from the SFDPS (FAA) and NMB2B (EUROCONTROL) Services
- Components that fuse raw aviation data, and serve the fused data through OGC APIs. Two fusion components were built, each one featuring an API.
    - The Flight Restrictions Data Fusion Service (identified as *D106*): Built to identify flight restrictions from past or future flights.
    - The International Flight Data Fusion Service (identified as *D107*): Built to fuse flight plans from a same flight spanning across the US and European airspaces which are commonly served separately.
- Client components which present aviation data to end users. Two clients were built: One meant to serve an aviation domain expert, and the other to serve a developer of aviation software applications.
    - The Aviation Domain Expert Client (identified as *D108*): A client built to display aviation data to domain experts in a 3D map environment.
    - The Aviation Developer Client (identified as *D109*): A client built to explore the provision of data for the sake of helping developers access API information.

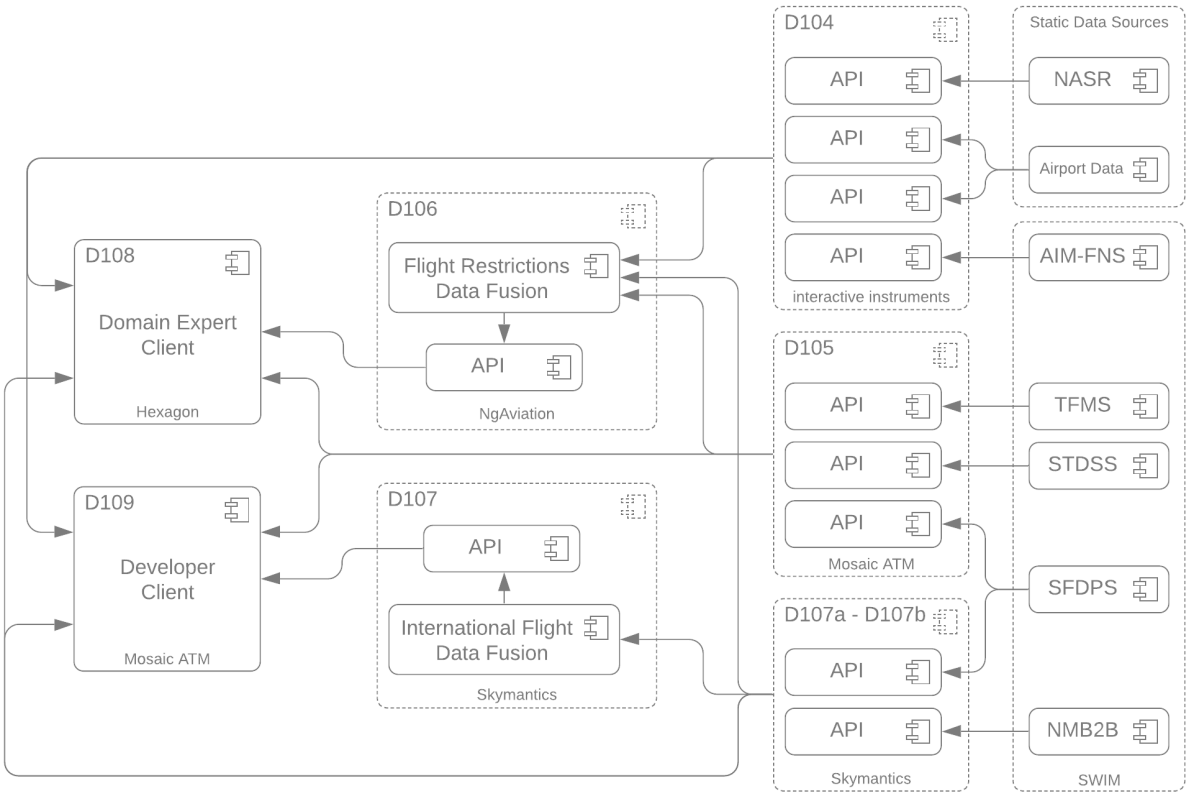


Figure 3 — Component Diagram for the Aviation API Task

The following table summarizes the SWIM data sources accessed by the façades:

Table 1 — SWIM Data Sources Accessed by TB-17 Façades

#	DATA SOURCE NAME	DATA SOURCE DESCRIPTION	FAÇADE ACCESSING THE DATA SOURCE
1	Federal NOTAM System (FNS)	The modernized Notice to Airmen (NOTAM) management system designed to digitize the collection, dissemination, and storage of NOTAMs. The new process moves the responsibility for originating NOTAMs to those who directly observe the hazardous condition(s) within the National Airspace System (NAS). A digital NOTAM describes the temporary status change of a static feature caused by an event within the NAS in a standardized manner enabled by the Aeronautical Information Exchange Model (AIXM).	Clause 4
2	Traffic Flow Management System (TFMS)	TFMS processes all available data sources such as flight plan messages, flight plan amendment messages, and departure	Clause 5

#	DATA SOURCE NAME	DATA SOURCE DESCRIPTION	FAÇADE ACCESSING THE DATA SOURCE
		and arrival messages. The FAA's NAS Data Warehouse assembles TFMS flight messages into one record per flight. TFMS is restricted to the subset of flights that fly under Instrument Flight Rules (IFR) and are captured by the FAA's en route computers.	
3	SWIM Terminal Data Distribution System (STDDS)	Converts legacy terminal data collected from airport towers and Terminal Radar Approach Control (TRACON) facilities into easily accessible information. STDDS is installed at 38 TRACONS within the NAS, providing access to data from over 200 airports, and over 400 individual systems.	Clause 5
4	SWIM Flight Data Publication Service (SFDPS)	Provides en route flight data to National Airspace System (NAS) consumers. SFDPS allows consumers to receive real-time data for analytics, business processes, research, and other activities. Data is provided by data distribution systems located at each of the 20 Air Route Traffic Control Centers (ARTCCs) in the contiguous United States.	Clause 5 and Clause 6
5	Network Manager business-to-business web services (NM B2B)	An interface provided by the EUROCONTROL Network Manager (NM) for system-to-system access to its services and data. NM B2B services include Flight Plan Preparation and Management, Flight Data Retrieval, Departure (DPI) and Arrival (API) Planning Information, among others.	Clause 7

3.3.1. Component Interactions

Two scenarios involving end users were identified — one for each of the two clients developed in this Testbed: The Aviation Domain Expert Client (D108) and the Aviation Developer Client (D109). Both scenarios consist of the end user interfacing a client application to visualize aviation data. These scenarios are meant to satisfy the Testbed requirement of exploring both an end-user and a developer perspective:

On the backend two types of scenarios were identified. On one hand, the Aviation Domain Expert Client retrieves information from the APIs built on top of the fusion components and the APIs built as facades for SWIM services. On the other hand, the Fusion Services retrieve information by connecting to the APIs built as facades for SWIM services. All these interactions are meant to satisfy the Testbed requirement of exploring the potential of OGC APIs within the aviation domain.

4

AERONAUTICAL DATA FAÇADE SERVICE

4 AERONAUTICAL DATA FAÇADE SERVICE

The Aeronautical Façade Service provides Federal Notices to Airmen (NOTAMs) after extracting NOTAMs from the System Wide Information Management System (SWIM). Additionally, this component provides airport and airspace data extracted from various static data sources. All data is provided through APIs designed following approved or draft OGC API Standards. The Aeronautical Façade Service component was created by interactive instruments.

4.1. Internal Architecture

4.1.1. Component Overview

This Façade Service features two data retrieval subcomponents: One for the Federal Notice to Airmen System (AIM-FNS) and another one for airspace and airport static data sources, a database to store the extracted data, and an Idproxy which delivers the APIs that serve the extracted data. Figure 4 describes the Façade Service subcomponents, the data sources (note that for AIM-FNS the retriever also accesses static sources on some occasions), and the TB-17 components consuming from the Façade APIs.

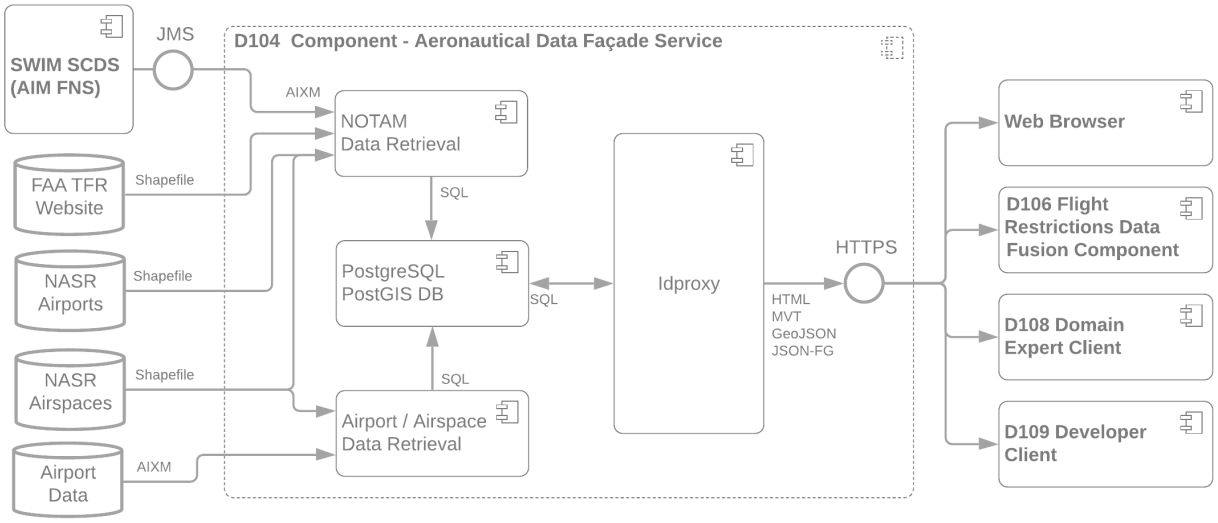


Figure 4 — D104 Component Overview

4.1.2. Idproxy (API Component)

The main component that delivers the Aeronautical Data APIs is Idproxy. Idproxy is a software product, written in the Java programming language, that implements the OGC API family of

standards. ldproxy enables sharing geospatial data using Web APIs based on OGC API standards. The key characteristics of ldproxy are:

- **Easy to use**: The APIs support both JSON and HTML. Users of an API can use their favorite programming environment to access the data or simply use their browser.
- **Browsable**: All content is linked from the landing page of each API. A user can navigate through the API in any web browser and quickly get an understanding of the available data and the API capabilities. Search engines can also index the data.
- **Linkable**: Each data item in the APIs has a stable URI and can be used in external links.
- **Based on standards**: ldproxy is a comprehensive implementation of the emerging OGC API Standards and an increasing number of clients or libraries can use the AAPIs directly. This also applies to the supported formats returned by the APIs, such as GeoJSON, Mapbox Vector Tiles, Mapbox Styles, or TileJSON. In addition, the APIs are documented in a developer-friendly way via OpenAPI 3.0.
- **Certified**: ldproxy is certified as an OGC Reference Implementation for OGC API — Features — Part 1: Core 1.0 and OGC API — Features — Part 2: Coordinate Reference Systems by Reference.
- **Open Source**: The source code is available under the Mozilla Public License 2.0 on GitHub.
- **Multiple Data Sources**: Currently three types of feature data sources are supported: PostgreSQL databases with the PostGIS extension, GeoPackage and OGC Web Feature Services. In addition, map or vector tiles stored in a MBTiles container are supported as a tile source.
- **Extensible**: ldproxy is modular, written in Java 11 and designed to be extended.

The landing page for all four Aeronautical Data APIs that comprise deliverable D104 is https://t17.ldproxy.net/.

All APIs provide access to data published by the US Federal Aviation Administration (FAA).

4.1.3. PostgreSQL/PostGIS (Database)

The data for all APIs is stored in a PostgreSQL/PostGIS database. The aeronautical datasets are:

- All Controlled Airspaces of the Classes B, C, D, and E from the National Airspace System Resource (NASR) Subscription;
- Airport data for the major airports in the United States provided by Hexagon (AIXM 5.1); and
- Notices to Airmen (NOTAMs) received from a Federal Notice to Airmen System (AIM-FNS) subscription in FAA's SWIM Cloud Distribution Service (SCDS), a

cloud-based infrastructure dedicated to providing near real-time FAA SWIM data to the public via Solace JMS messaging.

4.1.3.1. Communication Between ldproxy and PostgreSQL

ldproxy converts API requests to SQL queries and processes the results to convert them to API responses in the GeoJSON, Features and Geometries JSON (JSON-FG), HTML or Mapbox Vector Tile (MVT) formats.

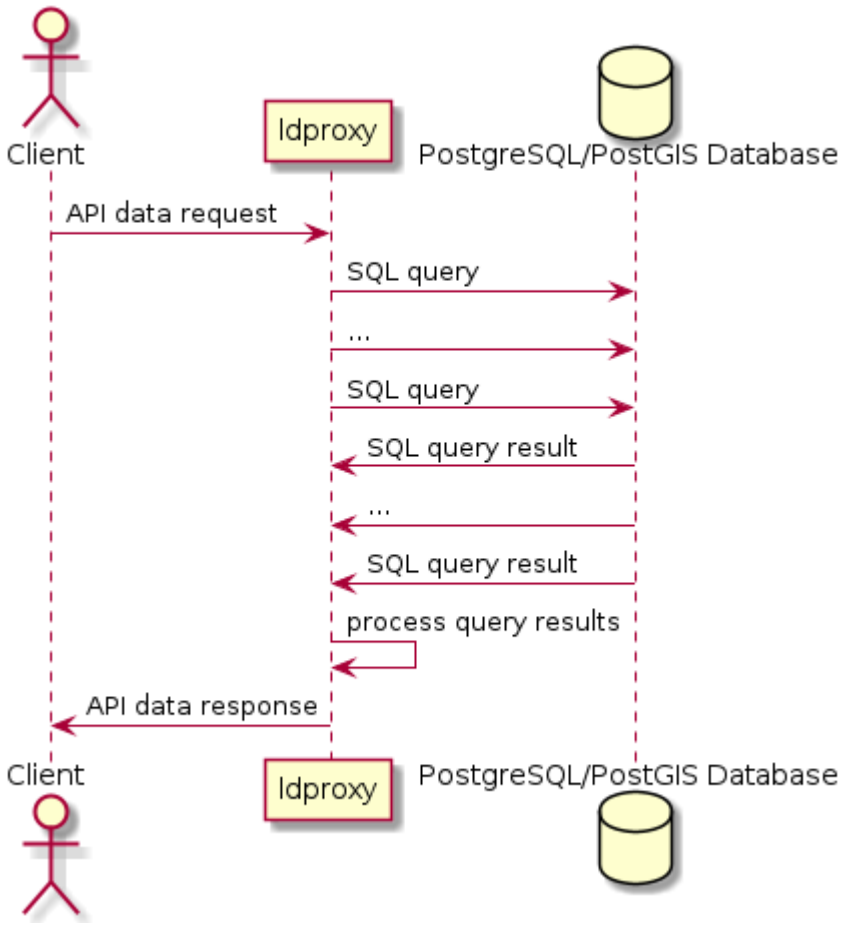


Figure 5 — Information Flow for Data Requests

While the first two datasets are static and do not change, the NOTAMs are dynamic: that is new NOTAMs are added to the database as they are received from the SCDS subscription. Since a new NOTAM may change information that is cached in ldproxy for performance reasons (in particular, the spatial and temporal extent of the NOTAM dataset, but also vector tiles) a database trigger is used to notify ldproxy about a new NOTAM. For TB-17 a new capability was implemented in ldproxy to "listen" for notifications about feature changes. For each new NOTAM, the spatial and temporal extents are evaluated and if needed the extents of the NOTAM feature collection are updated. In addition, vector tiles that include the spatial extent are invalidated in the tile cache.

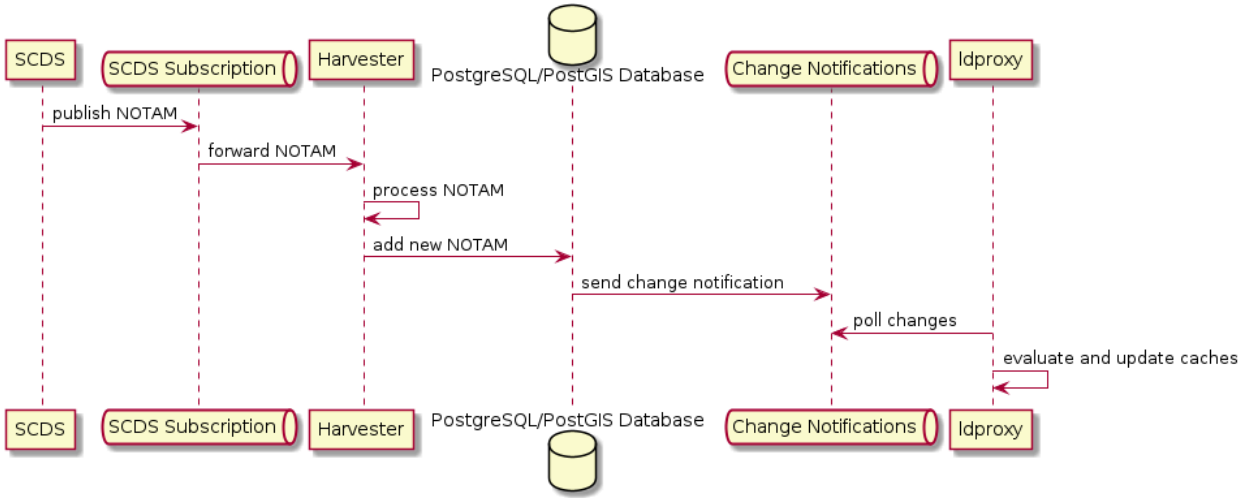


Figure 6 — Communicating Data Changes to ldproxy

The communication between ldproxy and PostgreSQL uses the standard PostgreSQL protocol, which is TCP/IP-based.

4.1.4. Airport and Airspace Data Retrieval

The airport and airspace datasets have been loaded into the database using the GDAL `ogr2ogr` tool.

The airspace data was available as a Shapefile and the schema in the database is the same as the schema for the Shapefile.

The airport data was provided as an AIXM 5.1 XML document for each airport. The data was loaded into the database with one table per AIXM feature type. Since every feature has a single time slice, the time slices were flattened. An additional column `airport` was added for each table to identify the airport to which the feature belongs. For example, the table definitions for two AIXM feature types in the database:

```
CREATE TABLE public.airportheliport (
    ogc_fid integer NOT NULL,
    gml_id character varying NOT NULL,
    identifier character varying(11),
    validtimebegin timestamp(0) with time zone,
    interpretation character varying(24),
    sequencenumber integer,
    correctionnumber integer,
    featurelifetimebegin timestamp(0) with time zone,
    designator character varying(24),
    name character varying(64),
    aixmtype character varying(2),
    wkb_geometry public.geometry(Geometry,4326),
    airport character varying(4)
);
```

Figure 7 — Table Definition of AirportHeliport Features

```
CREATE TABLE public.apron (
```

```
    ogc_fid integer NOT NULL,
    gml_id character varying NOT NULL,
    identifier character varying(11),
    validtimebegin timestamp(0) with time zone,
    interpretation character varying(24),
    sequencenumber integer,
    correctionnumber integer,
    featurelifetimebegin timestamp(0) with time zone,
    name character varying(21),
    composition character varying(24),
    associatedairportheliport character varying(24),
    airport character varying(4)
);
```

Figure 8 — Table Definition of Apron Features

When loading the data, the `airport` column was set to the filename of the AIXM file and the `gml_id` values were prefixed with the airport code to ensure their uniqueness. In a post-processing step some data inconsistencies in the source data were removed, in particular invalid geometries.

4.1.5. NOTAM Data Retrieval

The NOTAM API delivers a select set of the Digital NOTAM (DNOTAM) Event information, together with its geometry to indicate where the event is taking place. This section documents how the DNOTAM data is retrieved from SCDS, how it is filtered and enriched, and how a default geometry is computed for each NOTAM, before the resulting NOTAM data is finally stored in a PostgreSQL database. The [img_d104_retrieval] illustrates the NOTAM retrieval workflow.

4.1.5.1. Accessing NOTAM Data From SCDS

In order to access NOTAM data from the SWIM Cloud Distribution Service (SCDS), an SCDS account was created, followed by a subscription to the AIM FNS product. The subscription targeted all data items provided for that product; i.e. the subscription filter options for source type, NOTAM status / function / keyword, and airspace usage were not used.

SCDS delivers new data using a Java Messaging Service (JMS). JMS supports the *publish-subscribe messaging pattern*. The SCDS, acting as the data provider, publishes data on one or more JMS topics / message queues, to which subscribers connect in order to receive new messages.

In TB-17, interactive instruments developed a component that attaches itself to such a data feed. The component was written in Java, and leverages the SWIM feed handler project (open source, Apache License 2.0) to establish the connection and to receive NOTAM messages from SCDS.

4.1.5.2. Filtering and Enriching NOTAM Data

The interactive instruments NOTAM SWIM feed handler parses the following information from NOTAM messages received via JMS, including the JMS message headers:

- NOTAM keyword — Keyword associated with the NOTAM
- NOTAM function — Function of the NOTAM (New, Replacement, Cancelled)
- Valid time (begin)
- Valid time (end)
- Text — NOTAM condition text
- Q Code — Q Code value for the NOTAM.
- Year — NOTAM year values per Annex-15 of the International Civil Aviation Organization (ICAO) Convention on International Civil Aviation.
- Number — NOTAM number value per Annex-15 of the ICAO Convention on International Civil Aviation.
- Scenario — Identifier of the event scenario used for digital encoding. The mapping can be found in the Event Scenario documents.
- Flight Information Region — Flight Information Region (FIR) that is impacted by the NOTAM.
- Location designator — NOTAM location designator of the affected airport/ heliport or facility.
- ICAO location designator — ICAO location designator, if published.
- Series — NOTAM series value per Annex-15 of the ICAO Convention on International Civil Aviation.
- Type — NOTAM type value per Annex-15 of the ICAO Convention on International Civil Aviation. Accepted values are: New ('N'), Replace ('R'), Cancel ('C').
- Issued — Issue date/time of the NOTAM.
- Traffic — NOTAM traffic value per Annex-15 of the ICAO Convention on International Civil Aviation.
- Purpose — NOTAM purpose value per Annex-15 of the ICAO Convention on International Civil Aviation.
- Scope — NOTAM scope value per Annex-15 of the ICAO Convention on International Civil Aviation.

- Minimum flight level — NOTAM minimum flight level value per Annex-15 of the ICAO Convention on International Civil Aviation.
- Maximum flight level — NOTAM maximum flight level value per Annex-15 of the ICAO Convention on International Civil Aviation.
- Coordinates
- Radius
- Schedule — Contains a schedule of activity/outage if the hours of effect are less than 24 hours a day.
- Lower limit — Specifies the lower height restriction of the NOTAM.
- Upper limit — Specifies the upper height restriction of the NOTAM.
- Geometry — Default geometry determined for the NOTAM

The last item, the default geometry, is exceptional because it is not parsed directly from the Digital NOTAM Event, but computed on-the-fly — as outlined below.

4.1.5.3. Determination of a Default Geometry for a NOTAM

Digital NOTAMs typically do not have a default geometry attached to them. However, provision of such a geometry is highly beneficial for NOTAM consumers (users and systems alike). This is because the default geometry would support a common means to search for and filter NOTAMs relevant to a given use case. In addition, a GeoJSON- which is highly popular amongst web application developers — based NOTAM encoding requires a default geometry in a GeoJSON encoded feature. Therefore in TB-17, interactive instruments developed an approach for determining a default geometry for a NOTAM event.

According to the Digital NOTAM (DNOTAM) Event Specification (version 1, chapter 5.3), the geographical data included in NOTAM messages is less mathematical and more descriptive. Even though the DNOTAM Event Specification contains rules for converting GML encoded geometries into NOTAM text, the NOTAMs received from SCDS rarely seem to contain such information. The correct way to compute a default geometry for each NOTAM would be to define production rules for each NOTAM scenario supported by AIM FNS. Since such rules could not be found for use in TB-17, a pragmatic approach for computing default geometries for NOTAMs was taken.

NOTE 1: On https://notams.aim.faa.gov — section Documentation — a number of documents for FNS NOTAM scenarios can be downloaded. These documents, even though they were issued in December 2010, are still referenced in recent publications — such as the SWIM Federal Notice to Airmen (NOTAM) System (FNS) NOTAM Distribution Service (NDS) Publish/Subscribe Operational Context Document (from Feb 28, 2019). However, it is unclear how the scenario identifier from digital NOTAM events can be matched against the relevant scenarios. Furthermore, rules for computing a default geometry for the scenarios could not be found.

The approach to determine a default geometry depends on whether the NOTAM message is airspace-related or not. A NOTAM message is airspace-related if the message header 'us_gov_dot_faa_aim_fns_nds_NOTAMKeyword' has value 'AIRSPACE'.

Non-airspace-related NOTAM message:

- NOTAM messages delivered by SCDS often (but not always) contain time slices — mostly snapshots, but also temp deltas — of aeronautical features affected by the event. Note that the snapshots received from SCDS typically are incomplete. In other words, values are only given for a select subset of the feature properties — even though, according to the AIXM Temporality Model, a snapshot should contain values for all feature properties. In any case, some of these time slices contain geometry data. The data harvester implemented for TB-17 gathers all (elevated) points, curves, and surfaces contained in the aeronautical features that accompany (i.e. they are in the same message as) the event feature. The geometries of highest dimension are selected and, if there are multiple geometries, merged (creating a union geometry). If, for example, the geometries gathered from the aeronautical features contained points as well as surfaces, then only the surfaces would be selected.
- If the NOTAM message does not contain such geometries, then the ICAO location of the event is inspected. The ICAO location is part of an Event extension (in namespace http://www.aixm.aero/schema/5.1/extensions/FAA/FNSE). It is not provided for all events, but if it has a value, then an attempt is made to match it against the 'locationIndicatorICAO' property of AirportHeliport features contained in the 28 day NASR subscription airport baseline data. If a match exists, then the Airport Reference Point (ARP) is used as default geometry for the event.
- If no match can be found via the ICAO location either, the NOTAM message is dismissed by the harvester.

Airspace-related NOTAM message:

- For airspace-related NOTAMs, the NOTAM text was inspected first. If the text matched the regular expression `^AIRSPACE SEE (\\w{3}) (\\d+/\\d+) .*$` (with dot also matching newlines) then the second group within the expression identifies the NOTAM number. This number may match one of the Temporary Flight Restrictions (TFR) published by FAA not via the SCDS subscriptions, but only on the TFR website as HTML and not in a machine processable form. Such TFRs may be accompanied by a Shapefile whose feature(s) provide a geometry (collection) that is useful as the geometry of the NOTAM. If a match was found, and the TFR did have a Shapefile, the harvester used its geometry.
    - As an example, the following text matches the regular expression: 'AIRSPACE SEE FDC 1/5005 ZLC 91.137 HAZARD'. '1/5005' would be parsed as NOTAM number and the Shapefile describing the extent of the temporary flight restriction would be at `https://tfr.faa.gov/save_pages/1_5005.shp.zip`.

- If no geometry could be determined via the TFRs, then the airspace designator ('ZLC' in the example NOTAM) — which appears to be encoded in the event location — was matched against the IDENT field of airspace features contained in the Shapefiles that are part of the 28 day NASR subscription files and that was published via the Airspace API. If matches were found, the default geometry was set to the union of the shapes of the matching airspace features stored in the Shapefile.
- Otherwise, the NOTAM message was dismissed by the harvester.

The pragmatic approach described above was taken for TB-17 in order to have a common set of relatively simple rules for computing default geometries for NOTAM events. For a production environment, rules for computing a default event geometry would need to be defined for each NOTAM scenario — and implemented as such. Figure 9 summarizes this process.

NOTE 2: The 28 day NASR subscription files can be downloaded from the FAA website. As the name suggests, these files have a time period of 28 days in which they are effective.

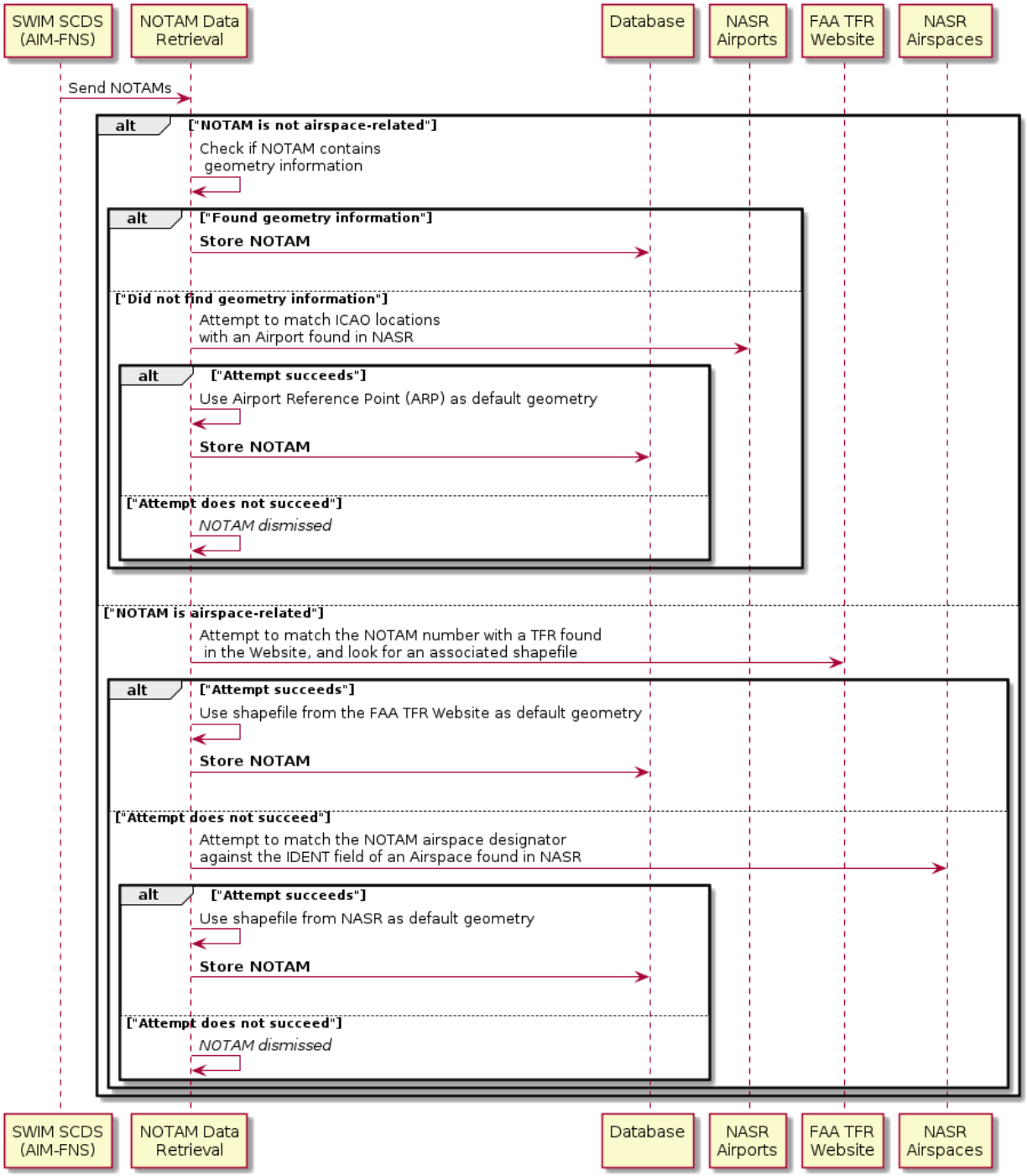


Figure 9 — Process for Determining Default Geometry

4.2. APIs

4.2.1. Overview

Four APIs have been set up using:

- Class B/C/D/E Airspaces from the National Airspace System Resource (NASR) Subscription,
- AIXM 5.1 airport data provided by Hexagon,
- NOTAMs received from the AIM-FNS feed from SCDS.

The APIs implement Parts 1, 2 and 3 of OGC API Features as well as drafts of OGC API Tiles, OGC API Styles and other draft extensions to OGC API Features.

The airport data is provided as two APIs:

- Organized by feature type. That is, each AIXM feature type is a feature collection that contains the features for all airports.
- Organized by airport. That is, each airport is a feature collection that contains the features of all feature Types for the airport.

The feature data is provided in two JSON encodings (GeoJSON and JSON-FG) and HTML, based on the AIXM application schema.

The data is provided in the following coordinate reference systems (CRS):

- WGS 84 longitude-latitude axis aligned (`http://www.opengis.net/def/crs/OGC/1.3/CRS84`)
- WGS 84 latitude-longitude axis aligned (`http://www.opengis.net/def/crs/EPSG/0/4326`)
- Web Mercator (`http://www.opengis.net/def/crs/EPSG/0/3857`)
- NAD83 (`http://www.opengis.net/def/crs/EPSG/0/4269`)

Only a few CRSs were configured since no client has requested support for additional systems. If needed, additional CRSs could be added.

4.2.2. API Design Considerations and Decisions

The data shared via the APIs are features that are specified in the Aeronautical Information Exchange Model (AIXM).

The APIs target web developers, but also aim to be usable by end-users familiar with aviation concepts in a browser.

Based on this, the following design decisions were made for all D104 APIs:

- There will not be a single Aviation API, but APIs for specific patterns of use. The APIs will re-use existing, standardized API building blocks mainly from IETF and the OGC API standards, where possible.
- The four Aeronautical Data APIs of D104 share base data for airspaces and airports as well as information about events with a temporal effect.
- Since the data, which is shared via the APIs, are features in the sense of the General Feature Model of OGC and ISO/TC 211, the APIs build on the standard API building blocks for feature data. In particular, they conform to OGC API — Features — Part 1: Core. An immediate benefit of this approach is that the APIs can be used with generic OGC API clients like QGIS or ArcGIS as well as in web browsers.
- To simplify the use of the data that is fetched by clients, every feature represents the state of the real-world object during a single time slice. In other words, an AIXM feature that consists of multiple time slices would be published via the API as multiple features, each with the temporal validity of the time slice.
- The features are provided in two JSON encodings: GeoJSON and JSON-FG.
- The AIXM feature type is encoded in each feature in a top-level JSON member with the key “featureType” and the AIXM feature type name with a “aixm:” prefix, e.g. `"featureType": "aixm:AirportHeliport"`.
- The feature properties use the unqualified name of the AIXM property. Values are simplified / flattened, where possible, based on the information contained in the source data.
- In addition to providing the features in JSON, the data are also provided as HTML. The APIs translate the AIXM based data to a HTML representation that is human-friendly. Instead of the property names, the titles (and descriptions) from the AIXM documentation are provided. Coded values are translated to their human-friendly labels.
- To support easy-to-use filtering of data, the AIXM features can be filtered using query parameters for key attributes — in addition to the spatial and temporal filtering using the standard `bbox` and `datetime` query parameters.

- To support retrieving the feature geometries in other coordinate reference systems beside WGS 84, all APIs conform to OGC API — Features — Part 2: Coordinate Reference Systems by Reference.
- To support more advanced filtering of features, all APIs conform to the current draft of OGC API — Features — Part 3: Filtering and the Common Query Language (CQL).
- To support large filter expressions, the APIs also support URL-encoded POST requests to the Features resources.
- To simplify filtering features that are currently valid, an additional token "now" is supported in the `datetime` parameter ("now" can also be the start or end value in an interval). In CQL filters a function `now()` can be used.
- To support clients that need a schema of the available data, the JSON schema of the features is available based on the current "Schema" proposal.
- To support clients that only need a subset of the feature properties, the content can be tailored based on the current "Property Selection" proposal.
- To support clients that want to render feature data on a map, the feature geometries can be simplified based on the current "Geometry Simplification" proposal.
- To simplify viewing the data in a web map — including in the HTML representation in the web browser, some of the APIs also conform to the drafts of OGC API Tiles and OGC API Styles. Where supported, the feature data is also provided as vector tiles in the Mapbox Vector Tiles format and styles are available in the Mapbox Style format. The data in the vector tiles is restricted to data this is valid at the time of the request.
- OpenAPI 3.0 is used to define / specify / document the API.
- Cross-Origin requests are supported.
- HTTP caching and conditional requests are supported to leverage web caches.

Just like the OGC API standards, the four APIs making up this Façade Service have been designed to support two different approaches for how clients will use the API:

In the first approach, clients are implemented with knowledge about the API specification and its resource types. The clients navigate the resources based on this knowledge and based on the responses provided by the API. The API definition may be used to determine details, such as filter parameters, but this may not be necessary depending on the needs of the client. These are clients that are in general able to use multiple APIs as long as they implement the same API building blocks.

The second approach targets developers that are not familiar with OGC API standards but want to interact with spatial data provided by an API that happens to implement an OGC API standard. In this case the developer will study and use the OpenAPI definition to understand the API and implement the code to interact with the API. This assumes familiarity with OpenAPI

and the related tooling. As such studying the OGC API standards documents should not be necessary.

4.2.3. General Rules for all APIs

The following sub-sections provide an overview of the available resources in the API. All paths are relative to the base URI `https://t17.ldproxy.net/{apiId}` where `{apiId}` is `airspaces`, `airports`, `airports2` or `fns`.

4.2.3.1. Read-only

The APIs only provide read-access.

With the exception to the resources at `/collections/{collectionId}/items` only the HTTP GET method is supported. That resource also supports the HTTP method POST with the form media type `application/x-www-form-urlencoded` in addition to GET in order to support queries with large geometries due to length restrictions for URLs.

4.2.3.2. Supported Encodings

Resources are in general available as HTML and JSON:

- To fetch HTML, use HTTP content negotiation (e.g. set the Accept header to `Accept: text/html`) or use a query parameter `f=html`.
- To fetch JSON/GeoJSON, use HTTP content negotiation (e.g. set the Accept header to `Accept: application/json, application/geo+json`) or use a query parameter `f=json`.
- To fetch JSON/JSON-FG, use HTTP content negotiation (e.g. set the Accept header to `Accept: application/json, application/vnd.ogc.fg+json`) or use a query parameter `f=jsonfg`.

The metadata about the data that is used in the API configuration and to provide a human readable HTML representation has been derived from AIM documentation, the AIS Open Data Dictionary and the AIXM documentation.

4.2.3.3. General Information

- `/` — the Landing Page, contains links to sub-resources in the API
- `/conformance` — the Conformance Declaration, lists the URIs of the OGC API conformance classes implemented by the API; this resource is only relevant for clients that implement OGC API standards.

These resources are specified by OGC API Common / OGC API Features.

4.2.3.4. The Data That is Available

- `/collections` — the Feature Collections available in this API
- `/collections/{collectionId}` — the Feature Collection with id `collectionId`
- `/collections/{collectionId}/schema` — the JSON schema describing the GeoJSON features that are returned for the Feature Collection with id `collectionId`
- `/collections/{collectionId}/queryables` — the JSON schema describing the properties that can be used to filter the features in the Feature Collection with id `collectionId`
- `/collections/{collectionId}/context` — the JSON-LD context referenced from the GeoJSON features that are returned for the Feature Collection with id `collectionId`

These resources are specified by OGC API Features or are ldproxy extensions.

The JSON-LD contexts are the result of early experiments, but are no longer used. The main reasons are that no clients in the testbed needed this information, and no vocabulary or ontology exists for AIXM that could be linked to.

4.2.3.5. Accessing Features

- `/collections/{collectionId}/items` — access to features in the Feature Collection with id `collectionId`. This request supports a range of query parameters to filter the features or modify the feature representation that is returned (see the example requests below or consult the API definition).
- `/collections/{collectionId}/items/{featureId}` — access the Feature with identifier `featureId` in the Feature Collection with id `collectionId`.

These resources are specified by OGC API — Features, sub-clauses 7.15 and 7.16.

4.2.3.6. Accessing Vector Tiles

- `/tiles/WebMercatorQuad/{tileMatrix}/{tileRow}/{tileCol}` — access to vector tiles with features organized in layers (one for each airspace class). Only the standard tiling scheme that is used by Google Maps is supported.
- `/collections/{collectionId}/tiles/WebMercatorQuad/{tileMatrix}/{tileRow}/{tileCol}` — access to vector tiles with features with a single

layer with features from the Feature Collection with id `collectionId`; only the standard tiling scheme that is used by Google Maps is supported.

There are more resources, but these are the main ones to use in a typical mapping client.

These resources are specified in the draft OGC API — Tiles specification.

4.2.3.7. Style

- `/styles/{styleId}` — access to styles that may have been configured for an API, available in the Mapbox Style encoding. The HTML representation renders a map using the style and the vector tiles.

There are more resources, but this is the main one to use in a typical mapping client.

These resources are specified in the draft OGC API — Styles specification.

4.2.4. FAA's NOTAM API

During the execution of the TB-17, the FAA announced their own NOTAM API development in a SWIFT meeting on May 27, 2021. The following section compares the similarities and differences between the API developed by FAA and the API developed in TB-17, based on the information in the presentation given at the meeting (slides 22 to 36).

4.2.4.1. Similarities

Both APIs

- aim at "a simplified machine-to-machine interface to help improve access to NOTAM data;"
- are Web APIs specified and documented using OpenAPI 3.0;
- target developers;
- enable direct NOTAM queries with filtering without the need to locally store all NOTAMs;
- use SCDS AIM-FNS as the source for NOTAMs;
- integrate Temporary Flight Restriction (TFR) geometries;
- publish the NOTAMs as GeoJSON with a very similar schema;

4.2.4.2. Differences

The FAA NOTAM API

- supports authentication through API keys (not relevant for TB-17);
- supports AIDAP and AIXM 5.1 output formats in addition to GeoJSON (not relevant for Testbed-17);
- integrates also Special Activity Airspace (SAA) geometries (no publicly available machine readable source for the SAA geometries could be determined); and
- provides a JavaScript SDK.

The TB-17 NOTAM API

- supports HTML for all resources which enables direct use in a browser by anyone;
- supports all filtering capabilities of the FAA API plus much richer capabilities through the support for CQL2 and a larger set of queryable properties (i.e. properties that can be used to filter the features in a Feature Collection);
- conforms to approved and draft OGC API standards;
- can be used with a growing number of tools, libraries and SDKs that support OGC API standards;
- improved visualization in map clients through the support for vector tiles; and
- publishes the NOTAMs also as JSON-FG encoded documents.

4.3. Challenges and Lessons Learned

Challenges retrieving input data:

- Finding aeronautical datasets on the FAA websites was challenging.
- The datasets discovered did not have information about the license under which they can be used.
- Some data still uses the old AIXM 5.0 standard, some data uses AIXM 5.1, but deviating from the guidance in the AIXM or Digital NOTAM specifications. interactive instruments was not successful in determining some of the FAA conventions (e.g. the NOTAM scenarios and their identifiers). The only reliable approach was to try and analyze the data and reverse engineer the content to determine how the data could be processed.

Challenges executing internal processes:

- See Clause 4.1.5.3.

Challenges providing output data:

- There was one issue that was raised by the map client (D108) with respect to the feature encoding. The client needs to know the feature type in order to select the appropriate portrayal style for the feature. Since GeoJSON does not have a feature type concept, it was decided to add a new JSON member "featureType" to indicate the AIXM feature type. This member is borrowed from the JSON-FG work and was included in the GeoJSON encoding to support the map client.
- An advantage of the JSON-FG encoding is the "when" member as it — together with the `datetime` query parameter — supports that clients provide a time slider without the need to understand the feature schema.
- JSON-FG also is clearer when geometries are not encoded in WGS 84 lon/lat due to the explicit support for other CRSs. However, the scenarios used in the testbed did not really have a need to use other CRSs.
- Due to the lack of a referenceable AIXM vocabulary or ontology, it was not possible to enrich the JSON data with semantic annotations using JSON-LD contexts. At the same time, none of the clients was expecting such information, so this was not really an issue in this testbed.

4.3.1. Contrast With Testbed-16 Aviation Task Challenges

- **What is a feature in SWIM for the OGC API — Features service?**: The issue still exists. The NOTAMs distributed via SCDS are not full Digital NOTAMs and much of the information would have to be parsed from the NOTAM text and referenced to a dataset with all referenced aeronautical features (which was not available to us, except for certain feature types like airspaces). The participant therefore decided to restrict the API to NOTAMs as features.
- **Unique identification of a feature**: The issue still exists. The AIXM datasets that were used do not have any persistent identifiers. A result is that the identifiers are basically assigned by the API. This was not an issue for the use cases in this testbed, but these would be an issue for a production API.
- **Spatial extent in a feature collection**: This issue was addressed. See Clause 4.1.3.1.
- **Temporal extent in a feature collection**: This issue was addressed. See Clause 4.1.3.1.
- **Spatial information in a feature**: This was an issue in this testbed. See Clause 4.1.5.3.

- **Limited semantic descriptions of feature collections and features**: See the comments on JSON-LD above.
- **Large data volume**: Not an issue for D104. The NOTAM volume has not been an issue for the duration of this testbed. Purging NOTAMs that have not been active for some time should not be a problem for most use cases.
- **Automatic code generation is not perfect**: No experiments with automatic code generation.

5

SWIM FLIGHT POSITION FAÇADE SERVICE

5 SWIM FLIGHT POSITION FAÇADE SERVICE

The SWIM Flight Position Façade Service provides flight position data from the Federal Aviation Administration's (FAA) System Wide Information Management System (SWIM). D105 exposes separately flight data from three different SWIM sources:

- The SWIM Terminal Data Distribution System (STDDS),
- The Traffic Flow Management System (TFMS), and
- The SWIM Flight Data Publication Service (SFDPS).

These three sources of information provide the complete gate-to-gate story of a flight.

5.1. Status Quo

SWIM delivers data for the three flight data feeds leveraged for D105 via Solace Java Message Service (JMS) messaging. JMS requires end users to read JMS Description Documents across the services (currently on NSRR), subscribe to a service, consume the data from the service, transform the data into a common language, and then map that data into a database.

Across that process of consuming the data, there are a few distinct challenges:

- Firstly, the data coming from these feeds is not only limited to flight position data. Therefore, the developer needs to decide early on in the process what data will be needed for their specific use case. Usually the answer is "give me everything" but there are challenges with that approach as typically the end user has a specific use case in mind.
- The data provided by SWIM is not necessarily in a consistent format. TFMS and STDDS data is provided in the FIXM format whereas SFDPS data is provided in a "custom" data format. These challenges, including the volume of data, make the data difficult to access and make sense of for the common user.

5.2. Functional Overview

This component adds value for an end user because it provides data from the three SWIM data feeds via a consistent interface, with consistent documentation, and in a consistent format. By leveraging OGC's APIs, a developer can expect to have a consistent experience with the data from these three feeds, which is not the case when accessing the data via SWIM JMS.

Additionally, the façade solves data collection and storage tasks for the user. A user does not have to ingest or maintain a big database to store it all.

As seen on Figure 10, the façade is made of three sets of connectors, parsers, and servers. Each one of the three connectors is designed to communicate with a different SWIM flight feed: SFDPS, TFMS, and STDDS. SFDPS feeds custom XML to its connector, while both TFMS and STDDS feed FIXM data to its respective connectors. Each parser is designed to interpret the data coming from each one of its corresponding flight feeds, and relay flight data encoded in JSON format to its respective Post GIS Server. The API of the façade is designed to communicate with each one of these three GIS servers to retrieve information when requested. Figure 11 shows the end-to-end data flow of D105.

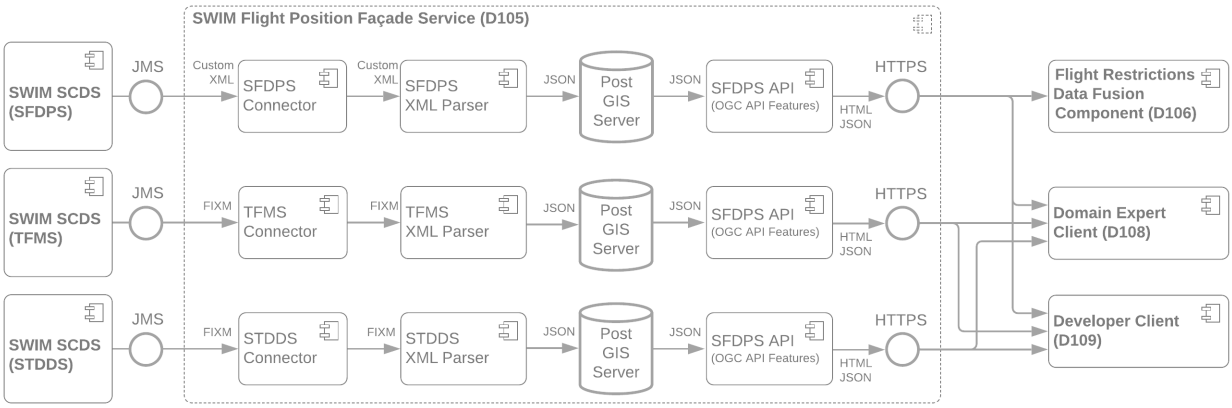


Figure 10 — Structure of the SWIM Flight Position Façade Service

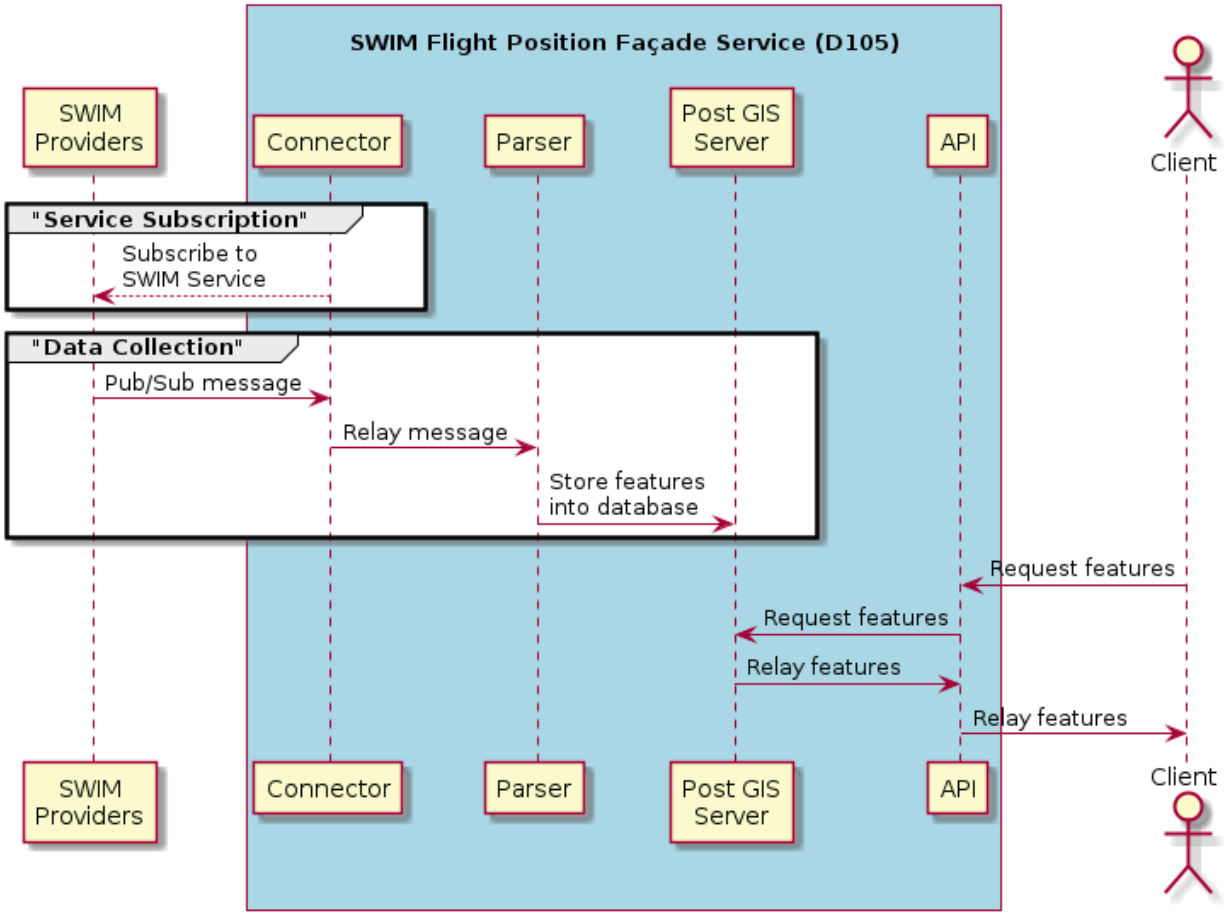


Figure 11 — Data Flow of the SWIM Flight Position Façade Service

5.2.1. Available APIs

D105 provides actual flight position data from TFMS, STDDS, and SFDPS. STDDS provides flight positions in and around airports versus TFMS and SFDPS focus more on en route flight positions. The API calls are defined using OpenAPI 3.0.1 and are structured similarly for each feed providing a standardized way for a developer to access the data from these three disparate feeds. The API calls available for each data feed include:

- Get a unique route (user inputs acid, origin airport, destination airport, and time).
- Get routes of flights using an origin airport (user inputs origin airport, start time, and end time).
- Get route of flight using a destination airport (user inputs destination airport, start time, and end time).
- Get route for a specific GUFI (user inputs GUFI).

The returns are structured in JSON. In cases where a single flight is returned (e.g., Get route for the specific GUFI), a single feature is returned whereas for cases where multiple flights are

returned, then a `featureCollection` is returned. However, it should be noted that for all the services, a flight maps to a single feature. Each feature includes the following:

- `type`
- `geometry`
- `gufi`
- `acid`
- `aircraft_address`
- `aircraft_registration`
- `arrival_aerodrome_icao_name`
- `departure_aerodrome_icao_name`
- `departure_time`
- `arrival_time`

Resulting feature collections are filtered only by the parameters defined for each function. In the case of functions that return `FeatureCollection`, these include a limit parameter. No other general use filtering strategy is provided at this time.

5.3. Challenges and Lessons Learned

- **Transitioning from pub/sub to request/reply environments**: The biggest challenge associated with retrieving the input data was transitioning from a pub/sub environment to a request/reply environment. The SWIM flight data feeds are bulky. Therefore, if simplifying assumptions are not made, a database designed for each feed will quickly reach capacity. Therefore, the APIs were designed to only support the data needs for the envisioned fusion services. The volume of data made some of the API calls slow. Therefore, steps were made to improve performance (e.g., limiting the number of feature returns).
- **Performance limitations due to data amount**: As noted before, there is a lot of data included in these feeds. As such the data returned was specific to the need for actual flight position data. Provided the volume of flight data in each return (flight position updates every 10-15 seconds), database performance was a challenge and would pose a significant challenge if this capability is deployed as a real-time service.
- **Data Mapping**: Data needs to be mapped into features to make them available through an implementation of OGC API – Features. This mapping is not a simple one-to-one match because a message may contain several features with different

spatial and temporal ranges. The identification of mappable features from SWIM services needs to be considered carefully case by case.

6

FAA FLIGHT PLANS FAÇADE SERVICE

6 FAA FLIGHT PLANS FAÇADE SERVICE

The FAA Flight Plans Façade Service, developed by Skymantics, was a component designed to fetch flight data from SWIM Flight Data Publication Services (SFDPS) and relay it through an OGC API — Features endpoint. Information from this component was retrieved by both the International Flight Data Fusion Component, and the Flight Restrictions Data Fusion Component.

6.1. Functional Overview

The Façade Service is made up of several interconnected elements, as seen in Figure 12. A JMS Client is designed to retrieve messages from the SWIM SCDS. The flight importer is designed to combine the XML coming from the JMS client with geometry information. Two databases, one for EUROCONTROL data and another one for FAA data, keep information of airports, waypoints, control centers, among other data. A third database stores flight data and geometries ready to be served on request by the API. Finally, the component includes an API based on OGC API — Features.

The process flow is the following:

- New message published in the SWIM subscription.
- Message is received by the client.
- Message information is decoded by flight importer. Geometry of the flight route is generated based on the flight route string and FAA's aeronautical information pertaining to waypoints and airport locations. EUROCONTROL's aeronautical information is kept as backup in case some locations are not found. Links to features expanding information (airports, ARTCCs) are also included. FAA and EUROCONTROL's aeronautical information were previously extracted and stored in a PostgreSQL database.
- The resulting flight information is stored in a PostgreSQL database.
- Flight data in the database is published using an implementation of OGC API — Features in three different formats (HTML, GeoJSON and JSON-FG).
- FAA's aeronautical information is also published using this same API.

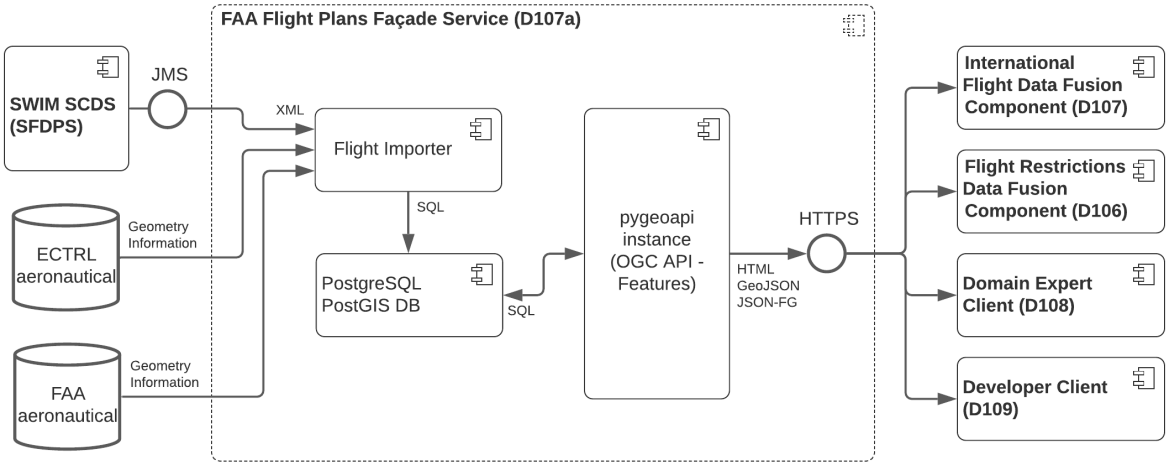


Figure 12 — Structure of the FAA Flight Plans Façade Service

A Java-based client listens to the subscription. The raw data is transformed before being served through the API, although the original basic FIXM structure is kept reasonably similar to the original. It is still possible to review the original FIXM documentation and understand the structure, values and semantics.

Two attributes/properties were modified in order to simplify the structure:

- altitude is a JSON object, combining original objects assignedAltitude and requestedAltitude for simplicity
- airspeed is a JSON object, combining original objects assignedAirSpeed and requestedAirSpeed for simplicity

Moreover, three attributes/properties were enriched to provide links to external features providing additional information:

- arrival (JSON object with embedded schema:Airport items that link to Airports collections)
- departure (JSON object with embedded schema:Airport items that link to Airports collections)
- centre (JSON object with embedded nasr:ARTCC item that link to ARTCCs collection)

GeoJSON format uses geometries as an essential component of the encoding. These geometries store the coordinates of a feature. For example, a specific coordinate for a waypoint, a list of coordinates for a flight trajectory, or a matrix of coordinates indicating a volume in the space affected by a restriction. They are very important as they allow clients to display visually a feature on a map or on a 3D model. They also allow to manipulate features and find geographical commonalities, such as a flight trajectory intersecting a restriction volume, becoming essential for service fusion.

NMB2B or SFDPS do not provide geometries for their features, but references to airports, waypoints and route segments. These elements are stored in separate airspace datasets with detailed description of their attributes, including their geographical properties. In D107a and D107b, coordinates for airports and waypoints were first extracted from these airspace datasets, so that route strings in flight plans could be interpreted and translated to a list of coordinates.

Finally, the data is transformed from XML to three different formats (HTML, GeoJSON and JSON-FG)

A Postgres database stores the latest status for a flight (GUFI). If a SWIM message arrives with a GUFI that already exists, the data for that flight is updated.

6.1.1. API Overview

This API was built following the Standard OGC API — Features and adapted to publish JSON-FG.

In this case, flight plans fit well in the definition of feature (according to ISO-19101, a feature is defined as an: “abstraction of real world phenomena”), so OGC API Features seems the natural API to publish flight data. As flight plans always have a temporal interval (the planned or actual departure and arrival times), the best format to encode them is probably JSON-FG, although it is important to keep GeoJSON format due to its wide adoption and ecosystem of ready-to-use tools.

The API supports **filtering** using the following parameters:

- limit: Will return the next elements until the amount specified in the limit.
- bbox: Will return all flights that cross the bounding box.
- datetime: Only available if format requested is `jsonfg`.
  - specific datetime: Will return all flights that are operating at that moment.
  - datetime interval: Will return all flights that are operating at any moment during that interval.
- queryables: The list of available queryables is here: https://aviationapi.skymantics.com/faa/collections/faa_flight_plans/queryables.

The API also provides **semantic information**. To start with, the HTML and GeoJSON formats provide one property named “feattype” that indicates the type of feature. In JSON-FG format the top-level member featureType is the one indicating the type of feature.

The content of the featureType (or the feattype property) for flight plans is `sfdps:FlightPlan`, which follows the JSON-LD standard. This serves as an anchor to define contexts and provide links to dictionaries providing further information of the featureType. In this case, the context is sfdps and could provide the link to a dictionary containing all the information on FAA’s SFDPS.

Moreover, JSON-FG provides additional links to JSON schemas describing the structure of the collections and features data. Several links can be provided for the same item, for example a link to the schema of a GeoJSON feature, a link to the schema of a JSON-FG feature and a link to the schema of a FAA SFDPS flight plan, being all simultaneously valid (because these features follow those three schemas simultaneously).

Apart from that, nested properties in features keep their original FIXM types (such as `ns5:NasRouteType`, `ns2:FixPointType` or `ns5:NasAircraftType`). In addition, other nested properties that provide a link to external features stored in the aeronautical databases have their own type definition, referring to the linked type, such as arrival or departure airports and alternates (type defined as `schema:Airport`) or control center (type defined as `nasr:ARTCC`).

Finally, the API also provides links to external features, in order to complete information such as airports and control center.

6.2. Challenges and Lessons Learned

- **Sheer number of messages**: Even though the subscription was limited to just three types of messages and six airports, the quantity of messages was overwhelming (about 40,000 messages/day, about 80% of which were updates to existing flight plans, over 300Mb of data downloaded each day). When a flight was updated, all the flight information was included in the message, not only the updated data, thereby unnecessarily increasing the size.
- **FAA aeronautical data format**: FAA Aeronautical data was relatively easy to find, but the encoding format was cumbersome to process. Besides, only US data is included, so the data needed to be complemented with EUROCONTROL's aeronautical data in order to build the geometries of the routes for the international flights.
- **Building the geometries of the routes**: Routes are encoded in compressed strings that are hard to decode (example: `KIAD./.DYR034086..FEWWW.SEEVR4.KDFW/0021`). It requires understanding the format (`waypoint.route.waipoint.route.waypoint` and so on), interpreting special characters, numbers and having access to all the waypoints that are encoded in the route string. Then, there is also the need to add logic, as some international waypoints are duplicated (that is, two waypoints using the same id but in different coordinates) and you need to select the most probable one.
- **Missing Data**: Additionally, specially from inbound flights to the US, the FAA misses crucial information, such as departure time and arrival time, or all the route information between the departure airport and the beginning of the portion of the flight occurring in FAA airspace.

6.2.1. Contrast With Testbed-16 Aviation Task Challenges

- **Spatial information in a feature**: Flight plans do not explicitly include spatial information but must be inferred from the route string, which is hard to interpret and is sometimes ambiguous.
- **Limited semantic descriptions of feature collections and features**: In JSON-FG this issue is solved with the `featureType` member, the JSON-LD context, and the links to JSON schemas. However, these dictionaries and schemas need to be implemented and available for linking. https://semantics.aero/ would be a good place to start. This is a pretty similar conclusion as in TB-16
- **Large data volume**: This issue was resolved by cleaning the database of messages older than 7 days. But it is an issue as it limits the scalability of the service. For example, if updates were limited to only the data that has changed, the data transfer will likely drop to less than half of current level.

7

EUROCONTROL FLIGHT PLANS FAÇADE SERVICE

7 EUROCONTROL FLIGHT PLANS FAÇADE SERVICE

Developed by Skymantics, the EUROCONTROL Flight Plans Façade Service was a component designed to fetch flight data from EUROCONTROL NMB2B and relay the data through an OGC API — Features endpoint. Information from this component was retrieved by the International Flight Data Fusion Component.

7.1. Functional Overview

Depending on the use case, the NMB2B service offers two types of interfaces: a request/response and a sub/pub. For this task, as the FAA façade was built on top of a pub/sub service, the participants decided to build the EUROCONTROL façade on top of a request/response one, with the purpose of comparing building a façade on top of both types of interfaces.

As shown in Figure 13, the component is made up of several interconnected elements. The flight importer is designed to request flight plans from NMB2 and combine it with geometry information. Two databases, one for EUROCONTROL data and another one for FAA data, keeps information from airports, waypoints, control centers, and so on. A third database stores flight data and geometries ready to be served on request by the API. Finally, the component includes an API based on OGC API — Features.

The process flow is the following:

- The NMB2B service is a request/response type, so there is no client listening to a subscription. Instead, periodically a process triggers a flight importer script that requests flight plans, storing new ones and updates existing ones.
- Message information is decoded by the Flight Importer. Geometry of the flight route is generated based on the flight route string and FAA's aeronautical information pertaining to waypoints and airport locations. EUROCONTROL's aeronautical information is kept as a backup in case some locations are not found. Links to features expanding information (airports, ARTCCs) are also included. FAA and EUROCONTROL's aeronautical information were previously extracted and stored in a PostgreSQL database.
- The resulting flight information is stored in a PostgreSQL database.
- Flight data in the database is published using an implementation of OGC API — Features in three different formats (HTML, GeoJSON and JSON-FG).
- EUROCONTROL's aeronautical information is also published using this same API.

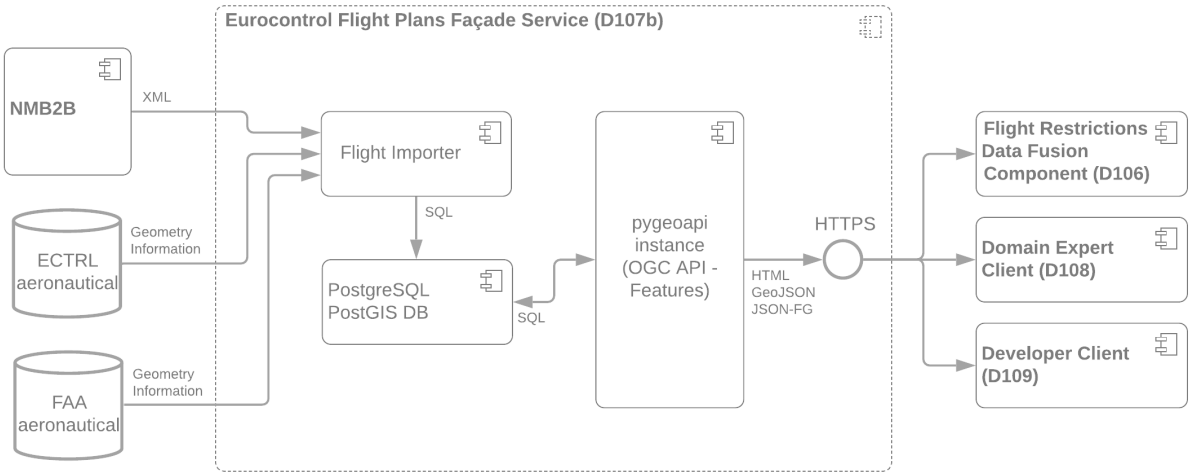


Figure 13 — Structure of the EUROCONTROL Flight Plans Façade Service

In order to reuse the original NMB2B Services documentation to a great extent, the data structure has been kept as close to its original XML implementation as possible.

The data structure is defined in the service request and can be tailored to the needs. This is a very convenient capability, as it supports adjusting the bandwidth and storage space to the real needs. For this case, all possible data is requested, meaning that messages from NM B2B service will be bulky.

Geometry in GeoJSON and JSON-FG formats is a `LineString` depicting the planned route. Although NM B2B provides a similar route string as in the case of D107a, there is a different element providing a more detailed trajectory (probably generated using an internal fusion service). This element is `ftfmpointprofile` and is fused with EUROCONTROL's Airports and FIXes collections (and also FAA's Airports and FIXes collections) to generate the coordinate list that will depict the route. In fact, the resulting 4D trajectories were so nicely generated that one sample was easily packed following the draft OGC Route Exchange Model specification and presented to the OGC Routes SWG with a pretty enthusiastic reception.

Similarly, as in D107a, the JSON-FG format includes `@context` information, linking to semantic descriptions of the `featureType`. The `when` attribute provides information on the time interval between departure and arrival, if that information is available.

The data is published in three different formats (HTML, GeoJSON and JSON-FG)

7.1.1. API Overview

- **Main endpoint (OGC API — Features)**: Data for this endpoint is periodically downloaded from **NM B2B PREOPS** service for all flights leaving or arriving to six airports (KJFK, KBOS, KPHL, KCLT, KIAD and KMEM), same as FAA façade.
    - **Flight plans**: https://aviationapi.skymantics.com/eurocontrol/collections/eu_flight_plans/

- **Additional endpoints (OGC API — Features)**: Data for these endpoints is static and has been extracted from EUROCONTROL NM Airspace Datasets.
    - **Airports**: https://aviationapi.skymantics.com/eurocontrol/collections/airports
    - **FIXes**: https://aviationapi.skymantics.com/eurocontrol/collections/fixes

7.1.1.1. Data Structure

Some of the most relevant data per flight are:

- **`flightid`**: Different from FAA's GUFI.
- **`aircraftid`**: Same as the one used in FAA façade, in flightidentification queryable.
- **`aerodromeofdeparture`**: A JSON object adapted to use similar structures as in the FAA façade, with embedded schema:Airport items that link to Airports collections.
- **`aerodromeofdestination`**: A JSON object adapted to use similar structures as in the FAA façade, with embedded schema:Airport items that link to Airports collections.
- **`estimatedoffblocktime`** / **`estimatedtimeofarrival`**: Properties used to estimate the time interval of the flight.
- **`ftfmpointprofile`**: A detailed route plan, with important similarities to the Route Exchange Model being defined in Routes SWG. According to EUROCONTROL's documentation:

    > FTFM (Filed Tactical Flight Model) point profile. The FTFM flight profile corresponds to the trafficType DEMAND. So in the operational dataset, it reflects the latest AO flight plan: i.e. the latest filed flight plan but updated (shifted) with the latest CDM related info and READY messages or amended by NM OPS room.

Each feature is published in GeoJSON and JSON-FG formats in order to compare these two encodings.

7.1.1.2. Process

Every hour a process is launched to request flight plans from NM B2B service. Allowed traffic bandwidth is limited, so this needs to be done with caution. New messages are parsed and entered in a PostgreSQL database (either as a new row or updating an existing one). The properties `aerodromeofdeparture` and `aerodromeofdestination` are parsed including a link to features in other collections and a context-referenced type.

The route is processed to generate the feature geometry by finding departure and arrival coordinates in the Airports collection and then either by searching the waypoint coordinates in the FIXes collection, or by calculating them if these are non-published geopoints. Geometries are stored as PostGIS geometries.

If an airport or waypoint was not found in the EUROCONTROL collections, it was searched for in the FAA collections.

Please note that the service is accessing a PREOPS version of NM B2B and some flight plans might not be inserted or might be updated late.

Data is published by an adapted pygeoapi instance, set up as an OGC API – Features endpoint.

7.1.1.3. Usage

The collection queryables can be requested. Several JSON object properties have been disabled as queryables (and thus cannot be found in the queryables list), but others have been adapted and will search in one of their subproperties. For example: properties `aerodromeofdeparture` and `aerodromeofdestination` will search in their respective airport ids.

Flight plans are dynamic and old plans (over one week old) are deleted from the database.

NOTE: When using JSON-FG format, it is possible to filter by datetime (and only when using JSON-FG format). This will query for flight plans that are occurring at a specific instant or that overlap total or partially with an interval.

Example requests from the API:

- List all flight plans departing Paris Charles de Gaulle Airport (LFPG): https://aviationapi.skymantics.com/eurocontrol/collections/eu_flight_plans/items?aerodromeofdeparture=LFPG
- List all flight plans from Paris Charles de Gaulle Airport (LFPG) to John F. Kennedy International Airport (KJFK): https://aviationapi.skymantics.com/eurocontrol/collections/eu_flight_plans/items?aerodromeofdeparture=LFPG&aerodromeofdestination=KJFK
- List all flight plans from Paris Charles de Gaulle Airport (LFPG) to John F. Kennedy International Airport (KJFK) that were flying anytime (UTC) during 2021 Sept

8 (note the usage of `f=jsonfg` parameter): https://aviationapi.skymantics.com/eurocontrol/collections/eu_flight_plans/items?f=jsonfg&aerodromeofdeparture=LFPG&aerodromeofdestination=KJFK&datetime=2021-09-08T00:00:00Z/2021-09-08T23:59:59Z

- List all flight plans from Paris Charles de Gaulle Airport (LFPG) to John F. Kennedy International Airport (KJFK) that were flying at 10 am (UTC) on 2021 Sept 8 (note the usage of `f=jsonfg` parameter): https://aviationapi.skymantics.com/eurocontrol/collections/eu_flight_plans/items?f=jsonfg&aerodromeofdeparture=LFPG&aerodromeofdestination=KJFK&datetime=2021-09-08T10:00:00Z
- List all flights operated by aircraft AFR014: https://aviationapi.skymantics.com/eurocontrol/collections/eu_flight_plans/items?aircraftid=AFR014

7.2. Challenges and Lessons Learned

- **Obsoleteness and Bandwidth**: The process was different than in D107a because of the different nature of the service. The number of properties per flight could be selected in the query, making the Façade Service both flexible and adaptable. However, the fact that no subscription informed of changes in the flight information meant that it was unknown whether the retrieved data was obsolete or not. So, flights had to be requested again to ensure eventual changes and new flights were registered. However, EUROCONTROL imposes limits to the maximum bandwidth that can be used, needing to modulate the pace of requests and finding a compromise between update frequency and bandwidth consumption.

  As a conclusion, even though the number of properties per flight could be selected and -in theory- the size of the messages were adapted to the needs, the Façade Service ended up using a large amount of bandwidth because NM B2B had to be queried frequently for updates.

- **Waypoints with same ID**: Although the interpretation of the trajectory was easier in this case than in D107A, there was still the problem of waypoints with the same id but in different locations.

- **Data not Found on both Datasets**: When implementing the fusion service, some international flights from or to Europe that were in FAA's façade could not be found in EUROCONTROL's, even if querying directly the NM B2B service. It was assumed that the reason for that is that the NM B2B service is a PREOPS version and might lack some data. However, the participants also found that some flights in EUROCONTROL's façade were not in FAA's, so the problem might be of lack of coordination among international agencies.

- **Advantages Found**:
  - Flight data is encoded in GeoJSON / JSON-FG formats with all the associated advantages of readability, adoption, geometry information and semantics. Geometry information is particularly remarkable as it allows for filtering based on bounding boxes or areas, allowing also for coordinated-based fusion services.
  - The HTML output format allows to visualize the flight plan trajectory at the server level.
  - NMB2B imposes several restrictions, such as the datetime interval can only be +/1 day of current date, or the bandwidth limitations, that would easily limit intensive TIEs. All these restrictions are overcome by implementing a façade.
  - The façade was deployed close to the fusion service, improving the performance of the fusion service.

7.2.1. Contrast With Testbed-16 Aviation Task Challenges

- **Spatial information in a feature**: This service offers a much more convenient way to assess the 4D flight trajectory in detail (in fact, the EUROCONTROL trajectory was encoded using the rules specified in the draft Route Exchange Model). The only remaining problem was the ambiguity of some waypoints that share the same ID but are located in very different coordinates.
- **Large data volume**: NM B2B supports requesting flight information when needed and to select the properties to be downloaded, so this issue is solved. However, a new issue appears if there are no sub/pub services providing timely info on updates and new flights. This issue forces periodic requests that require a high consumption of bandwidth. This can be solved by using both request/response and pub/sub services provided by NM B2B to complement each other.

8

FLIGHT RESTRICTIONS DATA FUSION COMPONENT

8 FLIGHT RESTRICTIONS DATA FUSION COMPONENT

The Flight Restrictions Data Fusion Component, built by Ng Aviation, enables users to retrieve the flight restrictions found in a specific planned or already-flown flight. To do this, it fuses flight paths with flight restrictions coming from different sources.

8.1. Functional Overview

The Fusion Service consists of a submodule named *Data Joiner* which does the actual data fusion, and an API built to interface clients. The Data Joiner processes requests, retrieves data from Flight Providers and Restrictions Providers, and relays fused data to the API. The API was built following OGC API — Features — Part 1: Core. This architecture is described in Figure 14.

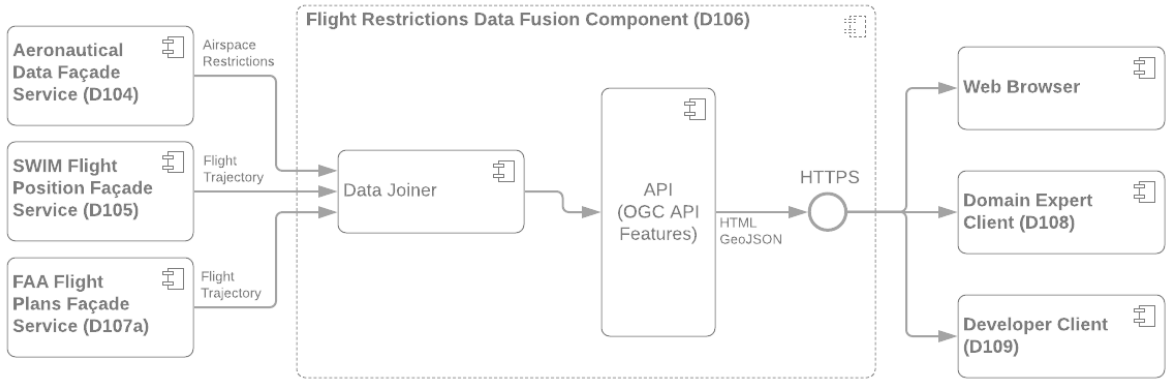


Figure 14 — Structure of the Flight Restrictions Data Fusion Component

The fusion service does not have a database for storing raw or fused data. It is neither possible nor effective to copy all the data from every data source used for the data fusion process. There are no pub/sub endpoints for the data sources and as such it would be extremely challenging to try to retrieve all the data from the data source repositories while keeping track of any changes to the data. There are also some required parameters (e.g. origin airport) that make it even harder to get all the data.

8.1.1. Fusion Process

The component has two separate processes based on whether the end user, as a filter parameter, has provided the GUFI number of the flight for which restrictions are being looked for. Both processes are described in the Figure 15.

- If no GUFI number is provided, the fusion service requests a list of ‘default’ flights are received from D107a. It was not possible to receive ‘default’ flights from D105 as it did not support the limit parameter and had some required parameters (origin, from, to). For each of the retrieved flights, the fusion service extracts the information about the geometry, departure and arrival time and uses it to request flight restrictions from D104. Results from D104 are returned to the client. Only the number of records defined by the limit parameter are returned.
- If a GUFI number is provided, the fusion service first tries to get the flight details based on the GUFI using the Clause 5. If no data is returned, the fusion service tries to get the flight details from the Clause 6. This means the component is supporting flown flights and flight plans at the same time. The geometry and information about the departure and arrival time are extracted from the retrieved flight details and used to request flight restrictions using the Clause 4. This service returns (if found) a collection of flight restrictions features encoded as a GeoJSON document.

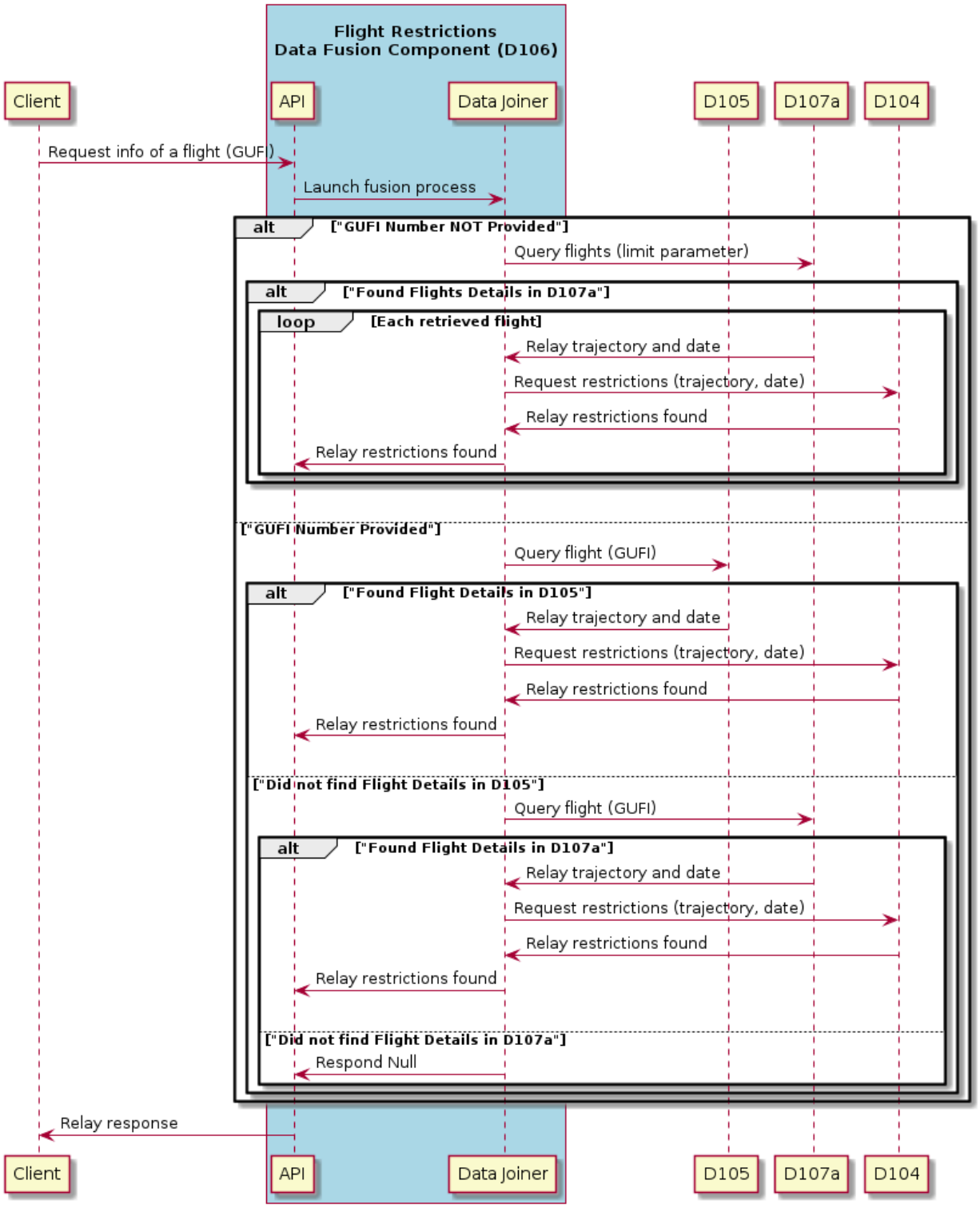


Figure 15 — Fusion Process of the Flight Restrictions Data Fusion Component

The fusion condition is an expression that can be evaluated to yes/no and can contain logical operators, spatial functions, time functions, data conversion functions or any other functions. The fusion condition is evaluated for each combination of rows from both data sources and if it evaluates as “yes”, the relation between these two rows exists. Obviously, the fusion condition

contains a reference to both rows that are currently evaluated. The simplest example for the fusion condition would be `rowDataSource1.id == rowDataSource2.id`.

The fusion condition used for this fusion service is following `spatial_intersects(rowFlightData.geometry, rowRestriction.geometry) && temporal_intersects(rowFlightData.timeRange, rowRestriction.timeRange)`. This condition is simplified. The actual fusion condition contains additional functions to unify the data formats. Omitted additional functions are conversion of geometry to the same coordinate reference system, conversion to the unified time range data type from the fields `departuretime`, `arrivaltime`, `valid_time_begin`, and `valid_time_to`.

As the geometry CRS from all data sources is the same (CRS84), the CRS conversion is not required and therefore not active. The reason why the CRS conversion was considered is the intention to build a data fusion service that is capable of fusing any two data sources that provide data as a GeoJSON document. While this is possible from the CRS point of view, the showstopper is the lack of a standardized way for encoding the temporal information. The "Features and Geometries JSON" draft offers a way on how to overcome this problem with the use of the proposed "when" parameter. For the testbed, it was not possible for the Fusion Component to use JSON-FG because the main data source for flight data did not offer data in the JSON-FG format. This is a good opportunity for future experiments, where all data sources will be delivered using the proposed JSON-FG format. At that time, generic fusion based on geometry and time would be possible.

8.1.2. Data Models Explored for Fused Data

The simplest way to fuse the data is to return all fields from both data sources in one row as displayed in the diagram (option A in Figure 16). However, there are many problems associated with this approach. First of all, there is not always a one-to-one relation. This is the TB-17 case. Second, encoding such a structure in GeoJSON format is not possible because there are two ids, two geometries and two sets of properties. A possible way to overcome this problem would be to select id and geometry from one entity and select only specific properties from entity `a` or `b`. In case of same name of the property in both entities, the property could be renamed.

The other extreme is to provide only ids of rows that are related (option B in Figure 16). However the user would be forced to gather the data manually based on ids received from the fusion service. On the other hand, the advantage would be that the data model is very simple and straightforward.

Another possibility would be to produce two datasets as result of the fusion (option C in Figure 16). The first dataset would contain all data from the first data source (id, geometry, properties) and list of IDs of related entities from the second data source. The second dataset would contain all data from the second data source and list of ids of related entities from the first data source. An alternative to this approach is to include the entire data object instead of a list of ids. However, this would produce a confusing nested data structure. The diagram displays the alternative with the list of ids.

The latter approach looked like a good alternative, but this option presented showstoppers as well. On one hand, the fusion service would have been able to get all flight restrictions for a specific flight. On the other hand, the fusion service would have not been able to get all flights for a specific flight restriction as the flight data sources (D105 and D107a) do not support

filtering based on geometry. Furthermore, as one of the flight data sources (D105) does not support the limit parameter, which controls how many records are returned, it is really hard to offer a responsive fusion service when no filter criteria are defined from the client.

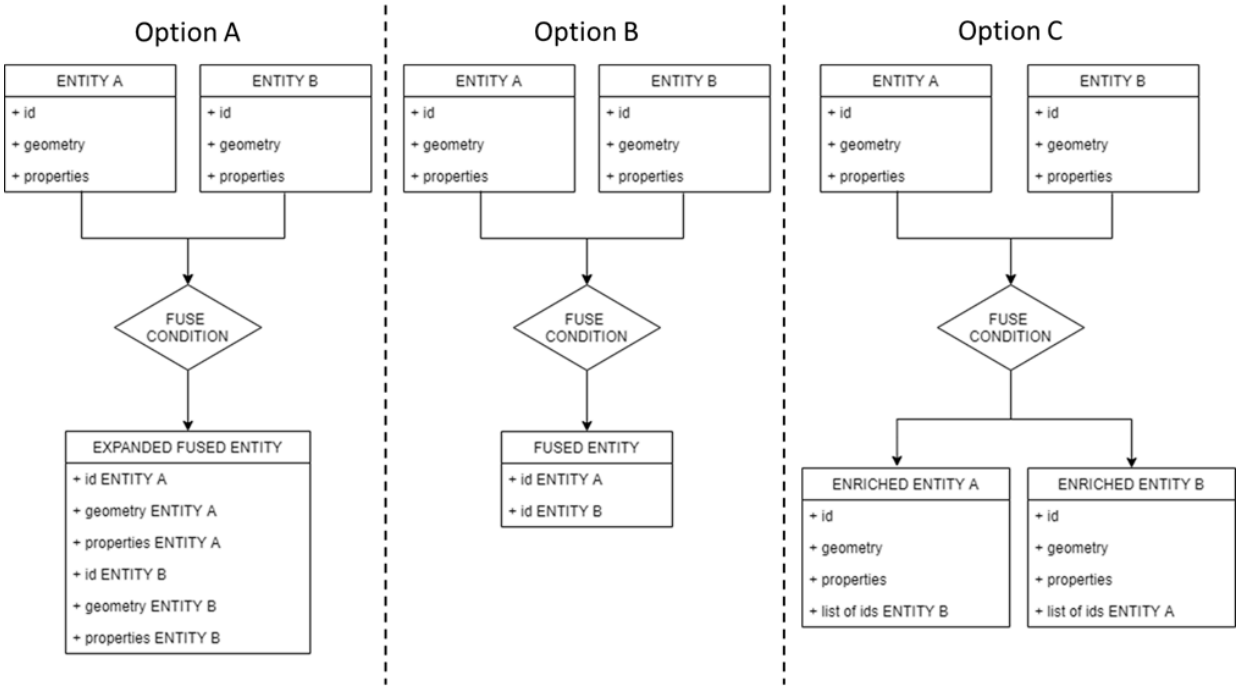


Figure 16 — Fused Data Model Options Explored

Another option considered was to force users to enter a GUFI filter to be able to use the fusion service. The API would return all restrictions that fulfill the fusion condition and are related to the flight according to the GUFI specified by the user. This idea was discarded because OGC API — Features shall support parameterless access as well.

The chosen model, as seen on Figure 17, consists of an “Enriched Entity” made up of the id, geometry and properties of an “Entity B” coupled with the ID of an “Entity A” with which it has been fused. In this example, Entity A would represent flights and Entity B would represent flight restrictions. As D105 nor D107a support filtering based on geometries (OGC API — Features: Part 3), the model was built with only one flight (Entity A) associated with a flight restriction (Entity B). Since in reality a restriction normally affects many flights, the data model uses the variable name `retrievedForGUFI` as a reference to the GUFI associated with the restriction in order to make clear that the restriction is associated to one flight in the model but can also be associated with another flight as well.

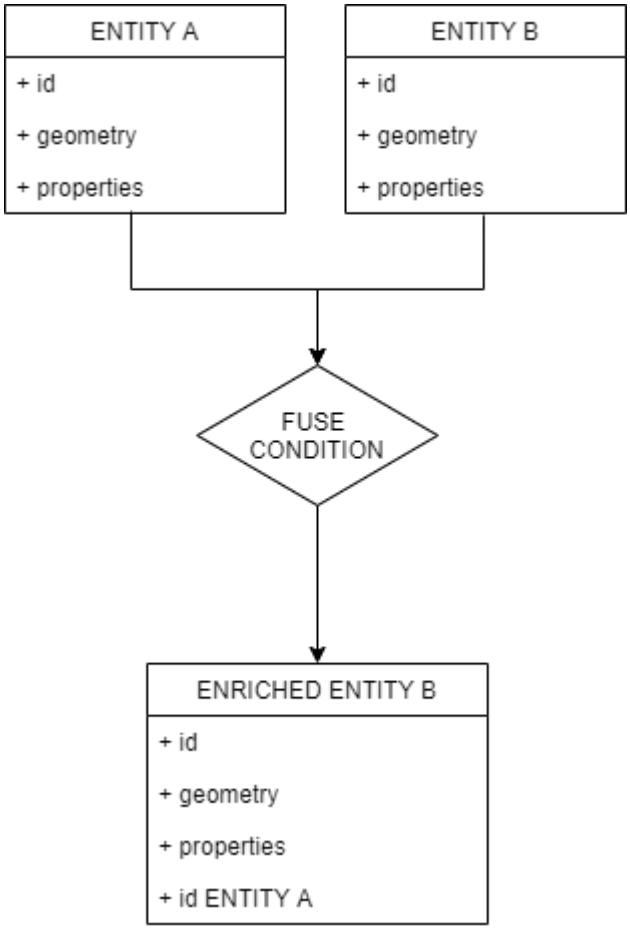


Figure 17 — Fused Data Model Option Selected

8.2. Before and After

If users manually performed this fusion, they would have to access D105 or D107a depending on whether they were requesting a flown flight or a flight plan. Users would also need to know which parameter to use for the filtering as both services use different ways of filtering based on GUFI. Next, users would have to extract the geometry information and the departure and arrival time and convert it to the format that is compatible with D104 filtering. As the geometry received from D105/D107a is complex, users would have to use a POST request to call D104 with geometry and date filter.

With this fusion service, users that have the GUFI of a flight visit only one service to retrieve the flight restrictions for that flight. This is independent of the fact whether the flight was already flown or will be flown in the future. The user does not have to make complicated transformations of geometry and date to be able to filter the flight restrictions.

8.3. Challenges and Lessons Learned

- **Potential Inefficiencies**: A potential downside of the approach is that for the use case where the user knows whether the flight was already flown or not, the system could unnecessarily contact the flight plan service as well. In the case the flight plan and flown flight should differ, it would be impossible to compare these two. The possibility is to split the fusion service into two endpoints: one for flight plan and one for flown flight. This sounds like a good use case to compare the flight plan and flown flight. Unfortunately, the data of D105 and D107a probably does not overlap and something that remains unknown due to limited data availability from the data sources.

- **Temporal Information Limitations of GeoJSON**: It is important that the input data contains data referenced by "fusion condition" that are comparable (convertible to the same unit and format). GeoJSON is a good way to do this, because of a standardized geometry encoding. However, GeoJSON lacks a standardized way of encoding temporal information. A possible solution is to transfer the data services to provide input data in JSON-FG format and use the proposed `when` property. In case any other property should be used for the fusion, additional information about the property is needed as the same data type and unit is required to be able to use it for data fusion. It would be really helpful if all data services that are providing the input data would support the 'limit' parameter and the 'filter' based on geometry. Data aggregation functions (count, group by) would be helpful as well.

- **Benefits of Storing Raw Data Locally**: Being able to persist the input data would enable the fusion service to do more advanced data fusion operations. A possible approach would be that the data services support pub/sub pattern as well. The problem of storage is of course there, but real and independent data fusion is only possible if the data are fully available for any operations or indexing. The indexing is important from a performance point of view. The input data are processed and discarded. No collecting is taking place.

9

INTERNATIONAL FLIGHT DATA FUSION COMPONENT

9 INTERNATIONAL FLIGHT DATA FUSION COMPONENT

The International Flight Data Fusion Component, developed by Skymantics, was designed to fuse flight plans from FAA and from EUROCONTROL. The component determines whether the other dataset has equivalent information of the flight, and if so, provides the link to the façade where the flight info can be accessed.

The main use case this fusion service covers is for an FAA operator to find what information EUROCONTROL has on a specific FAA flight plan, to find non-conformances, or to complete the available information. For example, for international flights, FAA databases often lack information on time interval and route for inbound flights. Also, note that the methods to calculate the route geometries are different (FAA dataset parses the `nasRouteText` in `route` property, whereas EUROCONTROL dataset uses the fusion product provided in `ftfmpointprofile` object), meaning that geometries will almost certainly differ.

The fusion service can also serve the reverse use case: that is, find what information FAA has on a specific EUROCONTROL flight plan.

9.1. Technical Architecture

As seen in the Figure 18, the Fusion Service was built with an API implementing OGC API — Processes. A script is included which queries the two façades and trying to find the flight in both datasets. APIs are queried on demand. This means there is neither a harvester collecting data nor is there a database to store the data.

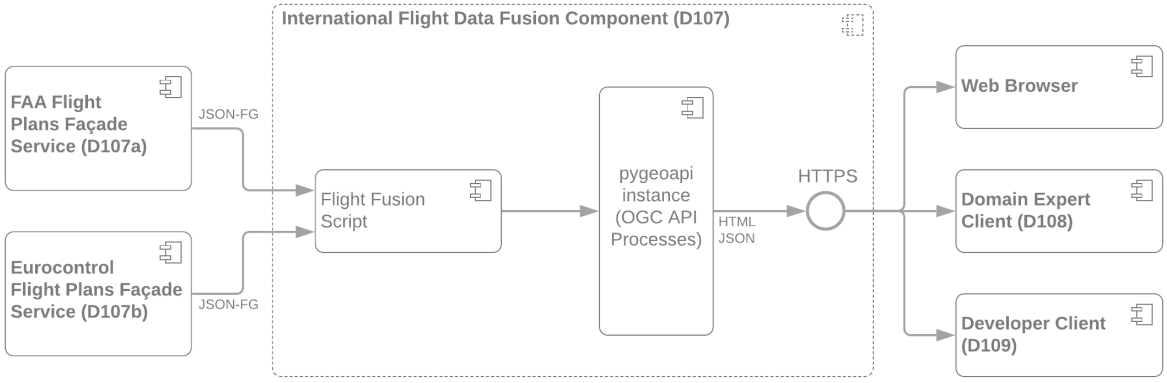


Figure 18 — Structure of the International Flight Data Fusion Component

Figure 19 describes the querying process. Input data is only an id (whether GUFI or EUROCONTROL's flight id). The client does not need to specify whether the input data is a GUFI or a flight id, as the fusion service logic handles the input accordingly.

Once the client has queried and provided a flight id, the fusion service initially seeks for the flight in FAA dataset, and turns to the EUROCONTROL dataset if not found. Once found, important fusion information is retrieved such as airports of departure and arrival, time of departure and arrival and aircraft id. If no time information is found, then other fields that might provide clues are scanned, such as coordination time.

This information is enough to query the other dataset for coincidences. It is important to take into account that the exact departure or arrival times might not match. However, searching for a flight with a specific departure and arrival airports, a specific aircraft id and that is flying at some specific instant is sufficient.

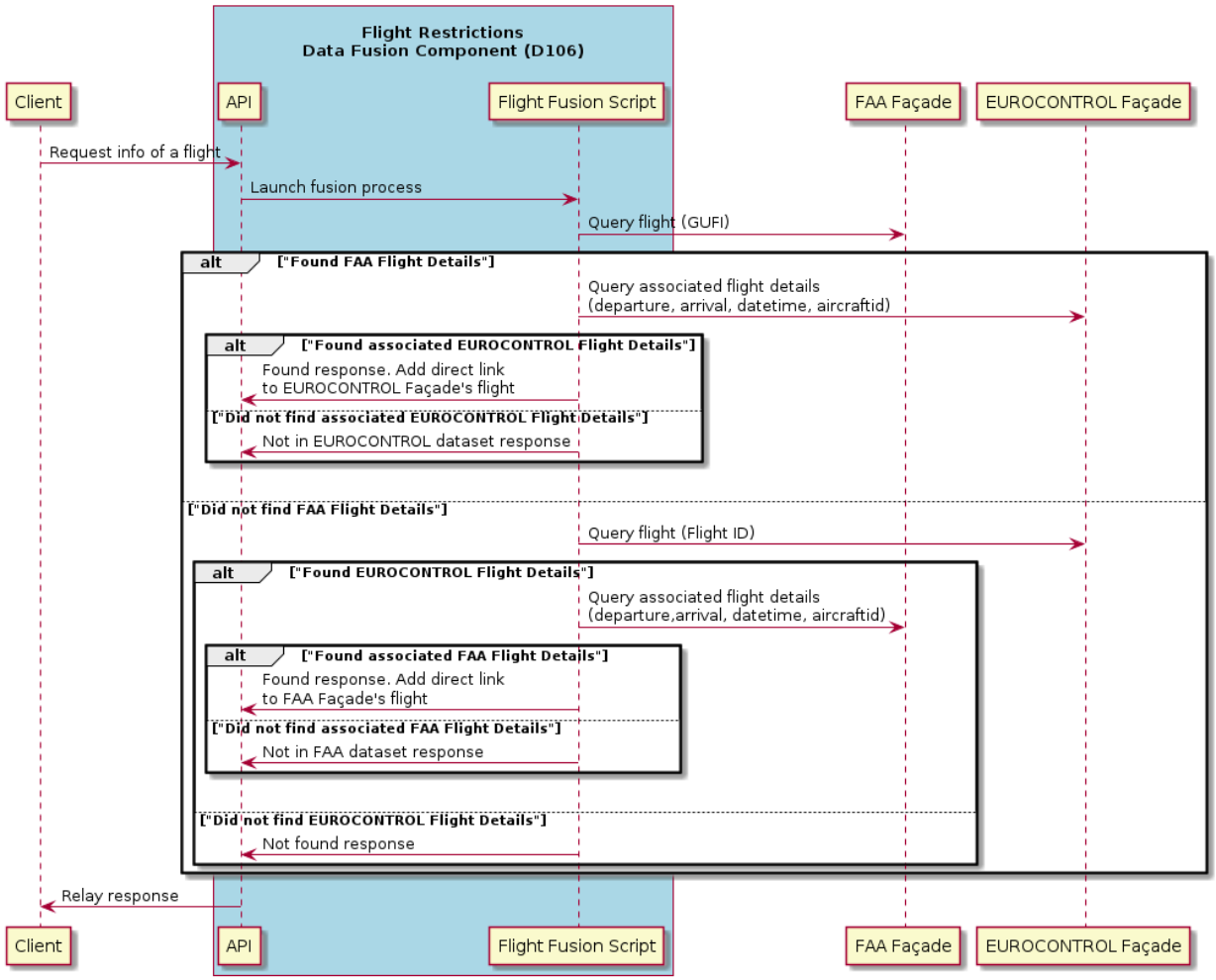


Figure 19 — Fusion Process of the International Flight Data Fusion Component

Taking into account that FAA and EUROCONTROL use different IDs for their flights, these records are not easily fused manually. The fusion service provides a simple process to find the equivalent in the other dataset or to inform that there is no equivalent.

The output is a simple JSON with the requested flight, the dataset the flight was found in (if so), the status of the process, and a list of results. The list of results is typically empty (if no match is found in the other dataset) or with one element (the exact match). However, it is sometimes possible to find several elements matching the criteria, typically due to a bug in the dataset, assigning different ids to different flight plans that correspond to the same flight.

The attributes of each result include the flight id in the corresponding dataset, the name of the dataset, the direct link to the feature (flight), and the date of last update.

9.1.1. Usage

As with any other OGC API — Processes implementation, the method used to launch the process is **POST**. Therefore, a browser is not a useful interface to test the service. Curl or Postman are good alternatives. Using the POST method and a JSON message with the aforementioned structure in the Body of the request makes the process execute.

Example 1 — Query the EUROCONTROL Flight Plan for FAA's GUFI `9fc90409-15c9-49b3-a697-7bfc6b6dc29b`:

```
curl -X POST "https://aviationapi.skymantics.com/fusion/processes/aviation-fusion/execution/" -H "Content-Type: application/json" -d "{\"inputs\": {\"flight\":\"9fc90409-15c9-49b3-a697-7bfc6b6dc29b\"}}"
```

Result:

```
{
    "outputs": {
        "flight": "9fc90409-15c9-49b3-a697-7bfc6b6dc29b",
        "dataset": "FAA",
        "status": "FAA flight found in EUROCONTROL dataset",
        "results": [
            {
                "id": "AT02815598",
                "dataset": "EUROCONTROL",
                "href": "https://aviationapi.skymantics.com/eurocontrol/collections/eu_flight_plans/items/AT02815598",
                "lastupdated": "Unknown"
            }
        ]
    }
}
```

Example 2 — Query the FAA Flight Plan for EUROCONTROL's flightid `AT02815598`:

```
curl -X POST "https://aviationapi.skymantics.com/fusion/processes/aviation-fusion/execution/" -H "Content-Type: application/json" -d "{\"inputs\": {\"flight\":\"AT02815598\"}}"
```

Result:

```
{
    "outputs": {
        "flight": "AT02815598",
        "dataset": "EUROCONTROL",
        "status": "EUROCONTROL flight found in FAA dataset",
        "results": [
            {
                "id": "9fc90409-15c9-49b3-a697-7bfc6b6dc29b",
                "dataset": "FAA",
```

```
                "href": "https://aviationapi.skymantics.com/faa/collections/
faa_flight_plans/items/9fc90409-15c9-49b3-a697-7bfc6b6dc29b",
                "lastupdated": "2021-09-08T05:18:31.394000"
            }
        ]
    }
}
```

9.2. Challenges and Lessons Learned

- **Using OGC API — Processes:** The rationale behind this decision was that the component is not providing a feature that is readily available in an existing dataset but launching a process to find a feature among two datasets. This logic fits more naturally in OGC API — Processes rather than in a standard OGC API — Features implementation or any other existing OGC API standard or draft specifications. This decision proved successful, as the Fusion Service was able to carry out its intended purpose.
    - Even though the input data was simplified to the minimum, it still **requires a nested JSON object**. This requirement is due to the requirements of OGC API — Processes. This is not a blocker but just a small inconvenience.

      For example, the input is:

```
{
    "inputs":
        {
            "flight": "9fc90409-15c9-49b3-a697-7bfc6b6dc29b"
        }
}
```

      Instead of:

```
{ "flight": "9fc90409-15c9-49b3-a697-7bfc6b6dc29b"}
```

- **Benefits of using the Fusion Service**: This component proved useful for performance (automatic process, no need of querying manually), no typo errors (queries and formats are properly encoded), and simplicity (the use of just one interface instead of two façades). If this service were not used, users would have to manually find the details of the flight he is interested to fuse, query the façade of interest entering the values of the properties that can help fuse and in the proper format and then find the complementary flight.

- Several issues were found with **data consistency** among the two datasets:
    - Some international flights in the FAA dataset were not found in the EUROCONTROL dataset and vice versa
    - Often, international FAA flights lacked information on time of departure and time of arrival, as well as trajectories outside the US airspace. This is particularly common in inbound flights.
    - Although uncommon, it was found that several GUFIs in the FAA dataset could refer to the same flight.
    - Details on flights might vary from one dataset to another, such as different departure times, arrival times or trajectories. Also, some of the properties are similar but the level of detail and information can be extremely different in others.
    - Among the dozens of properties provided for each flight, the key properties that were consistently sufficient to complete the fusion were just four: departure airport, arrival airport, aircraft id and a time reference of when the flight will be on.

10

AVIATION DOMAIN EXPERT CLIENT

10 AVIATION DOMAIN EXPERT CLIENT

The Aviation Domain Expert Client was the component designed to enable end users to request, retrieve, and visualize raw and fused aviation data coming through APIs implementing OGC API Standards.

OGC Member Hexagon developed the Aviation Domain Expert Client. This component demonstrated the usage of APIs based on OGC API Standards for the retrieval and visualization of aviation data.

10.1. Status Quo

The SWIM Data Client was based on LuciadRIA, a solution for the development of geospatial situational awareness applications through the display of information in 3D maps running on browser-based environments. LuciadRIA is part of the Luciad suite of GIS applications focused on aviation, defense and security. Hexagon has participated in previous OGC innovation initiatives exploring and demonstrating solutions for the consumption of SWIM data and the support of OGC services through their suite of Luciad applications.

The desktop version of the Luciad suite, Luciad Lightspeed, is capable of visualizing aviation data. Currently, LuciadRIA does not officially support AIXM and FIXM standards. Prototypes designed to decode AIXM and FIXM data directly in the browser were created for LuciadRIA, but are still not fully compliant with these standards. Previous LuciadRIA versions already supported connecting to OGC services such as WMS, WMTS, and WFS 1.0/2.0.

Last year, during Testbed-16, Hexagon worked with LuciadRIA version 2020.0 and had its first experience connecting to an OGC API — Features endpoint for retrieving AIXM, FIXM and GeoJSON data.

10.2. Functional Overview

Internally, the aviation client uses Hexagon LuciadRIA version 2021.0, which is fully compatible with its previous 2020.0 version used in Testbed-16. LuciadRIA is a JavaScript API that connects to, visualizes and analyzes geospatial data in the browser. LuciadRIA is used mainly for two purposes:

- **Connect to APIs, retrieve and decode data**: LuciadRIA can connect to Web Feature Services (WFS), Web Map Services (WMS), Web Map Tile Services (WMTS), and OGC API — Features endpoints. As part of this testbed, Hexagon further improved the implementation of its connector to OGC API — Features

in order to accommodate this testbed's more demanding requirements and to ensure the compatibility with the multiple servers available.

- **Render data on a map**: Once decoded, features are drawn on a 2D or 3D map. The features are geometries such as points, lines and polygons that are rendered and styled accordingly.

10.2.1. Visualization

The application displays aviation data through the LuciadRIA 3D map environment. When loaded, collections are listed in a window where users can look them up, focus the map on them, toggle their visibility or remove them from the application. Aviation data can be visualized combined with other non-aviation geospatial data. The capabilities developed in TB-17 were built to be compatible with LuciadRIA's map and feature visualization through its integration with other OGC services. Users are able to connect to a Web Map Service (WMS) or Web Feature Service (WFS) to retrieve and visualize a preferred base map (such as satellite imagery or road maps), as well as additional non-aviation data such as administrative jurisdictions or natural geographic features.

LuciadRIA has codecs built to process each data encoding. During TB-16, three prototypes were used to process AIXM, FIXM and GeoJSON data respectively. For TB-17, a prototype codec for JSON-FG was designed. This new codec extends the existing GeoJSON format and adds those features that are considered important for aviation — such as multiple CRS support, time dimension, feature type, and additional geometries. The data returned by the server is displayed "as is" in the map. The data is decoded and converted to the LuciadRIA native data model, but all original attributes are preserved including the properties, geometry and native CRS.

The data is styled accordingly to the feature type. If a feature type is recognized, the styles, such as stroke color, fill-in color, text and so forth, are properly defined for each feature type. If a feature type is not recognized, then the feature is still painted on the map but uses a default generic style. For instance, if a feature of type Runway is received, it is styled as a polygon draped on the terrain and colored in a blackish tone. Likewise, if a runway marking is received, it is rendered as white lines, polygons or points that are draped on top of the Runway. For TB-17, new feature types were introduced for which Hexagon added suitable styling.

10.2.2. Displaying Aviation Raw Data

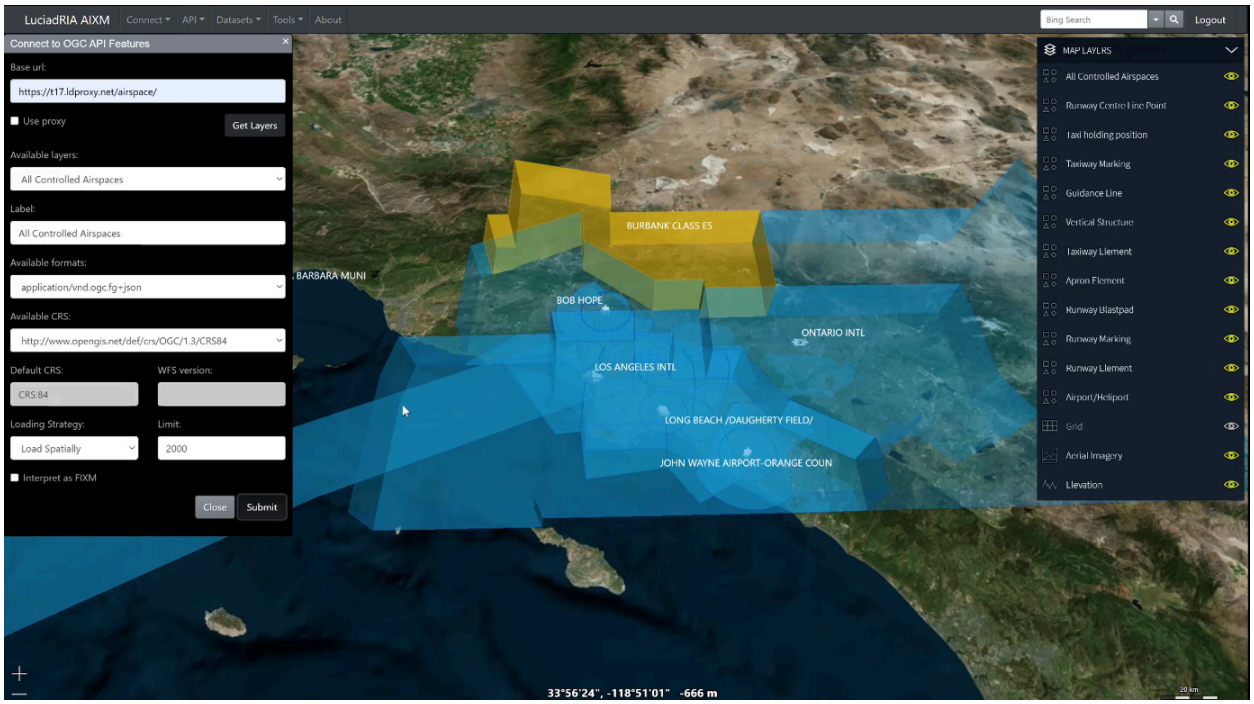


Figure 20 — LuciadRIA Displaying Airspaces Retrieved From D104

LuciadRIA can connect to multiple endpoints and load information about airspaces, airport structures, and NOTAMS. Features are visualized in the LuciadRIA 3D map environment, as seen in Figure 20, and its properties can be explored by selecting them on the map, right-clicking on them and selecting the option to view properties.

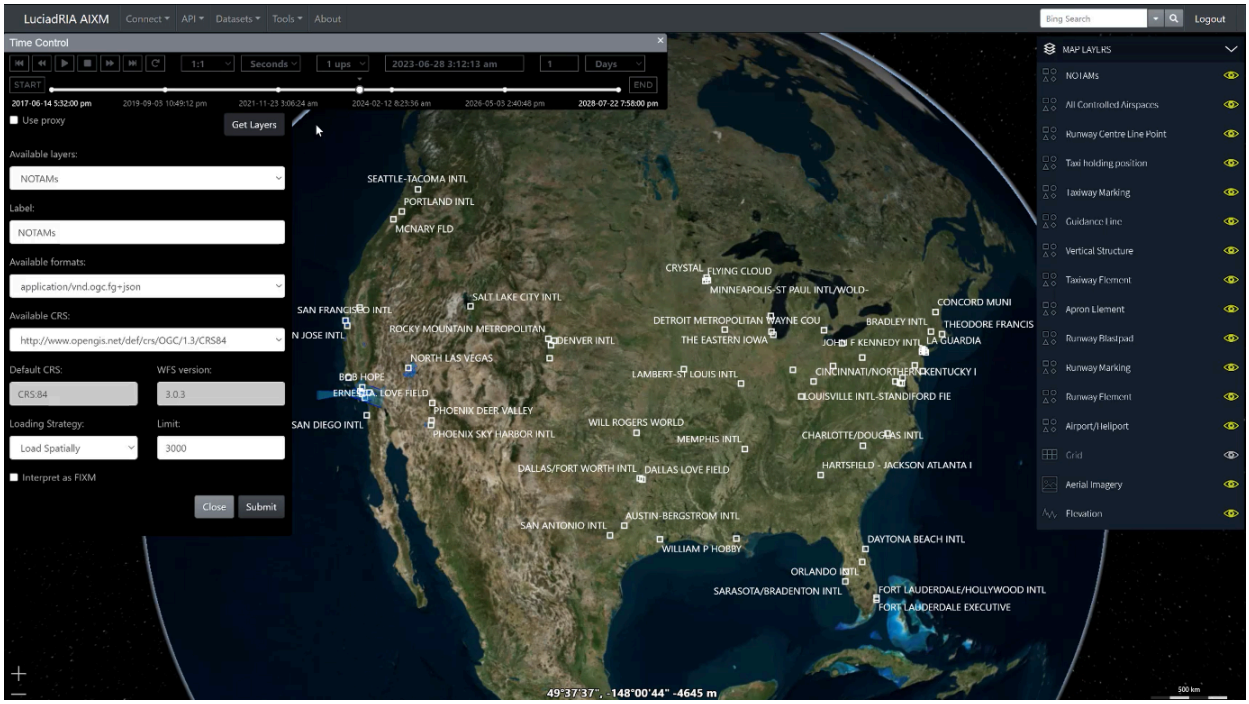


Figure 21 — LuciadRIA Displaying NOTAMs Retrieved From D104

NOTAMS are displayed on the map in the form of markings. As seen in Figure 21, a time control tool allows users to navigate throughout a given timespan. Users can move forward or backward in time and select different timespans, and NOTAMs belonging to the selected timeframe appear accordingly on the map based on the data loaded. Users can also explore feature properties of each NOTAM by right clicking them and selecting “show properties” from the context menu.

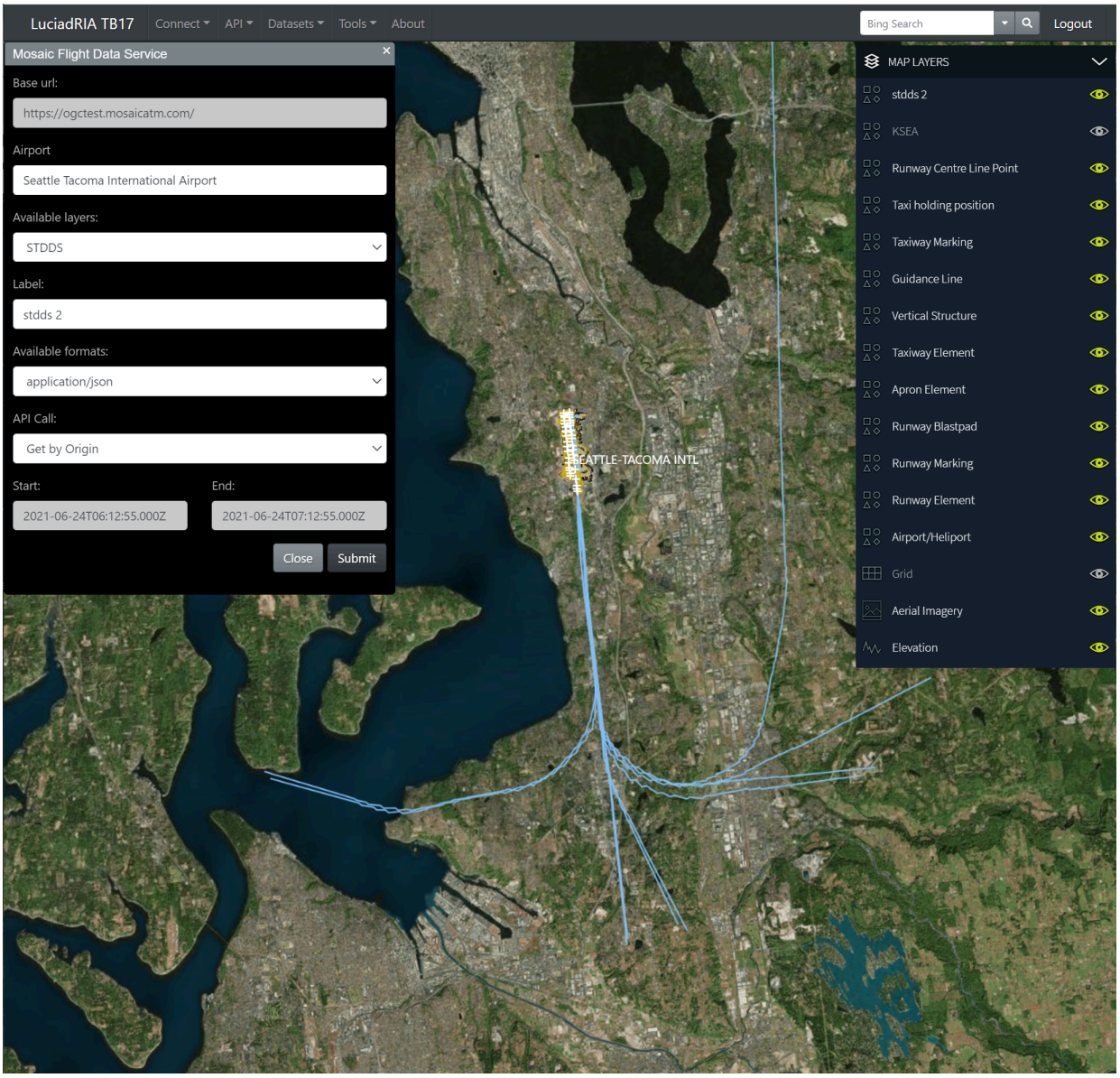


Figure 22 — LuciadRIA Displaying STDDS Flight Data Retrieved From D105

Flight data can be retrieved from D105 and D107a. For D105, a new form was created in LuciadRIA to enable users to connect to the API, select whether to retrieve data from SFDPS, STDDs, or TFMS, search flight data from different airports and display it (as seen on Figure 22).

10.2.3. Displaying Aviation Fused Data

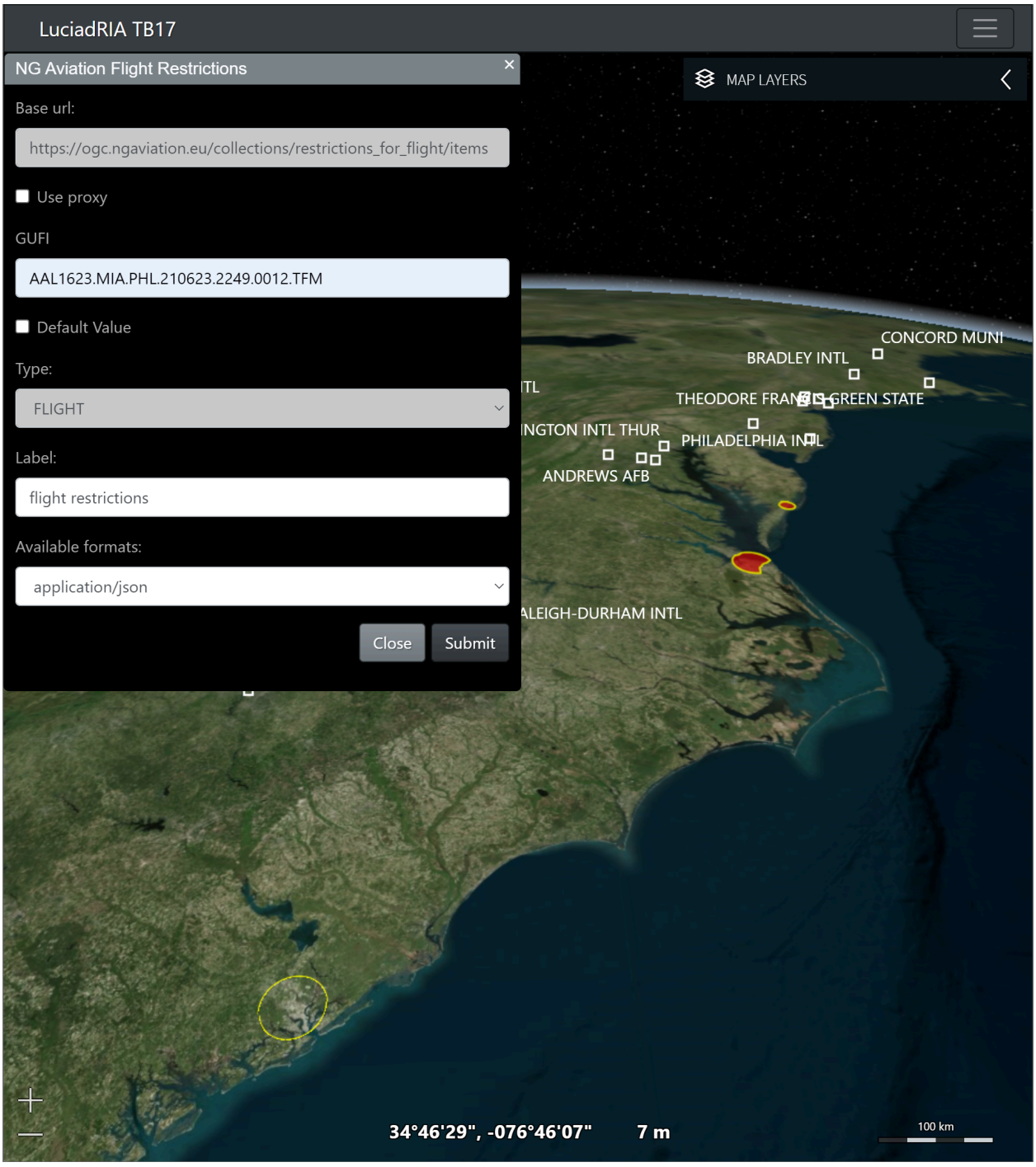


Figure 23 — LuciadRIA Displaying Flight Restrictions Retrieved From D106

A new form was created in LuciadRIA to enable users to connect to the Clause 8. In this form, the user inputs the service URL, the GUFI of the flight where restrictions are being looked for.

LuciadRIA then connects to the Fusion Service, retrieves the corresponding flight restrictions and displays them on the map, as seen on Figure 23.

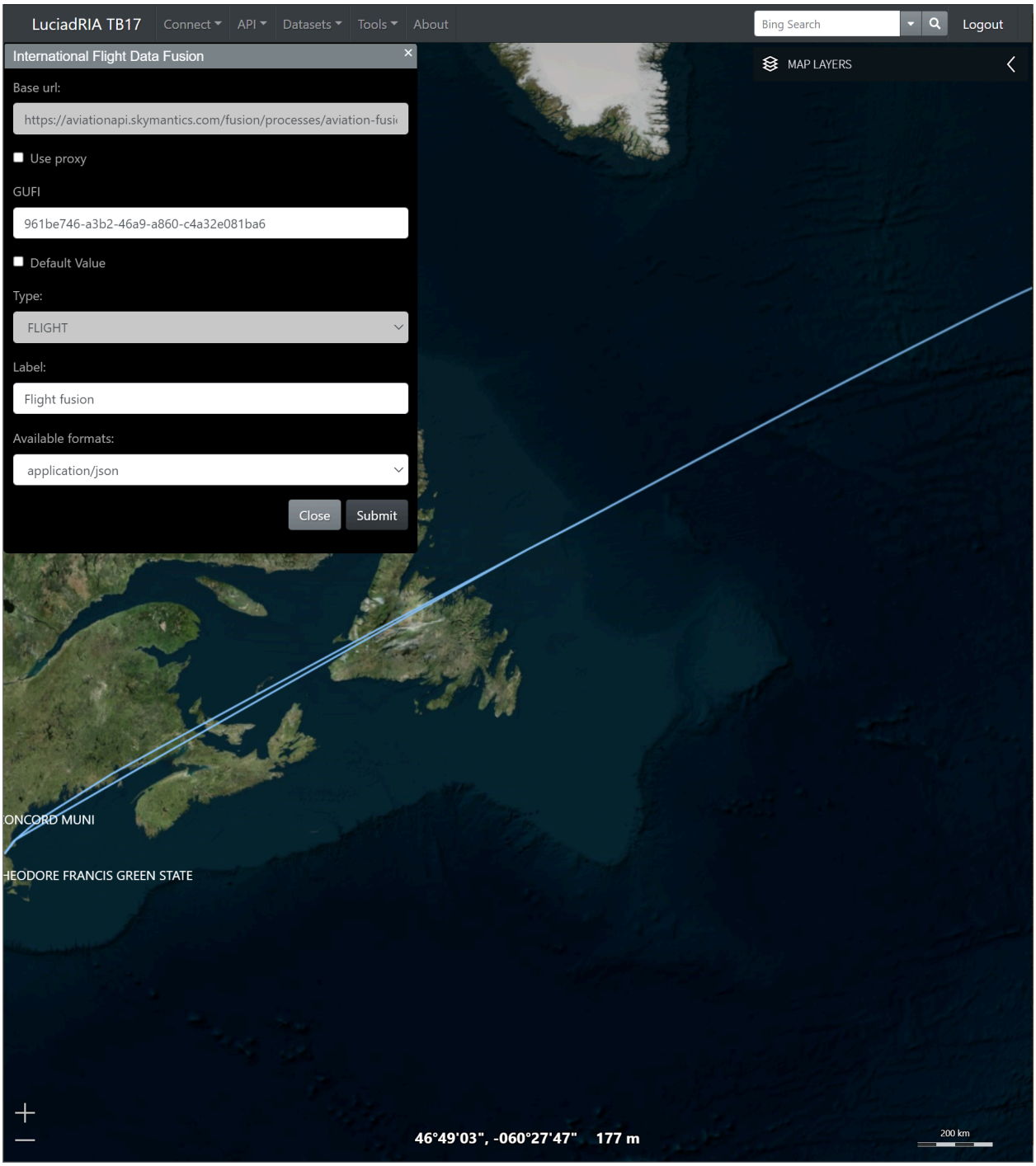


Figure 24 — LuciadRIA Displaying International Flight Data Retrieved from D107a, Using D107

A new form was created to connect in LuciadRIA to enable users to the Clause 9. The process to work with this Fusion Component is:

a) The user inputs the service URL and the GUFI or Flight ID being looked for

b) LuciadRIA requests the first flight feature from D107a based on the original GUFI of Flight ID input by the user.

c) LuciadRIA sends the request to the Fusion Service.

d) LuciadRIA retrieves from the response of the Fusion Component the corresponding D107a flight URI.

e) LuciadRIA requests the second flight feature from D107a based on the retrieved URI.

f) LuciadRIA displays both flight features on the map, as seen on Figure 24.

10.3. Lessons Learned

- **Use of JSON-FG**: The biggest challenge was to implement a new decoder for data in the draft JSON-FG format. This was mainly because during TB-17 the implementation evolved from the original set of requirements expressed in the Call for Participation.

  The original context for using JSON-FG was to implement support for multiple CRS other than CRS84. However, during the testbed additional features were identified and added, for instance

  - New syntax for geometry.
  - Support for time-dimension.
  - New geometries

- **Ensure consistency between the multiple backend servers**: Another challenge was the fact that the TB-17 participants had implemented diverse interpretations of OGC API — Features. During TB-16 there was only one backend server while for TB-17 there were multiple backend servers from multiple vendors, with each server having some unique characteristics. During TB-17, Hexagon managed to come to a consensus agreement where the multiple participants adapted their implementation of the OGC API — Features standard to ensure consistency between the many implementations. Thanks to small changes and fine tuning made by several participants, it became easier for the Aviation client to connect to the different back ends.

  Nevertheless, some APIs did end up presenting differences when compared with the OGC API — Features Standard, which forced Hexagon to make custom implementations.

10.3.1. Contrast With Testbed-16 Aviation Task Challenges

- One of the lessons learned from TB-16 was that the GeoJSON format is very generic and therefore it becomes **impossible to properly identify a feature type** (this was one of the reasons TB-16 used AIXM). In TB-17, during the design of JSON-FG, a decision was made to add a feature type attribute in a similar way as specified in AIXM. This supported the ability to identify the type of the feature and interpret it accordingly, thus solving the problem identified in TB-16.

11

AVIATION DEVELOPER CLIENT

11 AVIATION DEVELOPER CLIENT

While the Aviation Domain Expert Client allows an aviation decision maker to seamlessly access the raw and fusion data services to make operational decisions (i.e., a decision support tool), the needs of a developer are entirely different. Developed by Mosaic ATM, the Aviation Developer Client was designed to support aviation systems developers with an easy and quick access to documentation and metadata of the APIs serving Aviation raw and fused data.

11.1. Functional Overview

The Aviation Developer Client is a static web application built using custom HTML/ Javascript website that provides seamless access to partner API environments (endpoints and documentation). This approach was taken to support not only a developer's needs but also other aviation stakeholders that are looking for data to create an application and/or product. Client-side scripting supports rapid search capabilities of partner API functionality and immediate access to documentation. Additionally, a WebCrawler was created to provide search functionality to analyze and collect partner API metadata. This search capability, as seen in Figure 26, enables a developer to search for any endpoints within the entire data fusion architecture.

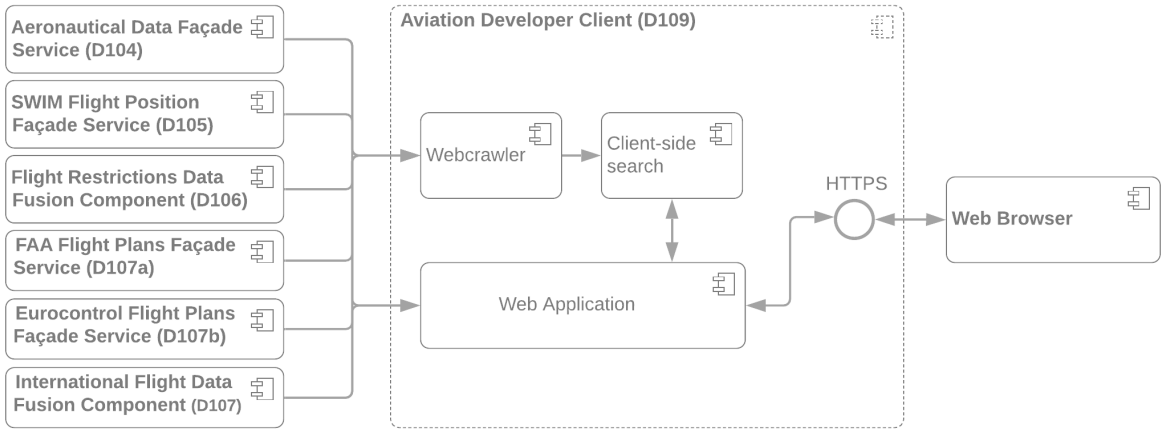


Figure 25 — Structure of the Aviation Developer Client

As seen on Figure 25, this component provides immediate, real-time access to all the API documentation and endpoints in the architecture (from D104 to D107). This provides developers with a common entry point to the API ecosystem where a developer (or any other stakeholder for that matter) can read and/or experiment with the APIs. The client application was built with the developer in mind but also built recognizing that other stakeholders in the value chain would benefit from a consolidated, easy-to-access platform that considers all the APIs available.

Without this component, end users would need to visit each API separately and understand the value of the service and how it plays a role in the broader API ecosystem. With the Aviation Developer Client, all the APIs are listed in one location and context is provided on the value of the services. Additionally, developers and/or other stakeholders for that matter have a common entry point to the services of services architecture and have everything at their disposal to understand and make use of the APIs.

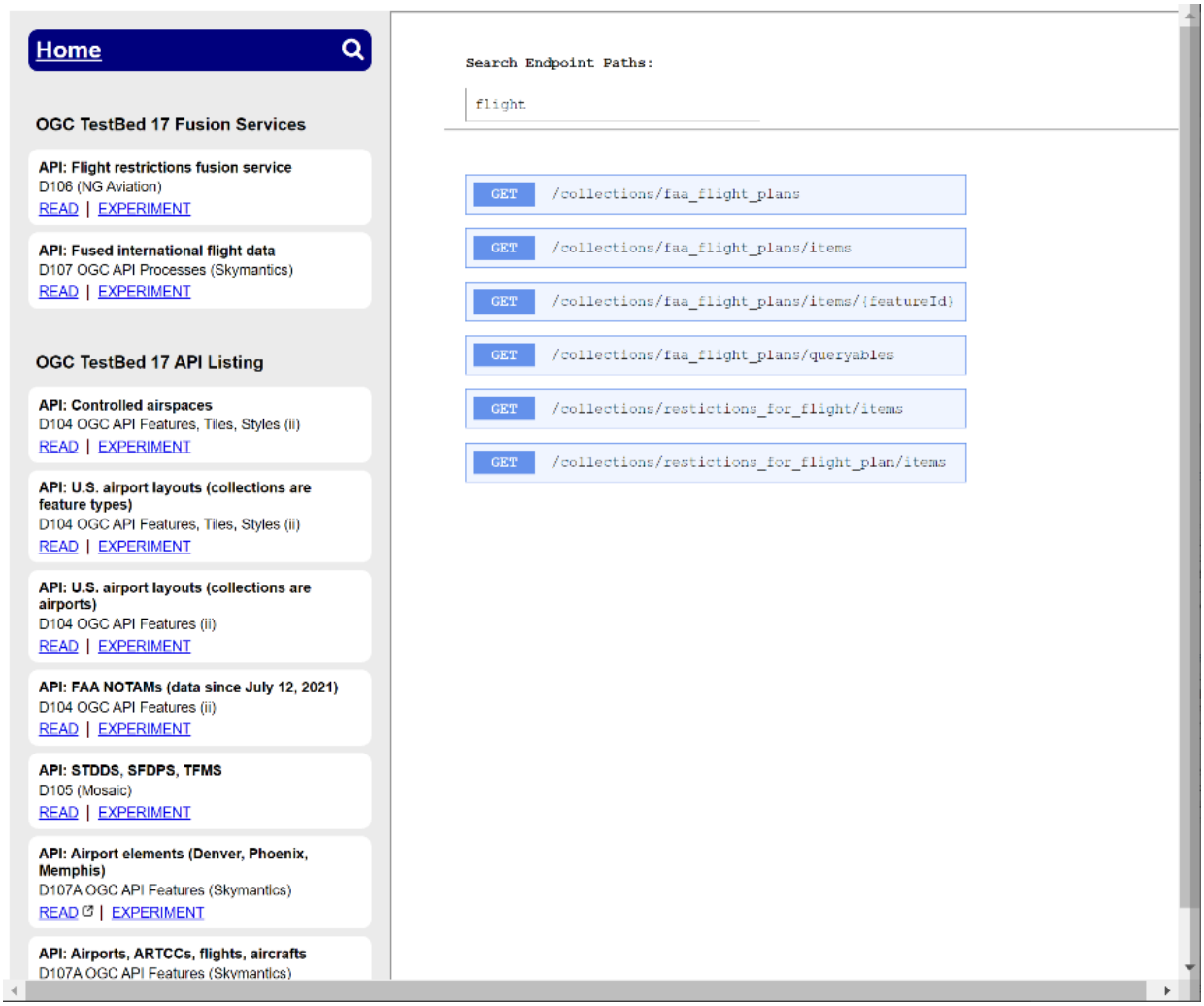


Figure 26 — Search Functionality of the Aviation Developer Client

11.2. Challenges and Lessons Learned

11.2.1. Importance of API Consistency

The integration of the services with D109 was very straightforward because all the services used the OpenAPI specification. The same documentation format allowed the web crawler to process the information in a consistent way. The primary lesson learned is the importance of API

consistency. If the APIs are built in a similar way, with similar path structures then the ability to search across the APIs is relatively simple and provides the end user with many benefits.

12

TECHNOLOGY INTEGRATION EXPERIMENTS (TIES)

12 TECHNOLOGY INTEGRATION EXPERIMENTS (TIES)

The TB-17 Aviation Task Technology Integration Experiments (TIEs) focused on the exchange of aviation data through OGC APIs. Each TIE explored under different circumstances the potential of OGC APIs with the objective of relaying aviation data.

12.1. TIE Summary Table

The following table summarizes the TIEs performed during Testbed-17

Table 2 — Technology Integration Experiments Overview

SERVER\CLIENT	D106 FUSION SERVICE	D107 FUSION SERVICE	D108 END-USER CLIENT	D109 DEVELOPER CLIENT
D104 Aviation APIs	4/4	n/a	2/2	3/3
D105 Aviation APIs	2/3	n/a	1/1	3/3
D107a Aviation APIs	3/3	4/4	2/2	3/3
D106 Fusion Service	n/a	n/a	3/3	3/3
D107 Fusion Service	n/a	n/a	2/2	3/3

12.2. TIE Functional Tests

12.2.1. D104 Aviation API for D106 Fusion Service

This TIE successfully tested the following scenarios. Details are provided in Table 3.

- Retrieve flight restrictions data in GeoJSON format / Ability to retrieve the 'default' data (no required filters, returning first X records).
- Retrieve flight restrictions data filtered by geometry.

- Retrieve flight restrictions data filtered by feature type (airport, airspace, etc.).
- Confirm that the data provided by (D104 and D105) and (D104 and D107a) are in the same date range.

Table 3 — TIE Functional Test — D104 > D106

#	FUNCTION	DESCRIPTION	CLIENT ACTION	SERVER RESPONSE	SUCCESS CRITERION
1	Retrieve flight restrictions data in GeoJSON format / Ability to retrieve the 'default' data (no required filters, returning first X records)	Request, receive, parse flight restrictions	User access the flight_ restrictions collection	The service returns flight restrictions encoded as Geo JSON	Flight restrictions encoded as Geo JSON
2	Retrieve flight restrictions data filtered by geometry	Request, receive, parse flight restrictions filtered by geometry	User access the flight_ restrictions collection with GUFI filter specified	The service use GUFI to retrieve geometry and returns all restrictions encoded as GeoJSON filtered by received geometry	All flight restrictions encoded as Geo JSON for this flight (GUFI) are returned
3	Retrieve flight restrictions data filtered by feature type (airport, airspace, etc.)	Request, receive, parse flight restrictions filtered by feature type	User access the flight_ restrictions collection with restriction type filter defined	The service returns all restrictions encoded as Geo JSON filtered by specified restriction type	All flight restrictions encoded as Geo JSON for specified restriction type are returned
4	Confirm that the data provided by (D104 and D105) and (D104 and D107a) are in the same date range	Check the documentation of D104, D105 and D107a to confirm that the data date orange overlaps	Open the documentation of D104, D105 and D107a	Returns the information about the data date ranges of both services	The received data date ranges overlaps

12.2.2. D105 Aviation API for D106 Fusion Service

This TIE successfully tested the following scenarios. Details are provided in Table 4.

- Retrieve historical flight data filtered by GUFI (Globally Unique Flight Identifier) including departure and arrival time.
- Confirm that the data provided by D104 and D105 are in the same date range.

Table 4 — TIE Functional Test — D105 > D106

#	FUNCTION	DESCRIPTION	CLIENT ACTION	SERVER RESPONSE	SUCCESS CRITERION
1	Retrieve historical flight data in Geo JSON format / Ability to retrieve the 'default' data (no required filters, returning first X records)	Request, receive, parse flight data	User access the data endpoint without any parameters	**NEGATIVE RESULT:** No server response is returned, as the data endpoint has required parameters. The request without parameters was not possible.	Flight restrictions encoded as Geo JSON
2	Retrieve historical flight data filtered by GUFI (Globally Unique Flight Identifier) including departure and arrival	Request, receive, parse flight restrictions filtered by GUFI	User access the data endpoint with GUFI filter specified	The service use GUFI to retrieve geometry, departure and arrival time and returns all restrictions encoded as Geo JSON filtered by received geometry, departure and arrival time	All flight restrictions encoded as Geo JSON for this flight (GUFI) are returned
3	Confirm that the data provided by D104 and D105 are in the same date range	Check the documentation of D104 and D105 to confirm that the data date orange overlaps	Open the documentation of D104 and D105	Returns the information about the data date ranges of both services	The received data date ranges overlaps

12.2.3. D107a Aviation API for D106 Fusion Service

This TIE successfully tested the following scenarios. Details are provided in Table 5.

- Retrieve flight plan data in GeoJSON format / Ability to retrieve the 'default' data (no required filters, returning first X records).
- Retrieve flight plan data filtered by GUFI (Globally Unique Flight Identifier) including departure and arrival time.
- Confirm that the data provided by D104 and D107a are in the same date range.

Table 5 — TIE Functional Test — D107a > D106

#	FUNCTION	DESCRIPTION	CLIENT ACTION	SERVER RESPONSE	SUCCESS CRITERION
1	Retrieve flight plan data in GeoJSON format / Ability to retrieve the 'default' data (no required filters, returning first X records)	Request, receive, parse flight restrictions	User access the flight restrictions collection	The service returns flight restrictions encoded as Geo JSON for all 'default' flights retrieved	Flight restrictions encoded as Geo JSON
2	Retrieve flight plan data filtered by GUFI (Globally Unique Flight Identifier) including departure and arrival time	Request, receive, parse flight restrictions filtered by GUFI	User access the flight restrictions collection with GUFI filter specified	The service use GUFI to retrieve geometry, departure and arrival time and returns all restrictions encoded as Geo JSON filtered by received geometry, departure and arrival time	All flight restrictions encoded as Geo JSON for this flight (GUFI) are returned
3	Confirm that the data provided by D104 and D107a are in the same date range	Check the documentation of D104 and D107a to confirm that the data date orange overlaps	Open the documentation of D104 and D107a	Returns the information about the data date ranges of both services	The received data date ranges overlaps

12.2.4. D107a Aviation API for D107 Fusion Service

This TIE successfully tested the following scenarios. Details are provided in Table 6.

- Find flight in FAA façade based on FAA GUFI, if it exists.
- Find flight in EUROCONTROL façade based on EUROCONTROL flightID, if it exists.
- Find flight in EUROCONTROL façade based on FAA GUFI, if it exists.
- Find flight in FAA façade based on EUROCONTROL flightID, if it exists.

Table 6 — TIE Functional Test — D107a > D107

#	FUNCTION	DESCRIPTION	CLIENT ACTION	SERVER RESPONSE	SUCCESS CRITERION
1	GUFI found in FAA	Fusion service is capable of finding a flight in FAA Façade if the flight requested is a valid FAA GUFI	Launch the process with input flight of a valid FAA GUFI	Process is successful, flight dataset is acknowledged as FAA	The server acknowledges that the input flight is in FAA dataset
2	EUR flightID in EUR	Fusion service is capable of finding a flight in EUROCONTROL Façade if the flight requested is a valid EUROCONTROL flightID	Launch the process with input flight of a valid EUROCONTROL flightID	Process is successful, flight dataset is acknowledged as EUROCONTROL	The server acknowledges that the input flight is in EUROCONTROL dataset
3	GUFI found in EUR	Fusion service is capable of finding a flight in EUROCONTROL Façade if the flight requested is a valid FAA GUFI (and if it exists in EUROCONTROL dataset)	Launch the process with input flight of a valid FAA GUFI	Process is successful, results provide a link to EUROCONTROL façade	The link points to a flight with similar features as the original flight but in different dataset, hence it is the same flight
4	EUR flightID found in FAA	Fusion service is capable of finding a flight in FAA Façade if the flight requested is a valid EUROCONTROL flightID (and if it exists in FAA dataset)	Launch the process with input flight of a valid EUROCONTROL flightID	Process is successful, results provide a link to FAA façade	The link points to a flight with similar features as the original flight but in different dataset, hence it is the same flight

12.2.5. D104 Aviation API for D108 Client Application

This TIE successfully tested the following scenarios. Details are provided in Table 7.

- Retrieve available collections and capabilities.
- Retrieve features per collection.

Table 7 — TIE Functional Test — D104 > D108

#	FUNCTION	DESCRIPTION	CLIENT ACTION	SERVER RESPONSE	SUCCESS CRITERION
1	Retrieve available collections and capabilities (supported formats and supported CRS's)	Request, receive, information about collections	Form landing page request, parse response, display and/or act on it	The service returns information about the available collections as JSON	Collections are found and presented to the user with available capabilities
2	Retrieve features per collection	Request, receive, parse features including properties and geometries	Request features, decode and display on a map	Features can be decoded from Geo JSON and JSON-FG and CRS can be selected	Features are properly decoded and presented in the map

12.2.6. D105 Aviation API for D108 Client Application

This TIE successfully tested the following scenarios. Details are provided in Table 8.

- Retrieve flight SFDPS.
- Retrieve flight STDDS.
- Retrieve flight TFMS.

Table 8 — TIE Functional Test — D105 > D108

#	FUNCTION	DESCRIPTION	CLIENT ACTION	SERVER RESPONSE	SUCCESS CRITERION
1	Retrieve flight SFDPS	Request, receive, information about flight based on airport and origin/destination	Load flight info for a specific airport, origin/destination	The service returns flight encoded as GeoJSON	Flight paths found are presented to the user in a map
2	Retrieve flight STDDS	Request, receive, information about flight based on airport and origin/destination	Load flight info for a specific airport, origin/destination	The service returns flight encoded as GeoJSON	Flight paths found are presented to the user in a map
3	Retrieve flight TFMS	Request, receive, information about flight based on airport and origin/destination	Load flight info for a specific airport, origin/destination	The service returns flight encoded as GeoJSON	Flight paths found are presented to the user in a map

12.2.7. D107a Aviation API for D108 Client Application

This TIE successfully tested the following scenarios. Details are provided in Table 9.

- Retrieve available collections and capabilities.
- Retrieve features per collection.

Table 9 — TIE Functional Test — D107a > D108

#	FUNCTION	DESCRIPTION	CLIENT ACTION	SERVER RESPONSE	SUCCESS CRITERION
1	Retrieve available collections and capabilities (supported formats and supported CRS's)	Request, receive, information about collections	Form landing page request, parse response, display and/or act on it	The service returns information about the available collections as JSON	Collections are found and presented to the user with available capabilities
2	Retrieve features per collection	Request, receive, parse features including properties and geometries	Request features, decode and display on a map	Features can be decoded from Geo JSON and JSON-FG and CRS can be selected	Features are properly decoded and presented in the map

12.2.8. D106 Fusion Service for D108 Client Application

This TIE successfully tested the following scenarios. Details are provided in Table 10.

- Retrieve flight restrictions and display in map.

Table 10 — TIE Functional Test — D106 > D108

#	FUNCTION	DESCRIPTION	CLIENT ACTION	SERVER RESPONSE	SUCCESS CRITERION
1	Retrieve flight restrictions and display in map	Request, receive, information about flight restrictions based on GUFI	Load flight restrictions for a specific GUFI	The service returns flight restrictions encoded as GeoJSON	Flight restrictions found are presented to the user in a map

12.2.9. D107 Fusion Service for D108 Client Application

This TIE successfully tested the following scenarios. Details are provided in Table 11.

- Retrieve a Flight ID based on a GUFI.
- Retrieve a GUFI based on a Flight ID.

Table 11 — TIE Functional Test — D107 > D108

#	FUNCTION	DESCRIPTION	CLIENT ACTION	SERVER RESPONSE	SUCCESS CRITERION
1	Retrieve a Flight ID based on a GUFI	Input GUFI	Receive the matching GUFI flight data and render both flights on screen EUROCONTROL / FAA	The service returns a link pointing to a valid D107a flight feature corresponding to the input GUFI	If a link is returned, the returned link points to a valid D107a flight feature corresponding to the input Flight ID
2	Retrieve a GUFI based on a Flight ID	Input Flight ID	The service returns a link pointing to a valid D107a flight feature corresponding to the input Flight ID	The service returns matching flight as an entry to a OGC API — Features entry with flight ID	If a link is returned, the returned link points to a valid D107a flight feature corresponding to the input GUFI

12.2.10. D104 Fusion Service for D109 Client Application

This TIE successfully tested the following scenarios. Details are provided in Table 16.

- Access API documentation from D104 through READ link.
- Access API endpoints from D104 through EXPERIMENT link.
- Search for D104 API endpoints.

Table 12 — TIE Functional Test — D104 > D109

#	FUNCTION	DESCRIPTION	CLIENT ACTION	SERVER RESPONSE	SUCCESS CRITERION
1	Access API documentation from D104 through READ link	Access to API documentation to understand service	User access to API documentation	The API documentation is accessible	Access to API documentation allowing user to READ through APIs
2	Access API endpoints from D104 through EXPERIMENT link	Access to API endpoints to explore and test service	User access to API endpoints	The API endpoints are accessible and working	Access to API endpoints allowing user to EXPERIMENT with APIs
3	Search for D104 API endpoints	Keyword search enabling the user to quickly access APIs on specific data	User access Airport, Airspace, and NOTAM endpoints	The search feature finds D104 endpoints	Searching and finding D104 APIs by keywords

12.2.11. D105 Aviation API for D109 Client Application

This TIE successfully tested the following scenarios. Details are provided in Table 13.

- Access API documentation from D105 through READ link.
- Access API endpoints from D105 through EXPERIMENT link.
- Search for D105 API endpoints.

Table 13 — TIE Functional Test — D105 > D109

#	FUNCTION	DESCRIPTION	CLIENT ACTION	SERVER RESPONSE	SUCCESS CRITERION
1	Access API documentation from D105 through READ link	Access to API documentation to understand service	User access to API documentation	The API documentation is accessible	Access to API documentation allowing user to READ through APIs
2	Access API endpoints from D105 through EXPERIMENT link	Access to API endpoints to explore and test service	User access to API endpoints	The API endpoints are accessible and working	Access to API endpoints allowing user to EXPERIMENT with APIs
3	Search for D105 API endpoints	Keyword search enabling the user to quickly access APIs on specific data	User access Flight Position endpoints	The search feature finds D105 endpoints	Searching and finding D105 APIs by keywords

12.2.12. D107a Aviation API for D109 Client Application

This TIE successfully tested the following scenarios. Details are provided in Table 14.

- Access API documentation from D107a through READ link.
- Access API endpoints from D107a through EXPERIMENT link.
- Search for D107a API endpoints.

Table 14 — TIE Functional Test — D107a > D109

#	FUNCTION	DESCRIPTION	CLIENT ACTION	SERVER RESPONSE	SUCCESS CRITERION
1	Access API documentation from	Access to API documentation to understand service	User access to API documentation	The API documentation is accessible	Access to API documentation

#	FUNCTION	DESCRIPTION	CLIENT ACTION	SERVER RESPONSE	SUCCESS CRITERION
	D107a through READ link				allowing user to READ through APIs
2	Access API endpoints from D107a through EXPERIMENT link	Access to API endpoints to explore and test service	User access to API endpoints	The API endpoints are accessible and working	Access to API endpoints allowing user to EXPERIMENT with APIs
3	Search for D107a API endpoints	Keyword search enabling the user to quickly access APIs on specific data	User access Airport Elements endpoints	The search feature finds D107a endpoints	Searching and finding D107a APIs by keywords

12.2.13. D106 Fusion Service for D109 Client Application

This TIE successfully tested the following scenarios. Details are provided in Table 15.

- Access API documentation from D106 through READ link.
- Access API endpoints from D106 through EXPERIMENT link.
- Search for D106 API endpoints.

Table 15 — TIE Functional Test — D106 > D109

#	FUNCTION	DESCRIPTION	CLIENT ACTION	SERVER RESPONSE	SUCCESS CRITERION
1	Access API documentation from D106 through READ link	Access to API documentation to understand service	User access to API documentation	The API documentation is accessible	Access to API documentation allowing user to READ through APIs
2	Access API endpoints from D106 through EXPERIMENT link	Access to API endpoints to explore and test service	User access to API endpoints	The API endpoints are accessible and working	Access to API endpoints allowing user to EXPERIMENT with APIs
3	Search for D106 API endpoints	Keyword search enabling the user to quickly access APIs on specific data	User access restrictions for flights endpoints	The search feature finds D106 endpoints	Searching and finding D106 APIs by keywords

12.2.14. D107 Fusion Service for D109 Client Application

This TIE successfully tested the following scenarios. Details are provided in Table 16.

- Access API documentation from D107 through READ link.
- Access API endpoints from D107 through EXPERIMENT link.
- Search for D107 API endpoints.

Table 16 — TIE Functional Test — D107 > D109

#	FUNCTION	DESCRIPTION	CLIENT ACTION	SERVER RESPONSE	SUCCESS CRITERION
1	Access API documentation from D107 through READ link	Access to API documentation to understand service	User access to API documentation	The API documentation is accessible	Access to API documentation allowing user to READ through APIs
2	Access API endpoints from D107 through EXPERIMENT link	Access to API endpoints to explore and test service	User access to API endpoints	The API endpoints are accessible and working	Access to API endpoints allowing user to EXPERIMENT with APIs
3	Search for D107 API endpoints	Keyword search enabling the user to quickly access APIs on specific data	User access fused international flight data endpoints	The search feature finds D107 endpoints	Searching and finding D107 APIs by keywords

13

API DISCUSSION

13 API DISCUSSION

The objective of the Testbed-17 Aviation Task was to explore the potential for OGC APIs in exchanging Aviation data in an interoperable manner. The activities performed during the Testbed, including the design of components, the execution of TIEs between them, as well as the conversations between participants, are outlined in the following lessons learned section.

13.1. Transitioning the Current APIs of SWIM Services Into Standards-Based APIs

The advantage of working with an API based on standard building blocks is that a single implementation at the client side is capable of connecting to servers from multiple providers with little to no customization. Transitioning current SWIM Services into standards-based APIs would therefore facilitate the upgrade and development of current and new aviation data services.

One of the differences between the current SWIM services and APIs based on OGC API Standards is that the latter work with a **request/response** messaging pattern while SWIM works mainly on a **publish/subscribe** pattern, also known as pub/sub (there are some request/response SWIM services). During TB-17, all façades but one (d107b, connecting to EUROCONTROL's NMB2B) connected to pub/sub data services.

Pub/sub services are best suited when a change in some attributes (like the cancellation of a flight or a delay) must trigger an action (such as informing passengers of the change or re-planning a route for other flights). Ideally, the data sent in these messages should only include the modified attributes. Request/response services are best suited for data that changes less frequently, when the current status of a feature is requested (like a flight plan or a flight restriction), or for static features (such as aeronautical information). An alternate architecture is a combination of both, with one request/response service offering the current status and a pub/sub service informing of changes. Another perspective is that pub/sub services require a higher commitment (setting up subscriptions, monitoring the feeds to extract and store the relevant information, etc.) while the Web APIs provide the relevant information on-demand when needed by the user.

13.1.1. Developing Façades in Front of SWIM Services

Due to the effort required and the large number of actors involved, the transition of SWIM services into using standards-based APIs can be considered a long-term endeavor, the development of façades becomes a practical consideration. A façade is a component that fetches data from a service and offers that data through its own API. The SWIM Data Relay API built during Testbed-16 was a façade, as well as this Testbed's D104 (only the API serving NOTAMs), D105, D107a and D107b.

The cases when deploying façades can be beneficial are the following:

- When transitioning the authoritative source into using OGC API Standards is not immediately possible.
- When the authoritative source provides data only through pub/sub services, and there is a value to support also request/response.
- When you need to transform the format of the data (for example, from GML to GeoJSON/JSON-FG).
- When reducing the amount of data published is needed in the service (and the original source does not offer a mechanism for that).
- When for performance reasons the data needs to be close to a service (for example deploying a fusion service).

The disadvantages or challenges for façades are the following:

- The volume of data to store is large, especially as there is a requirement to store a wide variety of messages during a long period of time.
- The extra processing consumes time, making the service slower.
- When a request/response façade is built in front of a request/response authoritative source, storing a local copy of the dataset in the façade can bring issues to keep the data updated. In contrast, when working with pub/sub, every data update on the authoritative source would be automatically notified to the façade as new incoming messages.
- Making a pub/sub façade in front of a request/response is not an option, as you just cannot provide the advantages of pub/sub service of informing of real-time changes.

13.1.2. Building Standards-Based APIs Natively on top of SWIM Data

A standards-based API that directly accesses the SWIM data would provide two main benefits:

- Be more performant and more stable, compared to an API façade that sits on top of an existing SWIM service and translates API requests to SWIM service requests (like an API façade that talks to a WFS at the backend), because of the additional processing and transformations.
- Reduce the computational complexity and resource requirements compared to an API façade that subscribes to pub/sub SWIM services and build a local datastore that is used to respond to API requests (which is what the NOTAM API does).

13.2. Implementing Standards on Aviation Data APIs

During TB-17, the participants developed eleven APIs serving aviation data, including the ones used for the façades and the fusion services. The original objective for the testbed was to create and implement a new Aviation API standard. However, early into the testbed there was consensus on not following this path and instead developing APIs implementing existing OGC API standards.

This approach proved successful, as all participants were able to fulfill the intended use of their APIs through the implementation of existing OGC API standards. However, this does not necessarily mean that these standards are optimal for the explored use cases.

13.2.1. Creating a new API Standard for Aviation Data

The implementation of new standards is a long process that requires consensus and involvement of the community. Once approved, the standard needs to be promoted and developers need documentation and support to understand and adapt the standard.

The standard will not have credibility and will not reach mass adoption until uptake reaches a tipping point, with enough functional implementations available. If the standard serves a clear use case, with real-world demand, that cannot be served using existing standards, then it will naturally follow that path and eventually be adopted. If the use case is not clear or if it can be served using existing standards, that will lead to community fragmentation or to lack of adoption for the new standard.

Before any attempt to define a new OGC API standard, its scope should be clearly — and narrowly – defined with specific use cases with real-world demand that cannot be served using existing APIs. A market need for such a standard and why it is feasible to develop and promote such a standard should be clearly stated.

Further, it is unclear whether standards-based APIs can be developed for certain use cases. From experience with aeronautical data in TB-17 it seems as if different AIXM datasets follow different conventions. So, a NOTAM API from FAA will likely have to be different from an API from another NOTAM distributor.

13.2.2. Implementing Existing OGC API Standards to Serve Aviation Data

As there is no reason to re-invent standards, for the geospatial aspects of aviation data, leveraging the relevant OGC API building blocks is a way forward. For other aspects, the best solution depends on the community and ecosystem as to whether to leverage the non-geospatial OGC API building blocks. The experiments carried out during the Testbed demonstrated that current OGC API standards (in particular, the Features and Processes APIs) can indeed support the exchange of aeronautical data, flight data, and fused data.

Using existing standards have a huge benefit in using readily available applications (clients and servers) which provides out-of-the-box functionality that has proven useful in many other

use cases and that has been tuned by a long iterative process. An example is the rich filtering capabilities provided by OGC API — Features with respect to searching for flight plans according to a variety of criteria, or searching for flight restrictions based on a geometry that represents a flight path. A new API would take a long time before it could have these capabilities agreed and completed.

Using existing standards is the most time efficient strategy for the adoption and implementation of standards by the community. This approach benefits from having existing documentation, tutorials and examples, existing experienced developer basis and existing ecosystem of functional solutions. New synergies come to play, as the use cases of existing standard expands, increasing the usage, communities, documentation and ecosystem of the existing standard.

13.2.2.1. Comparing Existing OGC API Standards for Fused Data

The two fusion components developed during the testbed implemented different OGC API standards: D106 implemented OGC API — Features, while D107 implemented OGC API – Processes. These fusion components demonstrated that both standards can be used in fusion scenarios. Each of these approaches had its advantages and disadvantages.

Despite implementing different API standards, both fusion components were developed to **fuse data on demand**, meaning every time the API was accessed by a consumer a backend process requested data from different sources, fused the data and relayed it to the consumer. Another valid approach would have been to have the components request data, fuse it and store it in a database for the component to access it whenever the API received a request. The benefits of fusing on demand were the elimination of the need of a large database, the flexibility of changing the fusion process without needing to update a database full of fused data, and the guarantee that the API would always provide updated data. The main disadvantage found of fusing on demand was, for both fusion components, the response time: D106 found the fusion process inefficient (but manageable for the use case implemented), while D107 found computation time was a limitation.

In the case of D106, the rationale for using OGC API — Features was that this standard was found to be more straightforward and less complex since implementing OGC API – Processes would have made the consumers of D106 deal with the process/execution/job endpoints and its logic. Using the OGC API — Features standard has the advantage that the fused data are presented as ‘data’, therefore they can be filtered in a convenient way. The disadvantage is that the data shall be available even when no filter is specified. As D106 is supporting parameterless access to data as well, the amount of data to be processed is crucial: the API supports the limit parameter with a default value of 10, so the default data was found to be acceptable; on the contrary, in case a user enters a large number for the limit parameter, the service would not be able to handle it in a reasonable amount of time. The storage of data would solve this performance problem, but on the other hand it would require a huge amount of storage. Furthermore, for the use case of D106 and because of limitations on one of the data sources, it would have been easier to require the consumers of the API to enter the flight number (GUFI) for each request, although this would have not followed the OGC API — Features Standard.

In the case of D107, the rationale for using OGC API — Processes was that the component would not be providing features that are readily available in an existing dataset but rather launching a process to fuse features among two datasets, if both exist. API consumers are

required to include a search parameter, something deemed necessary for the proper functioning of the fusion process as well as compliant with the standard. A small inconvenience found was that even though the input data was simplified to the minimum, it still required a nested JSON object due to the requirements of the OGC API — Processes Standard.

13.2.3. Additional Considerations for APIs Serving Aviation Data

13.2.3.1. Use of Encodings. The Value of JSON-FG

The Testbed-17 Features and Geometries Task worked on a new encoding based on GeoJSON named JSON-FG (Features & Geometries). This new encoding was widely used in the Aviation Task components, with many positive lessons learned from all participants. For more information on JSON-FG, please refer to TB-17 The Features and Geometries ER.

While in TB-16 some data was transferred with XML-based encodings, for TB-17 all XML-based formats were transformed to JSON-based ones. Due to the slow but steady, natural trend by the web developer community in adopting JSON and gradually abandoning XML in Web APIs, it seems natural for AIXM and FIXM formats to start evolving towards JSON-based encodings. The Aviation task tested with GeoJSON (a very well established standard) and JSON-FG (a draft extension of GeoJSON, that serves a list of use cases that were not properly covered by GeoJSON). Both encodings can work in aviation domains, but the additional capabilities of JSON-FG (such as semantic enrichment, temporal extents, schema definition, or new geometries definition) seem interesting for the domain.

GeoJSON and JSON-FG provide important benefits with regards to XML-based encodings:

- They are more human readable;
- They provide a standard way to store geometries, which makes their features easily displayable on a map and provides useful features based on the availability of the geometries, such as bounding box filtering or geography-based fusion services;
- JSON encodings are typically extensible with additional properties while XML encodings typically require the design of specific extension points as part of the encoding. This is directly reflected in the JSON Schema and XML Schema language designs — and it is, therefore, much easier to extend or customize JSON representations to address additional information needs of a data provider; and finally
- The popularity of JSON encodings is growing while the one of XML encodings is decreasing, so it is just a matter of time that these encodings should be adopted.

JSON-FG provides more benefits than GeoJSON in the aviation domain for the following reasons:

- Provides a standard way of encoding temporal extents (intervals or instants), facilitating the datetime filtering implementation in APIs. This has been proved important in the case of flight plans.
- Provides a standard way of extending the allowed geometries and, for example, encode volumes which is not supported in GeoJSON. Volumes are important to encode airspaces or to model restrictions, obstacles or operational volumes for UAS, or to provide fusion services that require volume information, to name a few.
- JSON-FG also is clearer when geometries are not encoded in WGS 84 lon/lat due to the explicit support for other coordinate reference systems. However, the scenarios used in the testbed did not really have a need to use other coordinate reference systems.
- Provides three different ways to add semantic enrichment to features: the featureType member, anchoring it to a JSON-LD vocabulary definition (by mapping the “featureType” member to “@type”), or using JSON schemas.
- Provides recommendations as to how to link to external features.

JSON-FG leverages the benefits of JSON, in particular the extensibility mentioned above. This can help to reduce the risk that aviation data providers have to form their own per-provider standards because each provider can adopt and extend standardized JSON-FG-based feature encodings to fulfill their own needs. At the same time, JSON-FG has explicit support for key concepts that have been important for aviation standards such as AIXM or FIXM, for example, the notion of feature types or time as a “first-class citizen”.

The main disadvantage of JSON-FG in relation to GeoJSON is that the former is a new draft standard and will take some time to stabilize and gain adoption. However, this disadvantage is greatly diminished due to the fact that JSON-FG is backward compatible, meaning that existing GeoJSON solutions can understand it and interpret it as GeoJSON.

13.2.3.2. Semantic Enrichment

Due to the lack of a referenceable AIXM vocabulary or ontology, it was not possible to enrich the JSON aeronautical data with semantic annotations using JSON-LD contexts. At the same time, none of the clients was expecting such information, so this was not really an issue in this testbed.

During the first stages of the project, an exploration of using JSON-LD with GeoJSON for flight data was carried out. Schema.org was used as the basis for the vocabulary definition as it had predefined types, such as *airport* or *flight*, that could be reused.

Mixing JSON-LD with GeoJSON and the additional OGC API links information and trying JSON-LD validators to accept the implementation was troublesome. Different implementations were tested and none of them could be validated. The workaround was to create the `featureType` member in JSON-FG that could be mapped to JSON-LD “@type” if necessary. The result was

not a 100% compatible JSON-LD document, but it should be enough to make use of JSON-LD vocabularies in a reasonable manner.

The experience with schema.org was mixed. The fact that useful types were standardized and inherited valuable properties (for example, Airport inherited from Place, which inherited from Thing) provided a useful and clear way of structuring the properties in GeoJSON/JSON-FG. They also provided a clear way of linking to external features, adding semantic information on the linked type. However, several important types and properties were missing and documentation was unclear as to how to extend it. The structure of geographical information in schema.org is different compared with GeoJSON. Some properties were unnecessarily complex while others were too simple.

Semantic enrichment could see a significant value for data fusion scenarios. This was reflected when the Flight Restrictions Data Fusion Component requested the same flight data from two different sources: the lack of semantically-enriched data forced the developers of the component to manually map the variables of both sources into a common structure.

The recommendations with regards to semantic enrichment are:

- Use the `featureType` member in JSON-FG to identify the generic semantic type of the feature.
- Use JSON-LD context and map it to `featureType` if there are existing vocabularies that you want to reuse.
- Define JSON schemas to describe your features and your collections.
- Insert links to features in other collections when possible and convenient.
- If consensus in the industry can be reached, work on defining common ontologies, vocabularies and JSON schemas that can be reused. A schema.org approach is recommended, by defining generic types first and creating new ones by inheriting and existing previous ones. JSON schemas can follow this logic, as `featureTypes` can conform to several schemas simultaneously.

14

RECOMMENDATIONS AND FUTURE WORK

14 RECOMMENDATIONS AND FUTURE WORK

Testbed-17 expanded the TB-16 API interoperability explorations by developing and testing eleven APIs based on OGC API Standards, two aviation data fusion scenarios, and two aviation client perspectives. Future work should build upon the findings that emerged from the development and testing of these components, and answer questions that were left out of scope.

14.1. Continue Evaluating the Potential of OGC API Standards in the Exchange of SWIM Data

During TB-17, the APIs developed demonstrated the use of implementing OGC API Standards with the purpose of serving aeronautical structures, airspaces, and flight data. However, the scenarios explored in this testbed do not encompass the complete extent of possibilities found in the SWIM System. Future work should be coordinated with the Aviation API SWG to continue evaluating OGC API Standards in the exchange of aviation SWIM data.

One of the data types left out of scope for this testbed was weather data. This type of data could present challenges that differ from those found while working with aeronautical and flight data; factors like data volume. Future work should explore the potential of standards-based APIs in the exchange of weather data as well. An example of such an API standard is the OGC API — Environmental Data Retrieval (EDR) standard which has been implemented by the US National Weather Service and the UK Met Office.

APIs could also be explored for aviation data with diverse levels of volume or update frequency, as diverse requirements might emerge. Additional exploration could be done regarding serving data on-demand (like this Testbed's fusion components) or storing it in databases (like this Testbed's façades).

14.2. Explore Additional Aviation Data Fusion Scenarios

The two fusion components developed during TB-17 demonstrated the implementation of two OGC API Standards (Features and Processes) in the scenarios for generating and serving aviation fused data. Each component had its own characteristics, operation, and challenges associated. Future work should explore additional data fusion scenarios within the aviation domain to continue gathering lessons learned for further development of API standards, and for proposing new aviation fusion services. The usage of both standards-based APIs and tailored APIs for fusion scenarios should be considered. Decision-making rules could be developed for when to implement certain API standards, when to use real-time data fusion, or when to use certain data structures for fused data.

14.3. Continue Exploring JSON-FG Within the Aviation Domain

JSON-FG proved to be a powerful encoding capability for aeronautical and flight data. However, the scope of data used during the Testbed did not include all possible uses or geometries. Future work should be coordinated with the Features and Geometries JSON SWG to continue exploring the uses of JSON-FG with aviation data to ensure the emerging standard is able to support the entire scope of aviation use cases, as well as more advanced geometries. These explorations could help obtain valuable lessons that support iterating this standard candidate.

14.4. Develop Aviation Vocabularies, Schemas and Ontologies

The work on TB-16 evidenced the issue of the limited amount of aviation vocabularies, schemas and ontologies. While some APIs developed during TB-17 did implement semantics into the data they offered, many were still limited by this issue. Future work should expand these aviation vocabularies, schemas and ontologies through the exploration of use cases and scenarios that require them.

14.5. Demonstrating the Value of Linked Data in the Aviation Domain

Testbed-16 outlined the need for demonstrating the value of linked data in the aviation domain. The development of JSON-FG during TB-17 opened a new spectrum of possibilities regarding the usage of linked data, and the OGC JSON-FG SWG is working on recommendations regarding the implementation of relationships and links.

For the aviation industry to adopt the usage of linked data, future work should demonstrate its value through the development of aviation clients and fusion scenarios that make use of linked data. Coordinating this work with the OGC JSON-FG SWG could help speed up both the development of the emerging standard but also the implementation of linked data within the aviation domain.

A

ANNEX A (INFORMATIVE) D104 AVIATION APIS: DATA AND EXAMPLE REQUESTS

ANNEX A (INFORMATIVE) D104 AVIATION APIS: DATA AND EXAMPLE REQUESTS

A.1. Airspaces: Data and Example Requests

A.1.1. General OGC API resources

- Landing Page
- Conformance Declaration
- OpenAPI 3.0 Definition in JSON
- API documentation and simple HTML client (using Swagger UI)
- Available data

A.1.2. Available Data

Each airspace is available twice via the API.

All airspaces are included in the feature collection All Controlled Airspaces.

In addition, each airspace is also accessible via a feature collection for each class (B, C, D, E2, E3, E4, E5, E6, E7), for example, the Class B Airspaces.

For each feature collection, the following additional information is available:

- the JSON Schema of the GeoJSON features returned by the API (sub-resource `feature schema`)
- the JSON Schema of the GeoJSON feature collections returned by the API (sub-resource `feature collection schema`)

- the JSON Schema listing the feature properties that can be used to filter the data (sub-resource `queryables`)

A.1.3. Fetching Data

Features are accessed via the sub-resource `items`. This request fetches the features in the collection.

By default, only 10 features are returned and the response includes a `next` link to the next page with data. This supports paged access to all available data. The number of features matching the request and the number of features returned are included in the response in the JSON members `numberMatched` and `numberReturned`.

To retrieve more features per page, use the query parameter `limit`, for example, `limit=1000`.

A.1.4. Filtering Data

Simple filtering of the data can be achieved using query parameters.

- For spatial filtering, the `bbox` query parameter can be used. Note that currently bounding boxes that span the anti-meridian are not yet supported. The following example selects Class B airspaces in the Washington area: https://t17.ldproxy.net/airspace/collections/class_b/items?bbox=-77.6,38.4,-76.2,39.6.

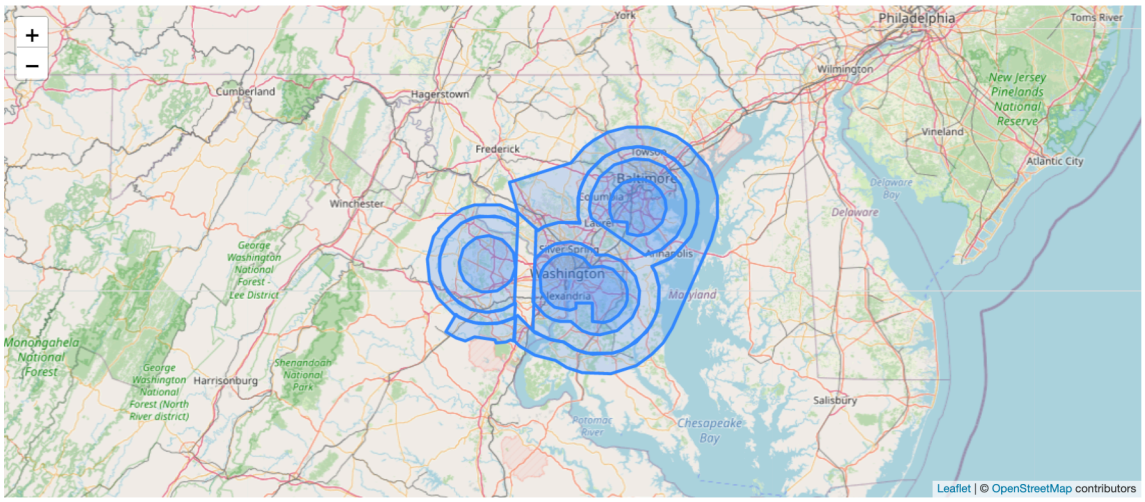
Home / Airspaces / Data / Class B Airspaces / Items GeoJSON | JSON-FG

Class B Airspaces

Generally, that airspace from the surface to 10,000 feet MSL surrounding the nation's busiest airports in terms of IFR operations or passenger enplanements. The configuration of each Class B airspace area is individually tailored and consists of a surface area and two or more layers (some Class B airspace areas resemble upside-down wedding cakes), and is designed to contain all published instrument procedures once an aircraft enters the airspace. An ATC clearance is required for all aircraft to operate in the area, and all aircraft that are so cleared receive separation services within the airspace. The cloud clearance requirement for VFR operations is "clear of clouds".

Filter Edit bbox=-77.60,38.40,-76.20,39.60

« ‹ 1 2 › »



« ‹ 1 2 › »

WASHINGTON-TRI AREA CLASS B

Id	4741
Airspace Identifier	DCA
Name	WASHINGTON-TRI AREA CLASS B
Upper Limit Description	To and Including
Upper Limit	10000
Upper Limit Unit	Feet
Upper Limit Reference	Mean Sea Level

Figure A.1 — D104 Airspace API — bbox Filter

- For filtering on attributes that are queryables, a query parameter with the attribute name can be used. For example, `ident=DCA` on the feature collection with all airspaces selects the Washington airspaces.

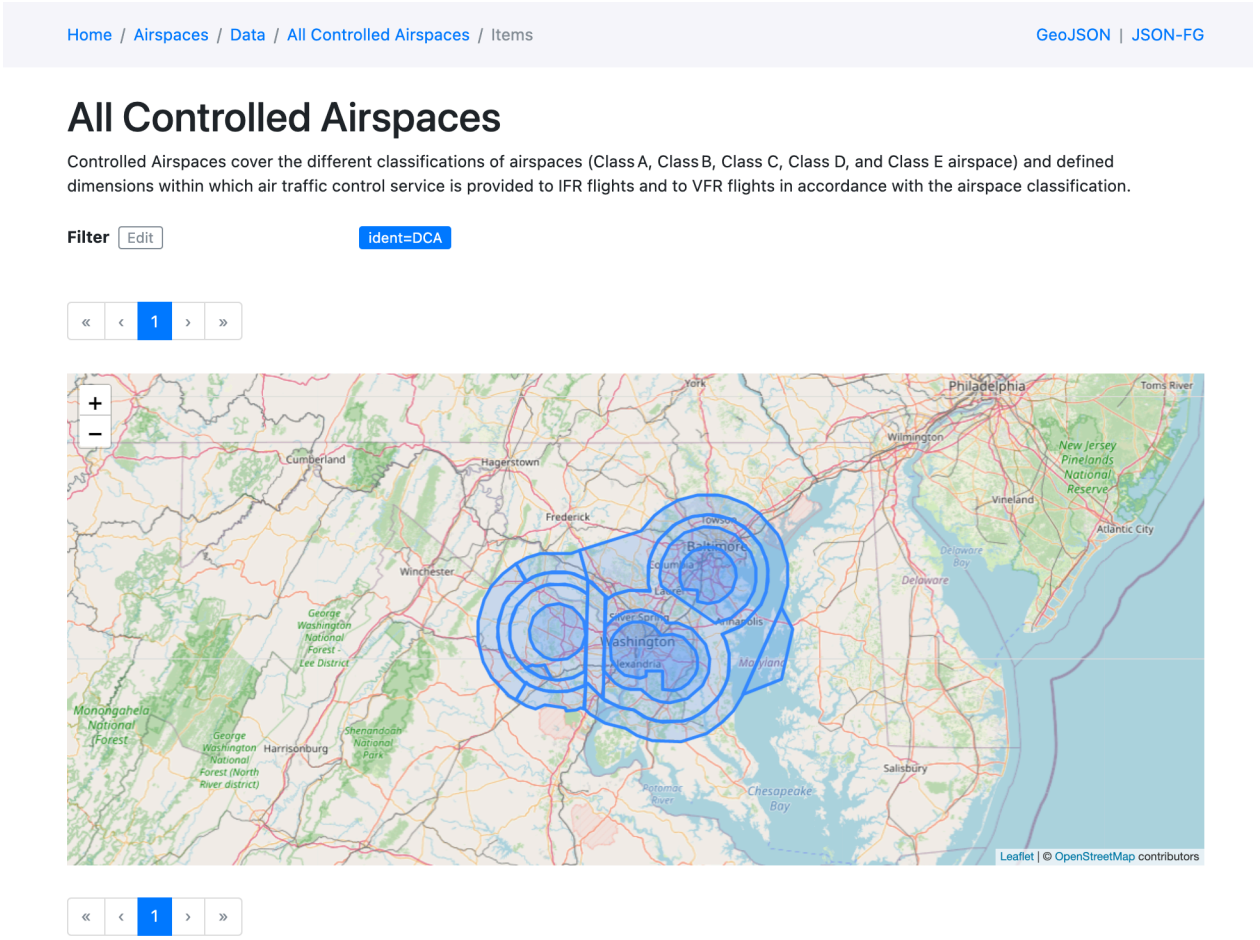


Figure A.2 — D104 Airspace API — attribute Filter

Note that for temporal filtering, the `datetime` query parameter could be used, but the airspace data does not include temporal information.

For more complex filtering, filter expressions using CQL can be used. Two simple examples:

- http://t17.ldproxy.net/airspace/collections/class_all/items?filter=INTERSECTS(geometry%2CPOLYGON((-79.53576165455738%2025.18569533817705%2C%20-79.50845450086926%2025.09155846018916%2C%20-79.50003716505086%2024.993903798755063%2C%20-79.51083312059717%2024.896483025086265%2C%20-79.54042748499097%2024.803040350710415%2C%20-79.58768296340534%2024.

71716672520052%2C%20-79.65078355430599%2024.64216222382644%2C%20-79.7273043373718%2024.580909227512972%2C%20-79.81430466182294%2024.53576165455737%2C%20-79.90844115398109%2024.508454500869263%2C%20-80.00609620124493%2024.50003716505086%2C%20-80.10351697491373%2024.510833120597162%2C%20-80.19695964928958%2024.54042748499097%2C%20-80.28283327479947%2024.58768296340534%2C%20-80.35783777617355%2024.650783554305985%2C%20-80.41909077248702%2024.72730433737181%2C%20-80.46423834544262%2024.81430466182295%2C%20-82.46423834544262%2029.81430466182295%2C%20-82.49154549913074%2029.908441153981084%2C%20-82.49996283494914%2030.006096201244937%2C%20-82.48916687940283%2030.103516974913735%2C%20-82.45957251500903%2030.196959649289585%2C%20-82.41231703659466%2030.28283327479948%2C%20-82.34921644569401%2030.35783777617356%2C%20-82.2726956626282%2030.41909077248702%2C%20-82.18569533817706%2030.46423834544263%2C%20-82.09155884601891%2030.491545499130737%2C%20-81.99390379875507%2030.49996283494914%2C%20-81.89648302508627%2030.489166879402838%2C%20-81.80304035071042%2030.45957251500903%2C%20-81.71716672520051%2030.41231703659466%2C%20-81.64216222382645%2030.349216445694015%2C%20-81.58090922751298%2030.27269566262819%2C%20-81.53576165455738%2030.18569533817705%2C%20-79.53576165455738%2025.18569533817705)))%20AND%20localType%20IN%20(%27CLASS_B%27%2C%27CLASS_C%27%2C%27CLASS_D%27)&limit=100Select all Class B/C/D airspaces along a trajectory in Florida with a buffer

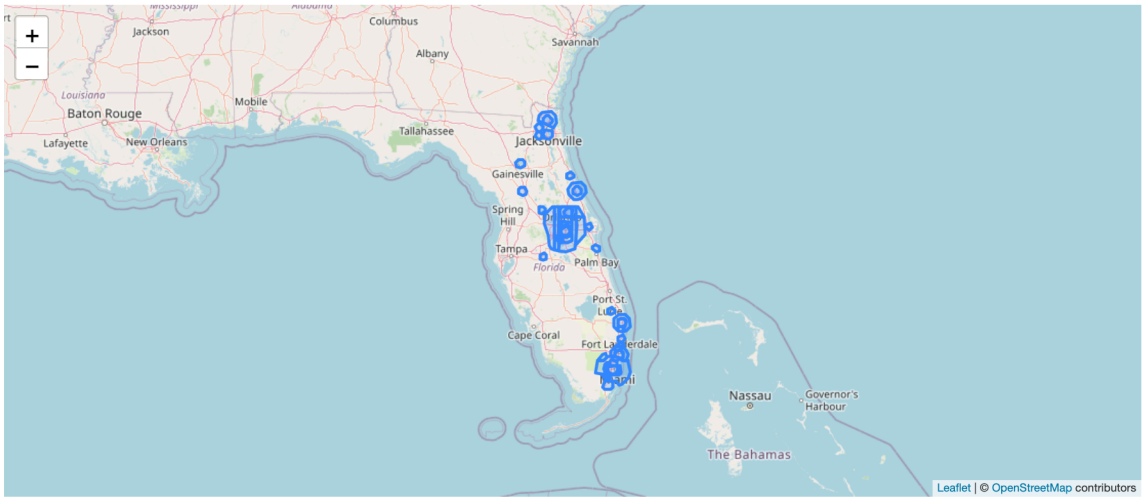
Home / Airspaces / Data / All Controlled Airspaces / Items GeoJSON | JSON-FG

All Controlled Airspaces

Controlled Airspaces cover the different classifications of airspaces (Class A, Class B, Class C, Class D, and Class E airspace) and defined dimensions within which air traffic control service is provided to IFR flights and to VFR flights in accordance with the airspace classification.

Filter Edit

« ‹ 1 › »



« ‹ 1 › »

JACKSONVILLE INTERNATIONAL AIRPORT CLASS C

Id	30
Airspace Identifier	JAX
Name	JACKSONVILLE INTERNATIONAL AIRPORT CLASS C
Upper Limit Description	To and Including
Upper Limit	4000
Upper Limit Unit	Feet
Upper Limit Reference	Mean Sea Level
Lower Limit	0
Lower Limit Unit	Feet

Figure A.3 — D104 Airspace API — Spatial Filter (Corridor)

- Select all airspaces with a lowerLimitValue >= 10000ft and upperLimitValue ⇐ 15000ft.

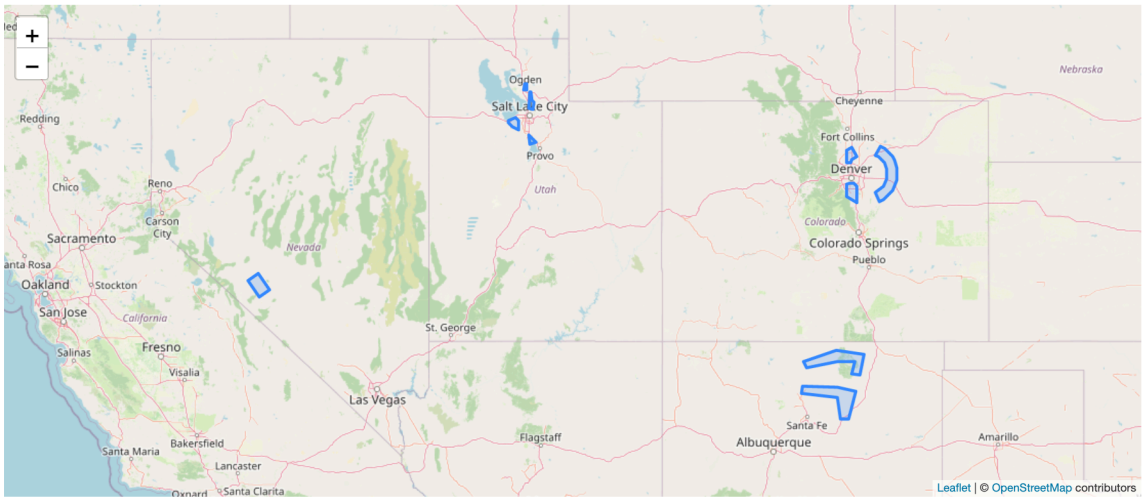
Home / Airspaces / Data / All Controlled Airspaces / Items GeoJSON | JSON-FG

All Controlled Airspaces

Controlled Airspaces cover the different classifications of airspaces (Class A, Class B, Class C, Class D, and Class E airspace) and defined dimensions within which air traffic control service is provided to IFR flights and to VFR flights in accordance with the airspace classification.

Filter Edit

« ‹ 1 2 3 4 5 › »



« ‹ 1 2 3 4 5 › »

SALT LAKE CITY CLASS B

Id	53
Airspace Identifier	SLC
Name	SALT LAKE CITY CLASS B
Upper Limit Description	To and Including
Upper Limit	12000
Upper Limit Unit	Feet
Upper Limit Reference	Mean Sea Level
Lower Limit	10000
Lower Limit Unit	Feet

Figure A.4 — D104 Airspace API — Attribute Filter

A.1.5. Reprojecting Geometries

To retrieve the geometries in another coordinate reference system (CRS), provide the CRS URI in the query parameter `crs`. Example: an airspace in Web Mercator.

A.1.6. Reducing Data

The API also supports to reduce the amount of data that is returned. Two options exist, both are currently based on initial proposals under discussion in the Features API SWG:

- If only a subset of all feature properties is needed, properties including the geometry can be dropped based on this proposal. Example:
    - https://t17.ldproxy.net/airspace/collections/class_all/items?filter=INTERSECTS(geometry%2CPOLYGON((-79.53576165455738%2025.18569533817705%2C%20-79.50845450086926%2025.091558846018916%2C%20-79.50003716505086%2024.993903798755063%2C%20-79.51083312059717%2024.896483025086265%2C%20-79.54042748499097%2024.803040350710415%2C%20-79.58768296340534%2024.71716672520052%2C%20-79.65078355430599%2024.64216222382644%2C%20-79.7273043373718%2024.580909227512972%2C%20-79.81430466182294%2024.53576165455737%2C%20-79.90844115398109%2024.508454500869263%2C%20-80.00609620124493%2024.50003716505086%2C%20-80.10351697491373%2024.510833120597162%2C%20-80.19695964928958%2024.54042748499097%2C%20-80.28283327479947%2024.58768296340534%2C%20-80.35783777617355%2024.650783554305985%2C%20-80.41909077248702%2024.72730433737181%2C%20-80.46423834544262%2024.81430466182295%2C%20-82.46423834544262%2029.81430466182295%2C%20-82.49154549913074%2029.908441153981084%2C%20-82.49996283494914%2030.006096201244937%2C%20-82.48916687940283%2030.103516974913735%2C%20-82.45957251500903%2030.196959649289585%2C%20-82.41231703659466%2030.28283327479948%2C%20-82.34921644569401%2030.35783777617356%2C%20-82.272695626282%2030.419090772487028%2C%20-82.18569533817706%2030.46423834544263%2C%20-82.09155884601891%2030.491545499130737%2C%20-81.99390379875507%2030.49996283494914%2C%20-81.89648302508627%2030.489166879402838%2C%20-81.80304035071042%2030.45957251500903%2C%20-81.71716672520051%2030.41231703659466%2C%20-81.64216222382645%2030.349216445694015%2C%20-81.58090922751298%2030.27269566262819%2C%20-81.53576165455738%2030.18569533817705%2C%20-79.53576165455738%2025.18569533817705)))%20AND%20localType%20IN%20(%27CLASS_B%27%2C%27CLASS_C%27%2C%27CLASS_D%27)&properties=ident&skipGeometry=true&limit=100&f=jsonReturn only the airspace identifier and no geometry when selecting all Class B/C/D airspaces along a trajectory in Florida with a buffer by adding `properties=ident` and `skipGeometry=true`

- Geometry can be simplified to reduce the number of vertices, e.g. for display at small map scales, based on this proposal. Example:
  - South Florida Low airspace without simplification
  - South Florida Low airspace with `maxAllowableOffset=1`

A.1.7. Maps

The API also makes the data available as Mapbox Vector Tiles. Two styles have been configured. For the styling, this FAA map with the same or similar data has been used as a starting point.

- 2D web map using the vector tileset and the Mapbox style

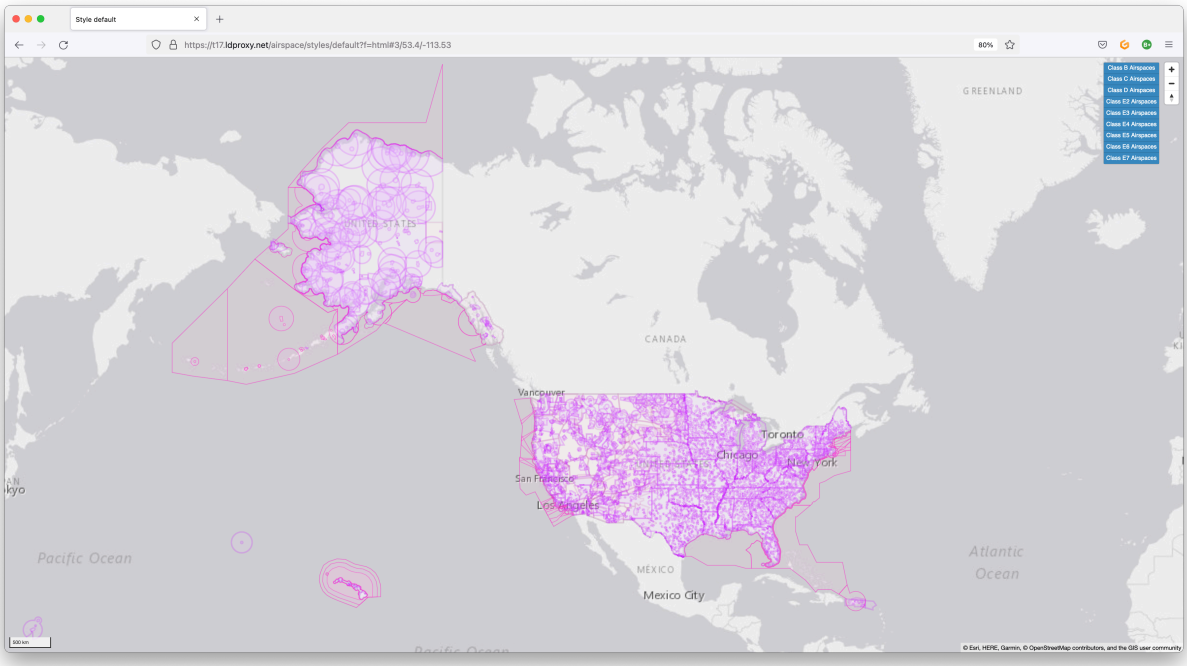


Figure A.5 — D104 Airspace API — 2D map

- Map with the same style, but with Class B/C/D airspaces extruded based on lower/upper limits on larger scales using the vector tileset and the Mapbox style

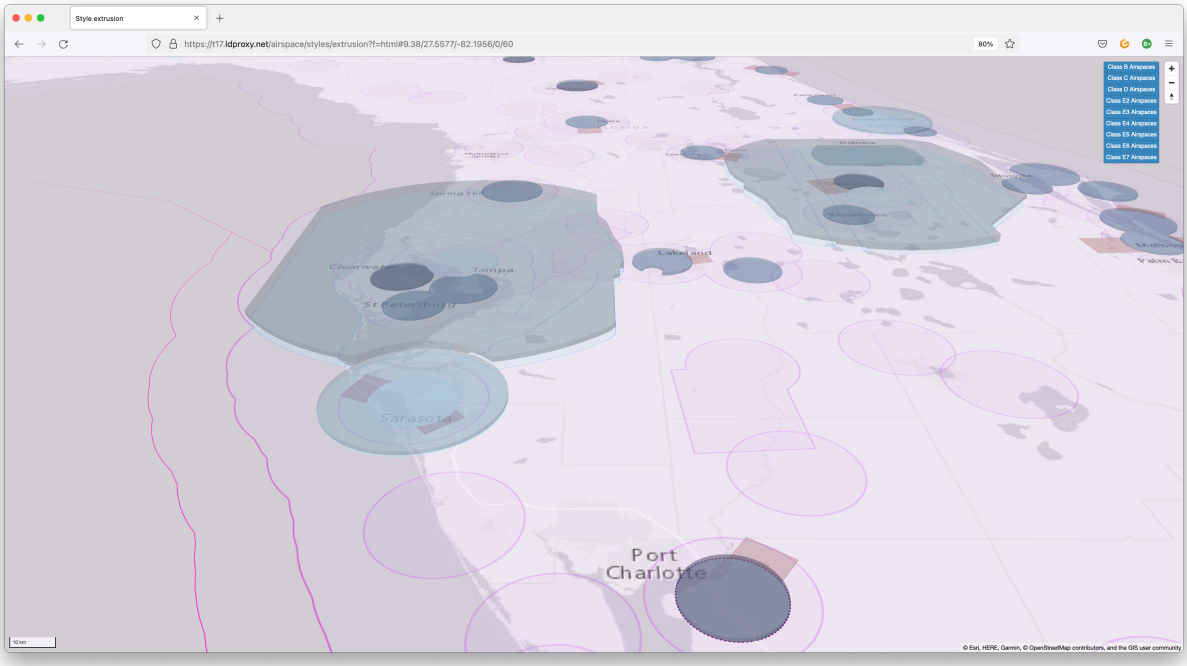


Figure A.6 — D104 Airspace API — map With Extruded Airspaces

A.2. Airports (Organized by AIXM Feature Type): Data and Example Requests

A.2.1. General OGC API Resources

- Landing Page
- Conformance Declaration
- OpenAPI 3.0 Definition in JSON
- API documentation and simple HTML client (using Swagger UI)
- Available feature types

A.2.2. Available Data

Airport data for 82 airports in the U.S. are currently available. The source data is from 2010 and is the AIXM 5.1 format. The data structures have been flattened for simpler use in clients. The following AIXM feature types are published as collections in the API:

- AirportHeliport
- Apron
- ApronElement
- GuidanceLine
- Runway
- RunwayBlastPad
- RunwayCentrelinePoint
- RunwayElement
- RunwayMarking
- SurveyControlPoint
- TaxiHoldingPosition
- Taxiway
- TaxiwayElement
- TaxiwayMarking
- VerticalStructure

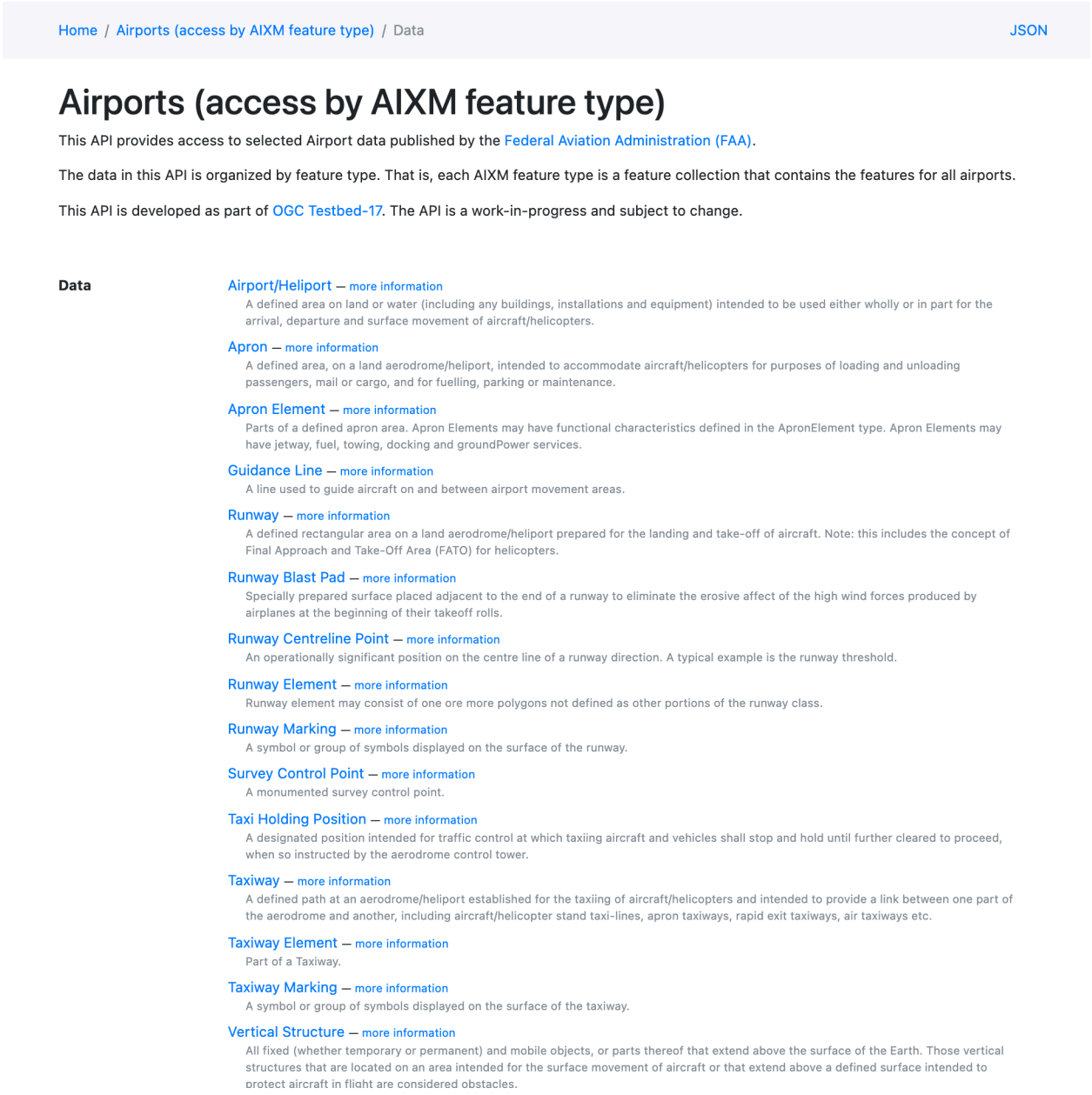


Figure A.7 — D104 Airport API — Available AIXM Feature Types

For each feature collection, the following additional information is available:

- the JSON Schema of the GeoJSON features returned by the API (sub-resource `feature schema`)
- the JSON Schema of the GeoJSON feature collections returned by the API (sub-resource `feature collection schema`)
- the JSON Schema listing the feature properties that can be used to filter the data (sub-resource `queryables`)

A.2.3. Fetching Data

Features are accessed via the sub-resource `items`. This request fetches the features in the collection.

By default, only 10 features are returned and the response includes a `next` link to the next page with data. This supports paged access to all available data. The number of features matching the request and the number of features returned are included in the response in the JSON members `numberMatched` and `numberReturned`.

To retrieve more features per page, use a query parameter `limit`, for example, `limit=1000`.

A.2.4. Filtering Data

To select features from a single airport, add `airport=XYZ` using the airport code, for example, `airport=LAX`.

As in the Airspace API, rich queries can be submitted using filter expressions in CQL.

A.2.5. Reprojecting Geometries

To retrieve the geometries in another coordinate reference system (CRS), provide the CRS URI in the query parameter `crs`. Example: airports in Web Mercator.

A.2.6. Reducing Data

As in the Airspace API, the `properties` and `skipGeometry` parameters can be used to tailor the responses to fields that are needed by the client.

A.2.7. Maps

The API also makes the data available as Vector Tiles. A style has been configured similar to the style used in the AIXM Viewer.

- Map (with LAX as the default location) using the vector tileset and the Mapbox style

Figure A.8 — D104 Airport API — map of LAX

A.3. Airports (Organized by Airport): Data and Example Requests

A.3.1. General OGC API Resources

- Landing Page
- Conformance Declaration
- OpenAPI 3.0 Definition in JSON
- API documentation and simple HTML client (using Swagger UI)
- Available airports

A.3.2. Available Data

Airport data for 82 airports in the U.S. are currently available. The source data is from 2010 and is the AIXM 5.1 format. The data structures have been flattened for simpler use in clients.

For the list of airports see the collections in the API.

For each feature collection, the following additional information is available:

- the JSON Schema of the GeoJSON features returned by the API (sub-resource `feature schema`)
- the JSON Schema of the GeoJSON feature collections returned by the API (sub-resource `feature collection schema`)
- the JSON Schema listing the feature properties that can be used to filter the data (sub-resource `queryables`)

A.3.3. Fetching Data

Features are accessed via the sub-resource `items`. This request fetches the features in the collection.

By default, only 10 features are returned and the response includes a `next` link to the next page with data. This supports paged access to all available data. The number of features matching the request and the number of features returned are included in the response in the JSON members `numberMatched` and `numberReturned`.

To retrieve more features per page, use a query parameter `limit`, for example, `limit=1000`.

A.3.4. Filtering Data

As in the Airspace API, rich queries can be submitted using filter expressions in CQL.

A.3.5. Reprojecting Geometries

To retrieve the geometries in another coordinate reference system (CRS), provide the CRS URI in the query parameter `crs`. Example: LAX airport features in Web Mercator.

A.3.6. Reducing Data

As in the Airspace API, the `properties` and `skipGeometry` parameters can be used to tailor the responses to fields that are needed by the client.

A.4. NOTAMs: Data and Example Requests

A.4.1. General OGC API Resources

- Landing Page
- Conformance Declaration
- OpenAPI 3.0 Definition in JSON
- API documentation and simple HTML client (using Swagger UI)
- Available data

A.4.2. Available Data

The current API includes all NOTAMs collected since mid July 2021 (more than 600000 NOTAMs by mid September 2021, approx. 65000 NOTAMs that are valid at that time).

All NOTAMs are published as a single OGC API collection, i.e. all NOTAMs are included in the feature collection NOTAMs.

The following metadata is available:

- the JSON Schema of the GeoJSON features returned by the API (sub-resource `feature schema`)
- the JSON Schema of the GeoJSON feature collections returned by the API (sub-resource `feature collection schema`)
- the JSON Schema listing the feature properties that can be used to filter the data (sub-resource `queryables`)

A.4.3. Fetching Data

Features are accessed via the sub-resource `items`. This request fetches the features in the collection.

By default, only 10 features are returned and the response includes a `next` link to the next page with data. This supports paged access to all available data. The number of features matching the request and the number of features returned are included in the response in the JSON members `numberMatched` and `numberReturned`.

To retrieve more features per page, use a query parameter `limit`, for example, `limit=100`.

In the HTML output, coded values (e.g., the Q values) are converted to text that is understandable to humans.

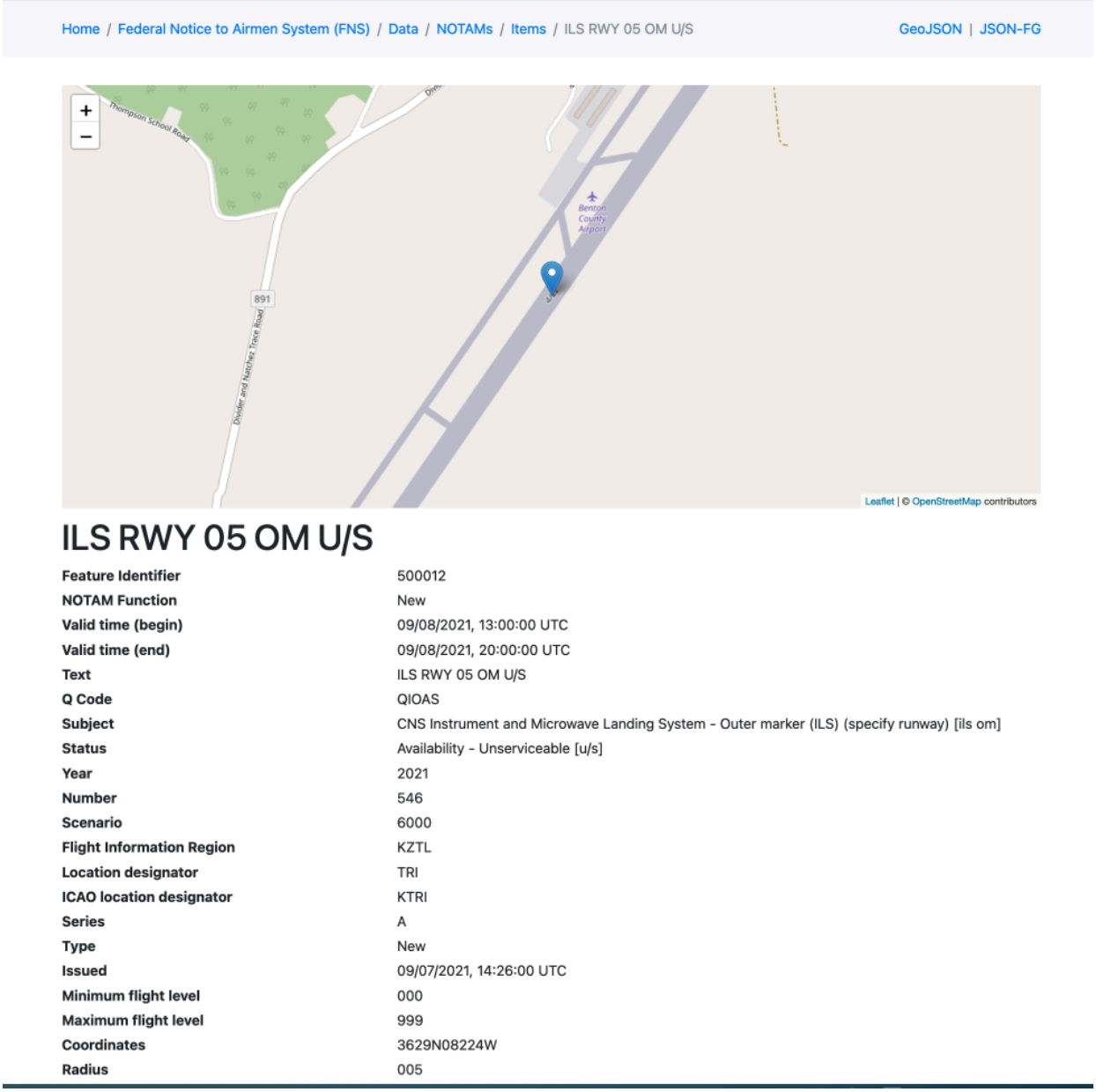


Figure A.9 — D104 NOTAM API — HTML Output of a NOTAM

A.4.4. Filtering Data

Simple filtering of the data can be achieved using query parameters.

- For spatial filtering, the `bbox` query parameter can be used. Note that currently bounding boxes that span the anti-meridian are not yet supported. The following example selects NOTAMs in the Washington area: https://t17.ldproxy.net/fns/collections/notam/items?bbox=-77.6,38.4,-76.2,39.6.

- For temporal filtering, the `datetime` query parameter can be used. This request selects NOTAMs that are effective at 1AM UTC on 2021-06-27: `datetime=2021-06-27T01:00:00Z`. This request selects NOTAMs that are effective during July 2021: `datetime=2021-07-01T00:00:00Z/2021-07-31T23:59:59Z`.
- For filtering on attributes that are queryables, a query parameter with the attribute name can be used. Examples:
- `location=IAD` selects the events related to Dulles airport.
- `notam_keyword=RWY` selects the events related to runways.
- `affected_fir=ZDC` selects the events affecting the Washington Air Route Traffic Control Center (ZDC).
- `icao_location=KIAD` selects the events related to Dulles airport.
- `scenario=82` selects the events related to scenario 82 (NOTE: no information could be found that describes the meaning of the scenario identifiers).

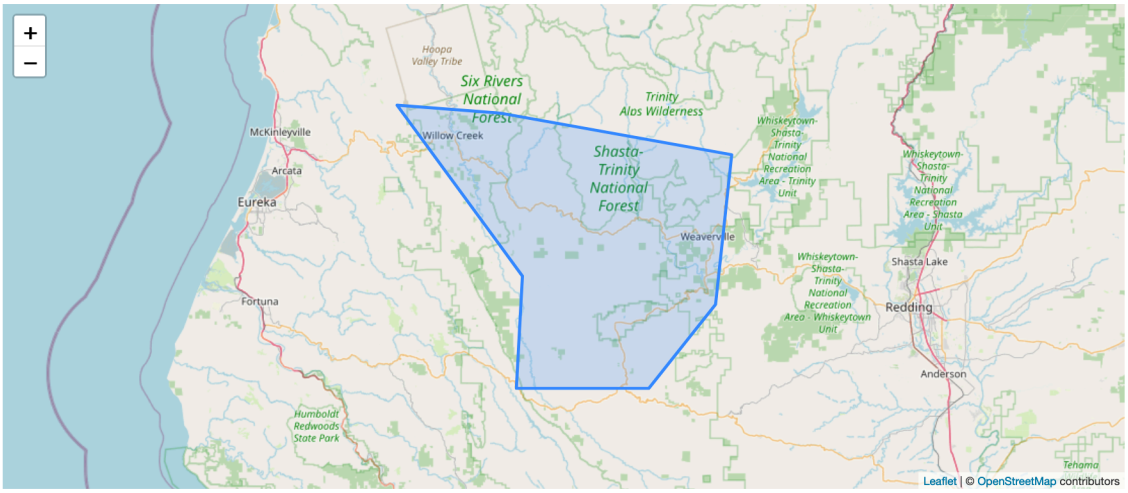
Any of these query parameters can be combined. Events are selected that meet all predicates. For example:

- Select all NOTAMs that are currently in effect, related to runways and affecting the Washington Air Route Traffic Control Center (ZDC).

For more complex filtering, filter expressions using CQL can be used. Examples:

- https://t17.ldproxy.net/fns/collections/notam/items?datetime=now&filter=text+LIKE+%27AIRSPACE+SEE+FDC%25%27Select all NOTAMs that are active Temporary Flight Restrictions

Home / Federal Notice to Airmen System (FNS) / Data / NOTAMs / Items / AIRSPACE SEE FDC 1/5326 ZOA 99.7 SECURITY GeoJSON | JSON-FG



AIRSPACE SEE FDC 1/5326 ZOA 99.7 SECURITY

Feature Identifier	615971
NOTAM Keyword	AIRSPACE
NOTAM Function	New
Valid time (begin)	09/10/2021, 00:30:00 UTC
Valid time (end)	09/30/2021, 04:00:00 UTC
Text	AIRSPACE SEE FDC 1/5326 ZOA 99.7 SECURITY
Temporary Flight Restriction	1/5326
Year	2021
Number	65
Scenario	101
Location designator	F62
Type	New
Issued	09/10/2021, 00:30:00 UTC

powered by ldproxy

Figure A.10 — D104 NOTAM API — a Temporary Flight Restriction NOTAM With a Link to the FAA Page for the Restriction

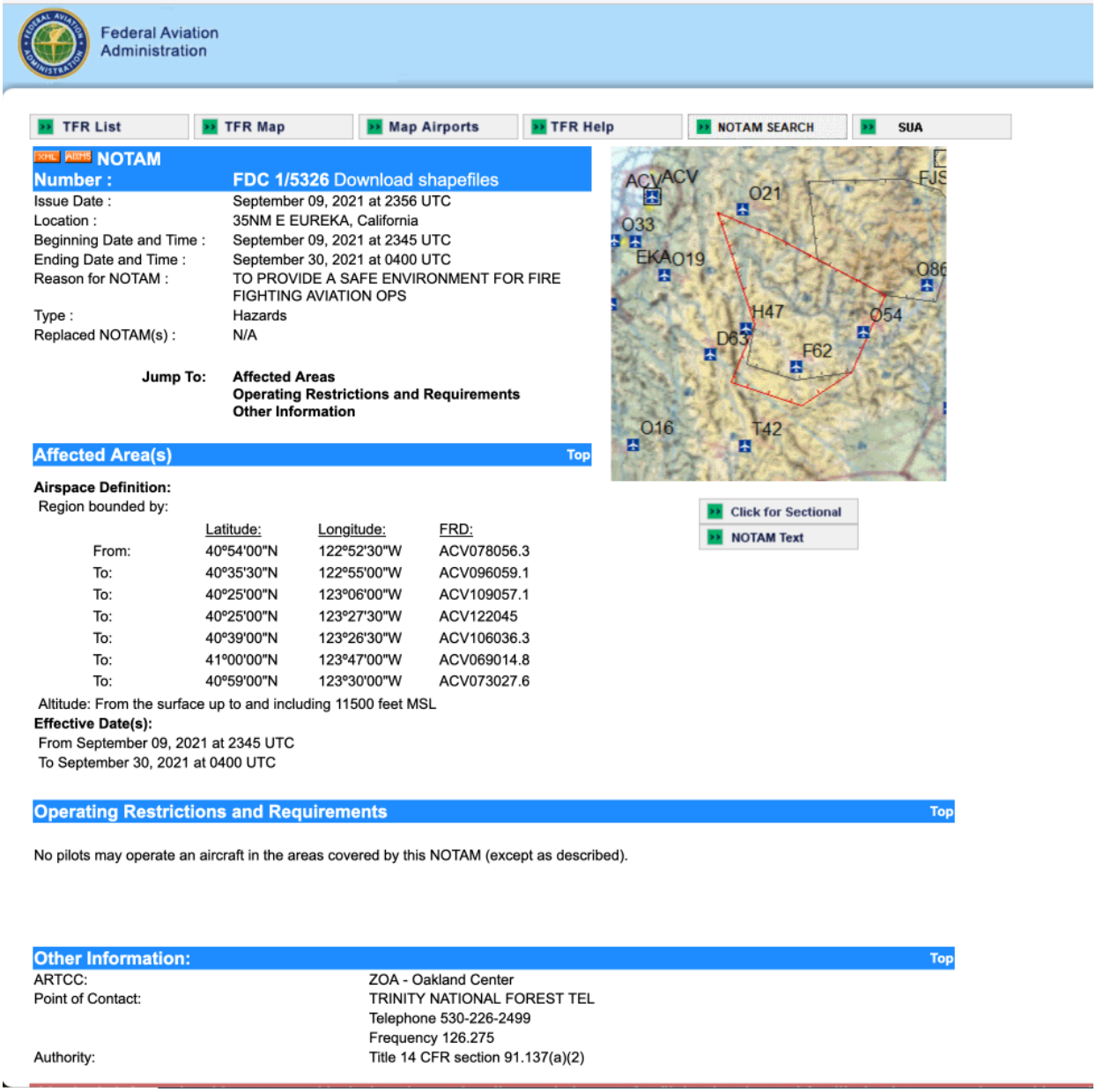


Figure A.11 — The FAA Page for the Restriction

- https://t17.ldproxy.net/fns/collections/notam/items?filter=INTERSECTS(geom%2CPOLYGON((-79.53576165455738%2025.18569533817705%2C%20-79.50845450086926%2025.091558846018916%2C%20-79.50003716505086%2024.993903798755063%2C%20-79.51083312059717%2024.896483025086265%2C%20-79.54042748499097%2024.803040350710415%2C%20-79.58768296340534%2024.71716672520052%2C%20-79.65078355430599%2024.64216222382644%2C%20-79.7273043373718%2024.580909227512972%2C%20-79.81430466182294%2024.53576165455737%2C%20-79.90844115398109%2024.508454500869263%2C%20-80.00609620124493%2024.50003716505086%2C%20-

80.10351697491373%2024.510833120597162%2C%20-80.19695964928958%2024.54042748499097%2C%20-80.28283327479947%2024.58768296340534%2C%20-80.35783777617355%2024.650783554305985%2C%20-80.41909077248702%2024.72730433737181%2C%20-80.46423834544262%2024.81430466182295%2C%20-82.46423834544262%2029.81430466182295%2C%20-82.49154549913074%2029.908441153981084%2C%20-82.49996283494914%2030.006096201244937%2C%20-82.48916687940283%2030.103516974913735%2C%20-82.45957251500903%2030.196959649289585%2C%20-82.41231703659466%2030.28283327479948%2C%20-82.34921644569401%2030.35783777617356%2C%20-82.2726956626282%2030.41909077248702%2C%20-82.18569533817706%2030.46423834544263%2C%20-82.09155884601891%2030.491545499130737%2C%20-81.99390379875507%2030.49996283494914%2C%20-81.89648302508627%2030.489166879402838%2C%20-81.80304035071042%2030.45957251500903%2C%20-81.71716672520051%2030.41231703659466%2C%20-81.64216222382645%2030.349216445694015%2C%20-81.58090922751298%2030.27269566262819%2C%20-81.53576165455738%2030.18569533817705%2C%20-79.53576165455738%2025.18569533817705)))%20AND%20notam_keyword%20IN%20(%27RWY%27%2C%27TWY%27%2C%27AIRSPACE%27)&datetime=2021-06-23T18:00:00Z/2021-06-23T22:00:00Z&limit=20Select all events related to runways/taxiways/airspaces along a trajectory in Florida with a buffer effective between 6PM and 10PM UTC on 2021-06-23

A.4.5. Reprojecting Geometries

To retrieve the geometries in another coordinate reference system (CRS), provide the CRS URI in the query parameter `crs`. Example: an event in Web Mercator.

A.4.6. Reducing Data

The API also supports to reduce the amount of data that is returned. Two options exist, both are currently based on initial proposals under discussion in the Features API SWG:

- If only a subset of all feature properties is needed, properties including the geometry can be dropped based on this proposal. Example:
  - https://t17.ldproxy.net/fns/collections/notam/items?filter=INTERSECTS(geom%2CPOLYGON((-79.53576165455738%2025.18569533817705%2C%20-79.50845450086926%2025.091558846018916%2C%20-79.50003716505086%2024.993903798755063%2C%20-79.51083312059717%2024.

896483025086265%2C%20-79.54042748499097%2024.803040350710415%2C%20-79.58768296340534%2024.71716672520052%2C%20-79.65078355430599%2024.64216222382644%2C%20-79.7273043373718%2024.580909227512972%2C%20-79.81430466182294%2024.53576165455737%2C%20-79.90844115398109%2024.508454500869263%2C%20-80.00609620124493%2024.5000371650508 6%2C%20-80.10351697491373%2024.510833120597162%2C%20-80.19695964928958%2024.54042748499097%2C%20-80.28283327479947%2024.58768296340534%2C%20-80.35783777617355%2024.650783554305985%2C%20-80.41909077248702%2024.72730433737181%2C%20-80.46423834544262%2024.81430466182295%2C%20-82.46423834544262%2029.81430466182295%2C%20-82.49154549913074%2029.908441153981084%2C%20-82.49996283494914%2030.006096201244937%2C%20-82.48916687940283%2030.103516974913735%2C%20-82.45957251500903%2030.196959649289585%2C%20-82.41231703659466%2030.28283327479948%2C%20-82.34921644569401%2030.35783777617356%2C%20-82.272695662628 2%2030.419090772487028%2C%20-82.18569533817706%2030.46423834544263%2C%20-82.09155884601891%2030.491545499130737%2C%20-81.99390379875507%2030.49996283494914%2C%20-81.89648302508627%2030.489166879402838%2C%20-81.80304035071042%2030.45957251500903%2C%20-81.71716672520051%2030.41231703659466%2C%20-81.64216222382645%2030.349216445694015%2C%20-81.58090922751298%2030.27269566262819%2C%20-81.53576165455738%2030.18569533817705%2C%20-79.53576165455738%2025.18569533817705)))%20AND%20notam_keyword%20IN%20(%27RWY%27%2C%27TWY%27%2C%27AIRSPACE%27)&datetime=2021-06-23T18:00:00Z/2021-06-23T22:00:00Z&limit=20&properties=text&skipGeometry=true&f=jsonReturn only the NOTAM text and no geometry when selecting all runways/taxiways/airspaces events along a trajectory in Florida with a buffer by adding `properties=text` and `skipGeometry=true`

- Geometry can be simplified to reduce the number of vertices, e.g. for display at small map scales, based on this proposal.

B

ANNEX B (INFORMATIVE) DELIVERABLES URLS AND DOCUMENTATION

B

ANNEX B (INFORMATIVE) DELIVERABLES URLS AND DOCUMENTATION

The following table includes the endpoints and documentation URLs of the deliverables of the Aviation API Task. Please note some or all endpoints and URLs might not be operational or updated since some of the services they connect to had access granted for a limited period of time.

Table B.1 — Deliverables URLs

TYPE	URL	DOCUMENTATION	NOTES
D104	https://t17.ldproxy.net/airspace	https://t17.ldproxy.net/airspace/api/?f=html	Controlled airspaces
D104	https://t17.ldproxy.net/airports	https://t17.ldproxy.net/airports/api/?f=html	U.S. airport layouts (collections are feature types)
D104	https://t17.ldproxy.net/airports2	https://t17.ldproxy.net/airports2/api/?f=html	U.S. airport layouts (collections are airports)
D104	https://t17.ldproxy.net/fns	https://t17.ldproxy.net/fns/api/?f=html	FAA NOTAMs (data since July 12, 2021)
D105	https://ogctest.mosaicatm.com/	https://ogctest.mosaicatm.com/api-docs	Flight position APIs for STDDS, SFDPS, TFMS — for min="2021-10-12T22:00" max="2021-10-15T20:00"
D107A	https://aviationapi.skymantics.com/faa/collections/faa_flight_plans	https://aviationapi.skymantics.com/faa/openapi?f=html	FAA flight plans (SFDPS) (data since d-7)
D107B	https://aviationapi.skymantics.com/eurocontrol/collections/eu_flight_plans/	https://aviationapi.skymantics.com/eurocontrol/openapi?f=html	EUROCONTROL flight plans (NM B2B PREOPS) (data since d-7)

TYPE	URL	DOCUMENTATION	NOTES
D106	https://ogc.ngaviation.eu/	https://ogc.ngaviation.eu/swagger/index.html	Flight restrictions fusion service
D107	https://aviationapi.skymantics.com/fusion/processes/aviation-fusion	https://aviationapi.skymantics.com/faa/openapi?f=html	Fusion of International Flight Data
D108	https://demo.luciad.com/tb17/		Aviation client
D109	https://ogctest.mosaicatm.com/d109/		Developer client

C

ANNEX C (INFORMATIVE) REVISION HISTORY

C

ANNEX C (INFORMATIVE) REVISION HISTORY

DATE	RELEASE	AUTHOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-05-18	0.1	S. Taleisnik	all	Initial Version
2021-10-07	0.7	S. Taleisnik	all	Draft Engineering Report
2021-11-05	0.85	S. Taleisnik	all	Final Reviews from Participants
2021-11-19	1.0	S. Taleisnik	all	Final Version

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Dictionary of Computer Science — Oxford Quick Reference, (2016).

2. Lóscio, B.F, Calegari, N., Burle, C.: Data on the Web Best Practices. W3C, https://www.w3.org/TR/dwbp/ (2017).

3. Service Facade Pattern, https://www.ibm.com/docs/pt-br/integration-bus/9.0.0?topic=SSMKHH_9.0.0/com.ibm.etools.mft.pattern.sen.doc/sen/sf/overview.htm.

4. SWIM Questions & Answers, https://www.faa.gov/air_traffic/technology/swim/questions_answers/, (2021).

5. Portele, C., Vretanos, P.A., Heazel, C.: OGC API — Features — Part 1: Core. Open Geospatial Consortium, https://docs.opengeospatial.org/is/17-069r3/17-069r3.html (2019).

6. Portele, C., Vretanos, P.A.: OGC API — Features — Part 2: Coordinate Reference Systems by Reference. Open Geospatial Consortium, https://docs.ogc.org/is/18-058/18-058.html (2020).

7. Vretanos, P.A., Portele, C.: OGC API — Features — Part 3: Filtering. Open Geospatial Consortium, http://docs.opengeospatial.org/DRAFTS/19-079.html .

8. Taleisnik, S.: OGC Testbed-16: Aviation Engineering Report. Open Geospatial Consortium, https://docs.ogc.org/per/20-020.html (2021).

9. Vretanos, P.A., Portele, C.: Common Query Language (CQL2).

10. OGC API — Processes. Open Geospatial Consortium, https://ogcapi.ogc.org/processes/ .

11. Masó, J., Jacovella-St-Louis, J.: OGC API — Tiles — Part 1: Core. Open Geospatial Consortium, http://docs.ogc.org/DRAFTS/20-057.html (2019).

12. Portele, C.: OGC API — Styles. Open Geospatial Consortium, http://docs.opengeospatial.org/DRAFTS/20-009.html .