Guia para Integração Técnica ao Projeto

OpenAPI Specifications

utm.yaml

remoteid.yaml

flights.yaml

injection.yaml

versioning.yaml

Chapter 1 — Introduction to BR-UTM

1.1 What Is BR-UTM?

BR-UTM is the **Brazilian UAS Traffic Management** system, operated and overseen by **DECEA** (Departamento de Controle do Espaço Aéreo), the Brazilian Department of Airspace Control, under the Brazilian Air Force.

Its purpose is to safely coordinate the operation of Unmanned Aerial Systems (UAS — drones) in Brazilian low-altitude airspace. As the number of commercial and industrial drone operations grows, it becomes critical to ensure that multiple operators can share the same airspace without conflict, and that public safety is maintained at all times.

A useful analogy: **BR-UTM works like a central bank for drone flight planning**. Just as a central bank does not conduct commerce itself but provides the infrastructure, rules, and oversight that allow banks to operate reliably and interoperably — DECEA does not fly drones, but it provides the infrastructure, protocols, and oversight that allow companies to fly drones safely and in coordination with one another.

1.2 The Role of DECEA

DECEA is responsible for:

- **Defining and maintaining the standards and protocols** that govern how drone operations are planned, coordinated, and tracked.
- **Operating the central services** of the BR-UTM ecosystem, including the Authentication Server, the Discovery and Synchronization Service (DSS), and the Portal UTM.
- **Validating and authorizing** companies that wish to integrate their software systems into the BR-UTM ecosystem.
- **Creating and managing airspace constraints** — restrictions that reflect no-fly zones, ATM (manned aviation) traffic, DASA-sourced airspace reservations, and other regulatory limitations.
- **Overseeing conformance** — monitoring whether drone operations are being conducted within their declared volumes.

DECEA also operates its own USS (UAS Service Supplier) instance, which it uses to inject high-priority constraints and respond to emergency situations in the airspace.

1.3 The Ecosystem — Key Players

The BR-UTM ecosystem has three main categories of participants:

DECEA (the Authority)

The regulator and infrastructure provider. Operates the DSS, Auth Server, Portal UTM, Interface UTM, and DECEA's own USS. Creates constraints. Validates and approves new USSs.

USS — UAS Service Suppliers (Integrated Companies)

Companies that have been validated and authorized by DECEA to operate within the BR-UTM ecosystem. A USS manages its own drone operations: it creates flight plans, coordinates with other USSs, activates flights, tracks conformance in real time, and exposes telemetry.

A USS is a software system that must:

- Communicate with DECEA's DSS to register and discover flight intentions.
- Communicate peer-to-peer with other USSs to share operational details.
- Expose a set of public HTTP APIs so other USSs and DECEA can query flight information.
- Monitor its own drones for conformance and react automatically to airspace changes.

Drone Operators / End Users

The humans or organizations that operate the physical drones. They interact with the USS software (which the company provides) to submit flight plans, receive approvals, and conduct operations. Their interaction is with the USS, not directly with DECEA's APIs.

1.4 BR-UTM Services

The ecosystem exposes several services, all available under the **Sandbox environment** base domain `*.sandbox.brutm.dcta.mil.br`:

Service	URL	Description
Portal UTM	`http://portal.sandbox.brutm.dcta.mil.br/`	Web portal for companies. Manage accounts, API keys, UTM zones, developer documentation, and support.
Interface UTM	`http://interface.sandbox.brutm.dcta.mil.br/`	3D airspace visualization tool. Displays live OIRs, Constraints, ISAs, UTM Zones, and Remote ID telemetry.
API Gateway	`http://api.sandbox.brutm.dcta.mil.br/`	Entry point for all machine-to-machine APIs.
Auth Server	`http://api.sandbox.brutm.dcta.mil.br/token`	OAuth2 token endpoint for acquiring JWT access tokens.
DSS	`http://api.sandbox.brutm.dcta.mil.br/dss`	Discovery and Synchronization Service — the coordination index for airspace operations.
UTM Zones API	`http://api.sandbox.brutm.dcta.mil.br/zonautm`	API for querying the UTM Zones a company is authorized to operate in.

> **Note:** The production environment URLs are separate and are only accessible after a company has completed the homologation process. The old `montreal.icea.decea.mil.br` URLs found in older documentation are **deprecated** and must not be used.

1.5 International Standards

BR-UTM is built on internationally recognized standards:

- **ASTM F3548-21** — Standard Specification for UAS Traffic Management (UTM) UAS Service Supplier (USS) Interoperability. Defines the strategic coordination protocol between USSs and the DSS.
- **ASTM F3411-22A** — Standard Specification for Remote ID and Tracking. Defines how UASs broadcast and share their identity and location.
- **InterUSS Platform** — An open-source implementation of the ASTM standards, upon which DECEA's DSS is based. This ensures the system is fully interoperable with other international UTM implementations.

The OpenAPI contracts that define the exact HTTP interfaces are published by DECEA and are the definitive reference for integration. They are provided in this repository under the `interfaces/` directory.

1.6 Phases of the Project

BR-UTM is being deployed incrementally. The current operational phase is **Phase 1**, which defines:

- The permission level for validated software: **U1**.
- All operational intents in Phase 1 operate at **priority level 0**. No priority differentiation between operators exists yet.
- All flight types (VLOS, EVLOS, BVLOS) are supported in terms of the protocol, but the business and regulatory rules for each are defined by ANAC and ANATEL separately.

Future phases will introduce priority differentiation based on USS scores, use cases (e.g., medical emergency), and other factors.

1.7 What This Guide Covers

This guide is intended for **software engineers and technical teams** at companies wishing to integrate their systems into BR-UTM as a USS. It covers:

- How to onboard your company and obtain API credentials.
- How authentication and authorization work.
- How to plan, register, activate, execute, and close a drone flight.
- What APIs your system must implement and expose.
- The non-functional requirements your system must satisfy.
- How the homologation (validation) process works.

For regulatory compliance (ANAC, ANATEL, SISANT, SARPAS), consult the applicable Brazilian aviation regulations separately — those topics are outside the scope of this guide.

Chapter 2 — Prerequisites and Core Concepts

Before diving into integration, it is essential to understand the foundational concepts and data structures that underpin the entire BR-UTM system. This chapter defines all key terms used throughout the integration guide.

2.1 DSS — Discovery and Synchronization Service

The **DSS** is the central coordination index of the BR-UTM ecosystem. It is operated by DECEA and is the single authoritative source for discovering what is happening in any given volume of airspace.

Critically, the DSS does not store the full details of any operation. It stores only *references* — lightweight records that tell other participants *who* is operating *where* and *when*, and *how to contact them* to get the full details. The actual volume geometry, flight profile, and telemetry are stored by each USS on their own systems and shared peer-to-peer on demand.

The DSS uses **Google S2 geometry cells** (approximately 1 km² each) to index airspace. Because S2 cells are rectangular approximations, the DSS may return references for operations that are geometrically close but do not precisely intersect with your query area. Your USS is responsible for performing the exact 4D intersection calculation locally.

The DSS is based on the InterUSS Platform open-source project, implementing ASTM F3548-21 and ASTM F3411-22A.

Base URL (Sandbox): `http://api.sandbox.brutm.dcta.mil.br/dss`

2.2 USS — UAS Service Supplier

A **USS** is a software system operated by a company that has been validated and authorized by DECEA to operate in the BR-UTM ecosystem. The USS is responsible for:

- Managing the full lifecycle of its customers' drone operations.
- Registering and coordinating those operations with the DSS.
- Communicating directly with other USSs (peer-to-peer) to share operational details.
- Exposing public HTTP endpoints that other USSs and DECEA can call.
- Monitoring drone telemetry and ensuring conformance with declared flight plans.

Every USS must have a publicly accessible base URL (e.g., `https://uss.yourcompany.com/utm`) registered in the DSS as part of every Operational Intent Reference it creates. This URL is used by other USSs and DECEA to contact your system directly.

2.3 OIR — Operational Intent Reference

An **Operational Intent Reference (OIR)** is a record stored in the DSS representing a company's *intention* to conduct a drone operation. It is the primary unit of coordination in the UTM system.

What the DSS stores about an OIR (the *Reference*):

- A unique entity ID (UUID)
- The managing USS identifier (`manager` — the JWT `sub` claim of the creating USS)
- The operational state (`Accepted`, `Activated`, `Nonconforming`, `Contingent`)
- The temporal window (`time_start`, `time_end`) and S2-indexed extents
- The USS's public base URL (`uss_base_url`) for peer-to-peer contact
- The current version number and **OVN (Object Version Number)**
- The subscription ID associated with this operation

What the USS stores about an OIR (the *Details*, not in DSS):

- The full 4D volumes (`volumes` — array of `Volume4D`)
- Off-nominal volumes (`off_nominal_volumes`) for emergency states
- Priority (currently always `0`)
- Flight type (`VLOS`, `EVLOS`, or `BVLOS`)

Other USSs retrieve these details by calling your USS directly at `GET /uss/v1/operational_intents/{entityid}`.

OIR States

State	Description

Accepted	The flight plan has been created and registered in the DSS. Pre-flight planning phase. No drone is in the air yet.
Activated	The flight is actively underway. The drone is (or is about to be) airborne. Telemetry must be available.
Nonconforming	The drone has temporarily left its declared flight volume. The situation is considered recoverable. The OIR includes `off_nominal_volumes`. Can return to `Activated`.
Contingent	The drone has been outside its declared volume for more than 60 seconds. The situation is considered unrecoverable. Only `off_nominal_volumes` are active. Must eventually be closed.

2.4 OVN — Object Version Number

An **OVN (Object Version Number)** is an opaque token (string) that uniquely identifies the *current version* of a specific Operational Intent or Constraint in the DSS. It changes every time the entity is updated.

OVNs serve a critical purpose in the deconfliction protocol: **they are proof that you have seen and acknowledged the latest state of a neighboring operation or constraint**. Before you can create or update your own OIR, you must collect the OVNs of all nearby OIRs and Constraints (by fetching their details from the respective USSs) and include them in the `key` array of your DSS write request.

If you provide an outdated or missing OVN, the DSS will reject your request with a `409 AirspaceConflictResponse`, listing the entities whose OVNs you are missing.

2.5 Subscription

A **Subscription** is a registration in the DSS that declares your USS's interest in a specific geographic area and time window. When any USS creates, updates, or deletes an OIR or Constraint that intersects your subscribed area, the DSS includes your USS in the `subscribers` list of its write response. The creating/updating USS then calls your USS at `POST /uss/v1/operational_intents` or `POST /uss/v1/constraints` to notify you.

There are two types of subscriptions:

- **Implicit (automatic):** When you create an OIR in the DSS, the DSS can automatically create a subscription for the same area and time window. You achieve this by providing a `new_subscription` object in your OIR creation request. This is the normal operational flow.

- **Explicit (manual):** You can create a standalone subscription using `PUT /dss/v1/subscriptions/{subscriptionid}`. This is useful for systems that need airspace awareness without having active operations — for example, a visualization or situational awareness tool.

2.6 ISA — Identification Service Area

An **ISA (Identification Service Area)** is a record stored in the **Remote ID DSS** (which shares the same DSS infrastructure) that indicates your USS is actively serving telemetry for a given geographic area during a given time window.

The ISA tells other systems (display providers, DECEA's monitoring tools) *where* your USS has active drone operations and *how to query* your telemetry endpoint (`/uss/flights`).

An ISA must be created **at the moment a flight is activated** (when the OIR transitions to `Activated`). It must be deleted when the flight ends and the OIR is deleted.

2.7 Volume4D — The 4D Airspace Volume

A **Volume4D** is the fundamental building block for describing airspace in BR-UTM. It combines a 3D geographic volume with a time window:

Component	Description
outline_polygon	A geographic polygon defined by a list of lat/lng vertices.
outline_circle	Alternatively, a circle defined by a center lat/lng and radius in meters.
altitude_lower	The floor altitude, in meters, WGS84 reference (`"W84"`).
altitude_upper	The ceiling altitude, in meters, WGS84 reference (`"W84"`).
time_start	Start of the time window, in RFC3339 format with UTC timezone (`Z`).
time_end	End of the time window, in RFC3339 format with UTC timezone (`Z`).

A single OIR can contain **multiple Volume4D objects** in its `volumes` array. This allows a complex operation (e.g., a vertical takeoff cylinder + a horizontal route polygon + a landing cylinder) to be described as a single coherent operational intent.

Example: A simple cylindrical volume

```
{
  "volume": {
    "outline_circle": {
      "center": { "lat": -23.2071, "lng": -45.8750 },
      "radius": { "value": 100, "units": "M" }
    },
    "altitude_lower": { "value": 0, "reference": "W84", "units": "M" },
    "altitude_upper": { "value": 120, "reference": "W84", "units": "M" }
  },
  "time_start": { "value": "2026-07-01T10:00:00Z", "format": "RFC3339" },
  "time_end": { "value": "2026-07-01T11:00:00Z", "format": "RFC3339" }
}
```

2.8 Constraint

A **Constraint** is a restriction on airspace usage, created and managed exclusively by DECEA (in Phase 1). Constraints represent no-fly zones, restricted areas, active ATM traffic zones, or reservations sourced from the DASA system (DECEA's airspace reservation tool).

From the USS's perspective, constraints are queried from the DSS just like OIRs. When a constraint exists in your area of interest, you must:

1. Retrieve its OVN from the DSS via `POST /dss/v1/constraint_references/query`.
2. Fetch its full volume details from DECEA's USS at `GET /uss/v1/constraints/{entityid}` (using the `uss_base_url` in the constraint reference).
3. Include its OVN in the `key` array of your OIR creation/update request.
4. Avoid creating operations that intersect active constraints.

Constraints arriving from DASA appear as regular `ConstraintReference` objects — your USS does not need to do anything special to handle them.

2.9 UTM Zone

A **UTM Zone** is a geographic allocation that a company creates in the Portal UTM to define the area where it intends to operate. It is defined by:

- A **geographic polygon** (the operational area)
- An **altitude range** (floor and ceiling)
- An optional **time validity** (can be permanent/indefinite)

All Operational Intents (OIRs) created by a company must be **strictly inside** their UTM Zone(s). A company can own multiple UTM Zones (e.g., for operations in different cities or regions). UTM Zones are permanently allocated — there is currently no renewal or expiration mechanism defined.

The UTM Zone is also the mechanism that links a validated software version (API Key) to a specific geographic area of operation. When a company registers a new UTM Zone using a valid API Key, DECEA can confirm that the company's software has been validated for operation in that area.

2.10 off_nominal_volumes

`off_nominal_volumes` is an array of `Volume4D` objects attached to an OIR that is in a `Nonconforming` or `Contingent` state. These volumes represent the **warning area** — the airspace that DECEA and neighboring USSs should treat as potentially hazardous due to the drone's deviation from its planned route.

- In **Nonconforming** state: both `volumes` (original plan) and `off_nominal_volumes` (warning area) are active.
- In **Contingent** state: only `off_nominal_volumes` are active — the original `volumes` are no longer relevant.

The size and shape of `off_nominal_volumes` is **entirely at the USS's discretion**, calculated based on the drone's current position, velocity, and the nature of the deviation. DECEA has not mandated a specific formula, but the off-nominal volumes should reasonably cover the route between the drone's actual position and the nearest point of the original volume.

2.11 Key Reference Summary

Term	Short Definition
DSS	Central coordination index for airspace. Stores only references.
USS	Your company's integrated software system. Manages flights, exposes APIs.

Term	Short Definition
OIR	An Operational Intent Reference — a flight plan registered in the DSS.
OVN	Object Version Number — proof you've seen the latest state of a neighbor entity.
Subscription	Registration of interest in an airspace area to receive notifications.
ISA	Identification Service Area — declares you're serving telemetry for an area.
Volume4D	3D polygon/circle + altitude range + time window.
Constraint	Airspace restriction created by DECEA. Must be respected by all USSs.
UTM Zone	Your company's authorized area of operation. OIRs must be inside it.
off_nominal_volumes	Warning area declared when a drone deviates from its planned volume.
`manager`	The JWT `sub` claim of the USS that created an OIR. Informational only.
`uss_base_url`	The public HTTPS base URL of your USS. Used for peer-to-peer calls and token audience.

Chapter 3 — Onboarding Process

This chapter describes the end-to-end process for a new company to become an authorized USS in the BR-UTM ecosystem, from account creation to operating in production.

3.1 Overview

The onboarding process has two phases:

1. **Development Phase** — Create an account, obtain a development API Key, and integrate your software against the Sandbox environment.
2. **Production Phase** — Submit your software for homologation (validation) by DECEA, receive a production API Key with U1 permission, create UTM Zones, and begin operating.

```
Create Account → Get Dev API Key → Sandbox Integration → Homologation → Production API Key → UTM Zone → Operate
```

3.2 Step 1 — Create an Account (Contas DECEA)

All access to DECEA digital services starts with a personal account on the **Contas DECEA** platform.

1. Navigate to the **Portal UTM**: http://portal.sandbox.brutm.dcta.mil.br/
2. Select the option to create a new account via **Contas DECEA**.
3. Complete the registration with your personal and professional information.

> **Note:** A single individual user account can be associated with multiple company accounts. The account is linked to the person, not the company.

3.3 Step 2 — Register Your Company

After your personal account is created:

1. Log in to the Portal UTM.
2. Navigate to the company registration section.
3. Register your company with the relevant information (CNPJ, razão social, etc.).

Your company account will be the entity that owns API Keys, UTM Zones, and validated software versions.

3.4 Step 3 — Obtain a Development API Key

With a registered company account, you can request an API Key for the **development (Sandbox) environment** directly through the Portal UTM.

- Navigate to the developer section of the Portal UTM.
- Request a new API Key for your company.
- The API Key will be created and available immediately in the Portal.

> **During the transition period** (while Portal UTM is not yet fully deployed in production): Contact DECEA through the official support channels — **Mattermost** or the **Central de Ajuda** — to request a development API Key manually.

This API Key is used in all subsequent requests to the Auth Server to obtain JWT tokens.

3.5 Step 4 — Download DECEA's Public Key

All JWT tokens issued by DECEA's Auth Server are signed with an RS256 private key. Your USS must validate incoming tokens (from other USSs and from the DSS) using DECEA's public key.

The public key is available through the Portal UTM developer section. It is a standard RSA public key in PEM format.

> **Important:** Your USS must validate every incoming request's JWT against this public key. See Chapter 4 — Authentication for validation details.

3.6 Step 5 — Integrate Against the Sandbox

With your development API Key and DECEA's public key, you can begin implementing and testing your USS software against the **Sandbox environment**.

All Sandbox services are accessible under `*.sandbox.brutm.dcta.mil.br`:

Service	Sandbox URL
Auth Server	`http://api.sandbox.brutm.dcta.mil.br/token`
DSS	`http://api.sandbox.brutm.dcta.mil.br/dss`
UTM Zones	`http://api.sandbox.brutm.dcta.mil.br/zonautm`

Your integration must implement all mandatory USS-side endpoints. The Interface UTM (`http://interface.sandbox.brutm.dcta.mil.br/`) can be used as a visual debugging aid — it shows active OIRs, Constraints, ISAs, and telemetry in a 3D view of Brazilian airspace.

Refer to the following chapters for detailed technical integration guidance:

- Chapter 4 — Authentication
- Chapter 5 — The full flight lifecycle

- Chapter 6 — APIs your USS must implement

3.7 Step 6 — Request Homologation

Once your software is ready and has been validated internally against the Sandbox, you request a **homologation process** with DECEA.

Homologation is a **manual validation process conducted by DECEA**, potentially assisted by internal automated testing tools. During this process, DECEA will exercise your USS's APIs through a set of defined test scenarios (see Chapter 8 — Homologation).

To be eligible for homologation, your USS must:

- Implement all mandatory USS-side endpoints as defined in the OpenAPI specifications.
- Expose the automated testing interfaces (`flights.yaml`, `injection.yaml`, `versioning.yaml`) on your server so DECEA's testing framework can call them.
- Satisfy all non-functional requirements (see Chapter 7).
- Be deployed and accessible from the internet (your system must have a publicly accessible base URL).

Contact DECEA through the official channel (**Mattermost** or the **Central de Ajuda**) to initiate the homologation request.

3.8 Step 7 — Receive a Production API Key (U1 Permission)

Upon successful homologation, DECEA grants your software a **production API Key with the U1 permission level**.

- The production API Key is tied to the specific **version of the software** that was validated. If you release a new major version, a new homologation may be required.
- The U1 permission is the first operational authorization level for Phase 1 of BR-UTM.
- The production API Key can also be used to create sub-keys for **third-party companies** that wish to purchase and use your USS software. As the validated software owner, you

create these sub-keys and distribute them to your clients.

3.9 Step 8 — Create UTM Zone(s)

With a production API Key, your company can create **UTM Zones** in the Portal UTM. A UTM Zone is the geographic, altitudinal, and temporal authorization for your company to operate.

To create a UTM Zone:

1. Log in to the Portal UTM with your production account.
2. Navigate to the **Create UTM Zone** section.
3. Provide your API Key — this confirms your software is validated and authorized.
4. Define the UTM Zone:
   - **Geographic polygon** (vertices in lat/lng)
   - **Altitude range** (floor and ceiling in meters WGS84)
   - **Time validity** (optional — can be indefinite/permanent)
5. Submit for creation.

A company can own **multiple UTM Zones** (e.g., one per city, or one per use case). There is currently no limit on the size of a UTM Zone — it can cover a neighborhood, a city, or an entire region.

> **Remember:** All OIRs (Operational Intent References) your USS creates must be **strictly contained within** one of your UTM Zones. Operations outside your UTM Zone boundaries are not permitted.

3.10 Step 9 — Begin Operating

Once your UTM Zone is active and your production API Key is in use, your USS can begin accepting flight plans from operators and creating OIRs in the production DSS.

The complete technical flow for each individual flight is described in Chapter 5 — The Flight Lifecycle.

3.11 Onboarding Summary

```
┌──────────────────────────────────────────────┐
│              ONBOARDING SEQUENCE             │
├──────────────────────────────────────────────┤
│ 1. Create personal account (Contas DECEA)    │
│ 2. Register your company in Portal UTM       │
│ 3. Request Development API Key (Portal UTM)  │
│ 4. Download DECEA's public key (Portal UTM)  │
│ 5. Implement & test against Sandbox environment │
│ 6. Request Homologation from DECEA           │
│ 7. Pass homologation → receive Production API Key │
│    with U1 permission                        │
│ 8. Create UTM Zone(s) in Portal UTM          │
│ 9. Begin operating in production             │
└──────────────────────────────────────────────┘
```

3.12 Support Channels

Channel	Purpose
Portal UTM	Self-service: API keys, UTM Zones, documentation
Central de Ajuda	General support requests, homologation requests
Mattermost	Real-time developer support and communication with DECEA
Interface UTM	Visual debugging of airspace state in the Sandbox

Chapter 4 — Architecture and Authentication

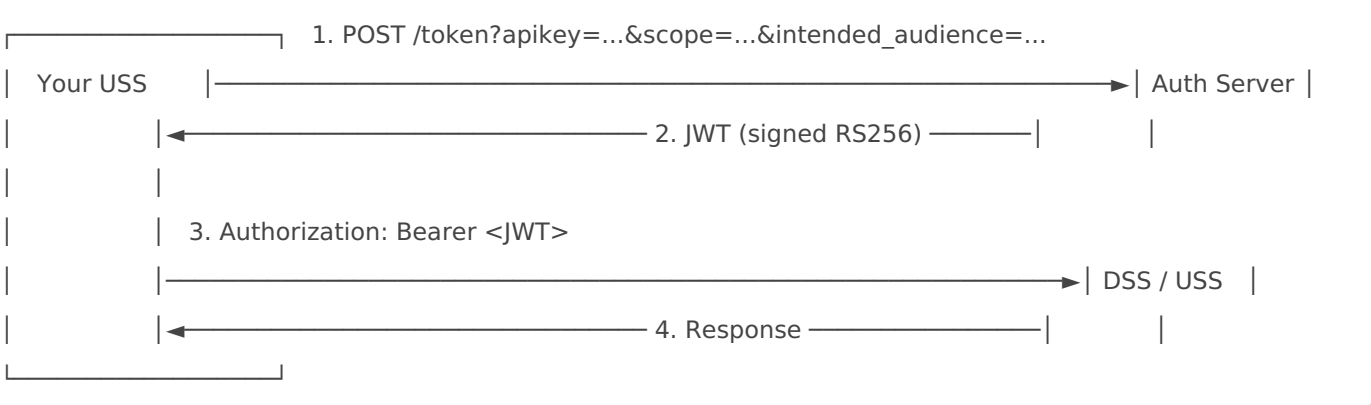
This chapter describes the authentication and authorization model used across the entire BR-UTM ecosystem, including how your USS obtains tokens, uses them to call DECEA's APIs, validates tokens received from other USSs, and authenticates peer-to-peer calls.

4.1 Authentication Architecture Overview

BR-UTM uses **OAuth2 Client Credentials flow** with **JWT (JSON Web Token) access tokens**, signed using **RS256 (RSA SHA-256)**. This is a stateless, distributed authentication model where:

1. Your USS presents its **API Key** to DECEA's **Auth Server** to obtain a signed JWT.
2. The JWT is included as a **Bearer token** in every API request (to the DSS, to DECEA's USS, or to other USSs).
3. The receiving server **validates the JWT locally** using DECEA's public key — it does not need to call the Auth Server again.

This architecture is critical for the distributed, peer-to-peer nature of the system: there is no central session state, and any server can independently validate any token.



4.2 Obtaining a JWT Token

To obtain a JWT, send an HTTP GET or POST request to the Auth Server token endpoint:

Endpoint: `http://api.sandbox.brutm.dcta.mil.br/token`

Query parameters (or request body):

Parameter	Description	Example
`apikey`	Your company's API Key, obtained from the Portal UTM. Can also be sent as an HTTP header.	`abc123...`
`scope`	The OAuth2 scope(s) you need for the operation. Space-separated for multiple scopes.	`utm.strategic_coordination utm.constraint_processing`
`intended_audience`	The FQDN (domain name only, no path) of the server this token will be sent to.	`api.sandbox.brutm.dcta.mil.br` or `uss-b.yourpartner.com`

Example request:

```
GET /token?scope=utm.strategic_coordination&intended_audience=api.sandbox.brutm.dcta.mil.br
Authorization: ApiKey abc123yourapikey
```

Example response:

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

4.3 JWT Token Structure

Every token issued by DECEA's Auth Server is a signed JWT containing the following claims:

Claim	Description	Example
`iss`	The URL of the Auth Server that issued the token.	`http://api.sandbox.brutm.dcta.mil.br/token`

Claim	Description	Example
`exp`	Token expiration timestamp (Unix epoch). Maximum 1 hour from issuance.	`1751327400`
`sub`	Your USS's unique identifier — the `manager` identifier in DSS records.	`"my-company-uss"`
`scope`	Granted scopes, space-separated.	`"utm.strategic_coordination utm.constraint_processing"`
`jti`	Unique token ID for replay protection (RFC 7519).	`"d3e8f921-..."`
`aud`	The FQDN of the intended recipient server.	`"api.sandbox.brutm.dcta.mil.br"`

Token Validity

- Tokens are valid for **up to 1 hour** (inspect the `exp` claim for the exact expiry time).
- **Cache and reuse tokens** until they are near expiry. Requesting a new token on every API call is wasteful and may cause rate limiting.
- A single token can carry **multiple scopes** (space-separated in the `scope` claim), so you can request all the scopes you need for a workflow in one token request.

4.4 Available Scopes

UTM API (`utm.yaml`)

Scope	Purpose
`utm.strategic_coordination`	Create, update, delete, and query OIRs in the DSS. Notify subscriber USSs. Fetch OIR details from other USSs. **Required for all standard flight operations.**
`utm.constraint_processing`	Query constraint references in the DSS and fetch constraint details from DECEA's USS. **Required if your area may have constraints.**
`utm.conformance_monitoring_sa`	Query OIRs and fetch telemetry for off-nominal situations. Used by DECEA's monitoring systems.
`utm.constraint_management`	Create, update, and delete constraints. **DECEA only.**
`utm.availability_arbitration`	Set USS availability state in the DSS. **Not required in Phase 1.**

Scope	Purpose
`utm.aviation_authority`	Access flight authorization details. **DECEA only.**

Remote ID API (`remoteid.yaml`)

Scope	Purpose
`rid.service_provider`	Create, update, and delete ISAs in the Remote ID DSS. **Required when activating/deactivating flights.**
`rid.display_provider`	Query ISAs and telemetry from other USSs. **Required for situational awareness.**

Automated Testing APIs

Scope	Purpose
`interuss.flight_planning.direct_automated_test`	Used by DECEA's test framework when calling your `flights.yaml` endpoints during homologation.
`interuss.flight_planning.plan`	Used by DECEA's test framework to simulate user flight plan actions.
`rid.inject_test_data`	Used by DECEA's test framework when calling your `injection.yaml` endpoints during homologation.
`interuss.versioning.read_system_versions`	Used by DECEA's test framework when calling your `versioning.yaml` endpoint.

Day-to-Day Scope Requirements

For normal production operations, your USS will primarily need:

```
utm.strategic_coordination
utm.constraint_processing
rid.service_provider
rid.display_provider
```

4.5 The `intended_audience` Parameter — Peer-to-Peer Calls

The `intended_audience` parameter (which becomes the `aud` claim in the JWT) is **critical for peer-to-peer security**. It binds a token to a specific recipient, preventing token replay attacks.

Rules:

- When calling **DECEA's DSS** or **Auth Server**: use the domain of the DSS (e.g., `api.sandbox.brutm.dcta.mil.br`).
- When calling **another USS** (e.g., to fetch OIR details): use **only the domain** of that USS's `uss_base_url` — no path, no port (unless non-standard).

Example:

If USS B's `uss_base_url` in the DSS is `https://uss-b.partnercompany.com/api/utm`, then when USS A wants to call USS B:

```
intended_audience = "uss-b.partnercompany.com"
```

USS A requests a fresh token with this audience and includes it in the `Authorization` header when calling `GET https://uss-b.partnercompany.com/api/utm/uss/v1/operational_intents/{id}`.

USS B, upon receiving this request, validates that the token's `aud` claim matches its own domain (`uss-b.partnercompany.com`). If it doesn't match, it rejects the request with HTTP 401.

4.6 Using the Token

Include the token in every outgoing HTTP request:

```
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...
```

This applies to calls to:

- DECEA's DSS (`/dss/v1/...`)
- DECEA's Remote ID DSS (`/dss/identification_service_areas/...`)
- Any peer USS's endpoints (`/uss/v1/...`)

4.7 Validating Incoming Tokens (Inbound Requests)

Your USS **must validate every inbound request** — whether from DECEA's systems or from other USSs. Failing to do so is a security vulnerability and a homologation failure.

The validation steps are:

Step 1 — Extract the Token

> Extract the JWT from the Authorization header:
>
> Authorization: Bearer <token>

Step 2 — Verify the RS256 Signature

Verify the token's digital signature using **DECEA's public RSA key** (obtained from the Portal UTM during onboarding). The signing algorithm is RS256.

If the signature is invalid → reject with **HTTP 401**.

Step 3 — Verify Token Expiry

Check that the `exp` claim is greater than the current UTC timestamp.

If the token is expired → reject with **HTTP 401**.

Step 4 — Verify the Audience

Check that the `aud` claim matches **your own server's FQDN**. This prevents a token intended for another USS from being replayed against your server.

If `aud` doesn't match your domain → reject with **HTTP 401**.

Step 5 — Verify the Scope

Check that the `scope` claim contains the scope required by the specific endpoint being called. Different endpoints require different scopes (see the OpenAPI specifications for each endpoint's required scope).

If the scope is insufficient → reject with **HTTP 403**.

Go implementation reference (from the BR-UTM workshop):

```
func verifyToken(token string, publicKeyFile string) (bool, error) {
    bytes, _ := os.ReadFile(publicKeyFile)
    publicKey, _ := jwt.ParseRSAPublicKeyFromPEM(bytes)
    parts := strings.Split(token, ".")
    err := jwt.SigningMethodRS256.Verify(
        strings.Join(parts[0:2], "."), parts[2], publicKey,
    )
    return err == nil, err
}
```

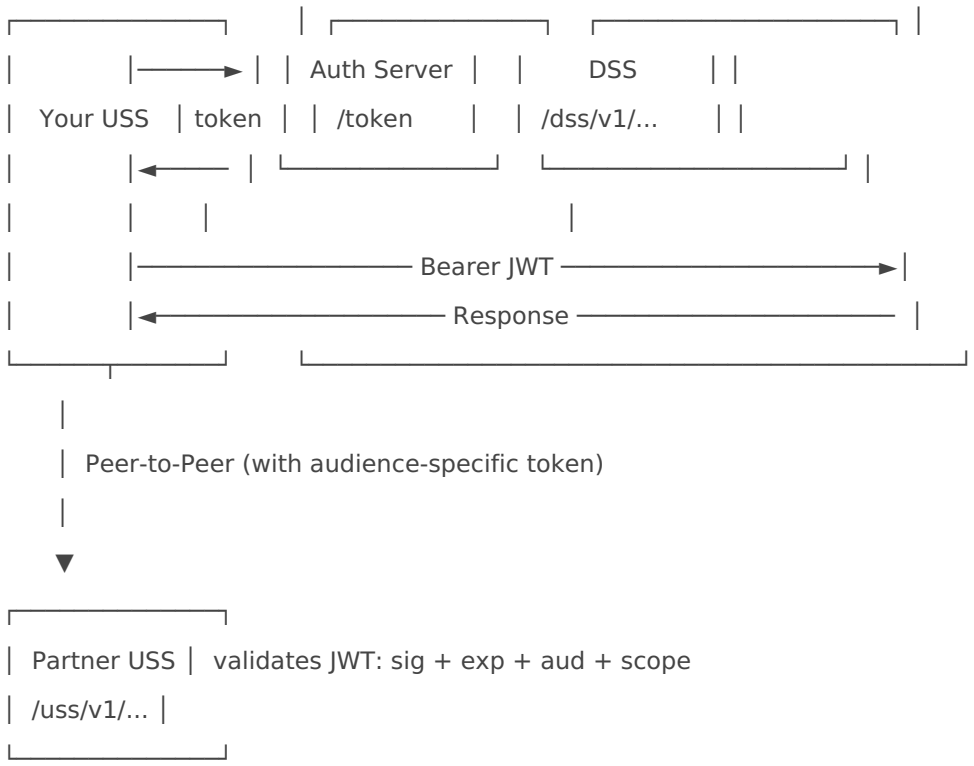
4.8 Token Caching Strategy

For optimal performance:

1. **Cache the token** after obtaining it.
2. **Reuse it** for all requests to the same audience with the same scopes, until it's within a safety margin of expiry (e.g., 60 seconds before `exp`).
3. **Request a new token** when the cached one is near expiry or when you need a different scope/audience combination.

You may maintain multiple cached tokens simultaneously — one per `(scope, audience)` combination that your USS needs.

4.9 Architecture Diagram — Full Authentication Flow



4.10 Security Principles

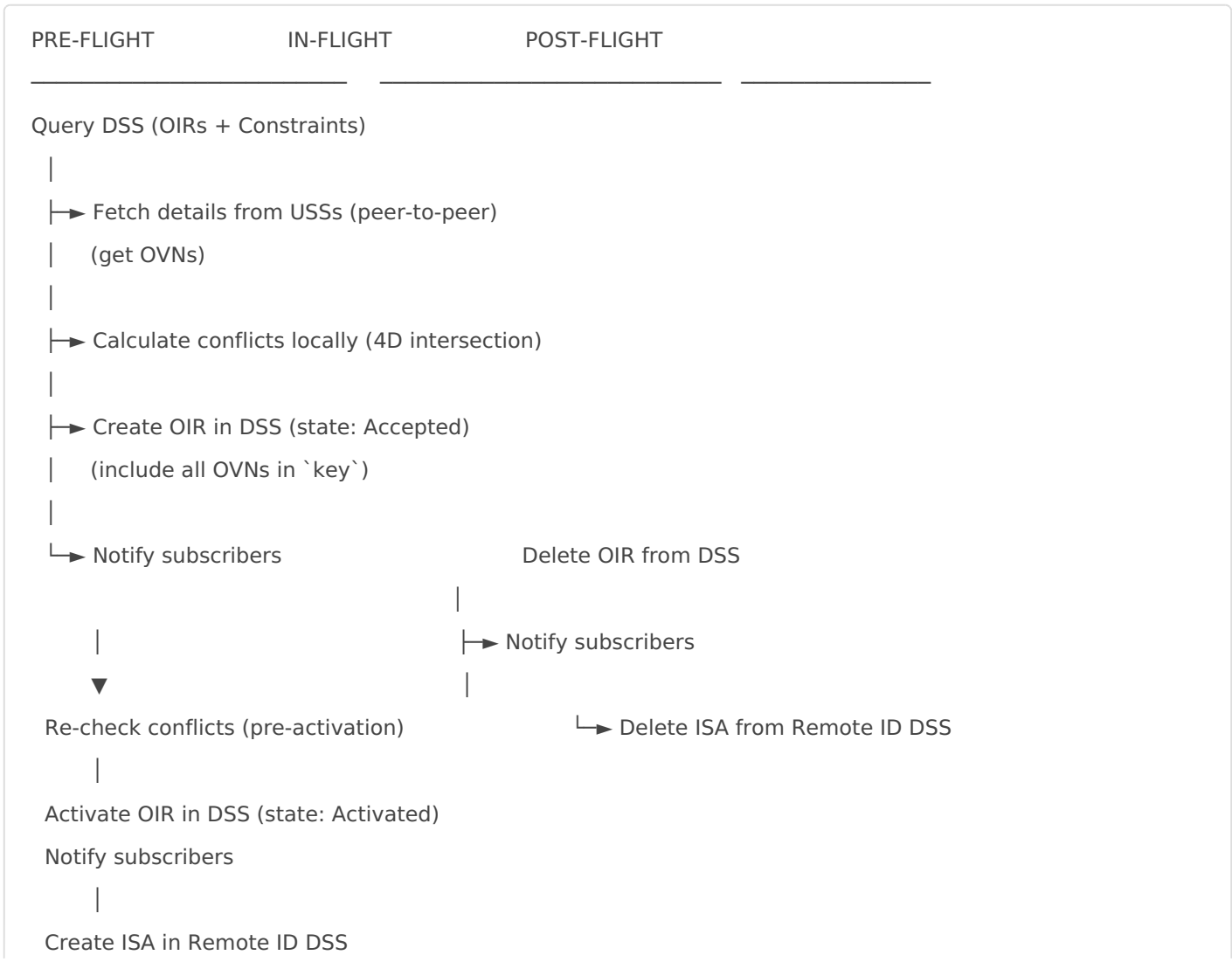
- **Principle of least privilege**: Request only the scopes your operation actually needs. Avoid requesting all scopes in every token.
- **No token sharing**: Tokens are bound to a specific audience (`aud`). A token obtained to talk to the DSS cannot be used to call another USS, and vice versa.
- **Time synchronization**: Your system clock must be synchronized with DECEA's NTP server (`ntp.decea.gov.br`) to ensure token expiry calculations are accurate. See Chapter 7 — Non-Functional Requirements.

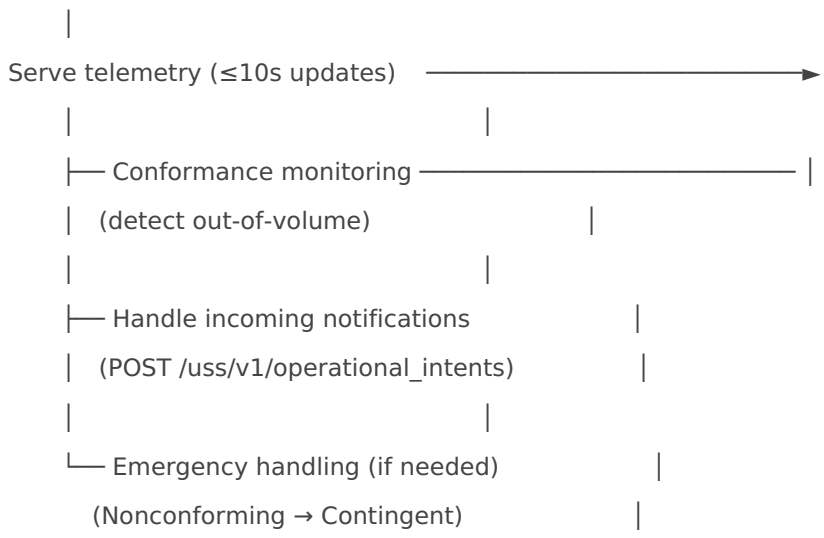
Chapter 5 — The Full Flight Lifecycle

This chapter walks through the complete technical lifecycle of a drone flight in BR-UTM — from pre-flight planning to post-flight cleanup. Every API call, data structure, and decision point is described in sequence.

5.1 Lifecycle Overview

A BR-UTM flight goes through the following stages:





5.2 Pre-Flight: Querying the Airspace

Before creating an OIR, your USS must understand the current state of the airspace in the intended operation area.

5.2.1 Query for Existing OIRs

Endpoint: `POST http://api.sandbox.brutm.dcta.mil.br/dss/dss/v1/operational_intent_references/query` **Scope:** `utm.strategic_coordination`

```
{
  "area_of_interest": {
    "volume": {
      "outline_polygon": {
        "vertices": [
          { "lat": -23.205, "lng": -45.878 },
          { "lat": -23.205, "lng": -45.872 },
          { "lat": -23.211, "lng": -45.872 },
          { "lat": -23.211, "lng": -45.878 }
        ]
      },
      "altitude_lower": { "value": 0, "reference": "W84", "units": "M" },
      "altitude_upper": { "value": 200, "reference": "W84", "units": "M" }
```

```
    },
    "time_start": { "value": "2026-07-01T10:00:00Z", "format": "RFC3339" },
    "time_end": { "value": "2026-07-01T11:00:00Z", "format": "RFC3339" }
  }
}
```

The response includes a list of `OperationalIntentReference` objects. Each contains the `uss_base_url` of the managing USS.

> **Important:** The DSS uses S2 cell indexing. It may return OIRs that do not precisely intersect your area. Your USS must perform the exact 4D intersection calculation locally after fetching the full volume details.

5.2.2 Fetch OIR Details from Peer USSs

For each OIR returned by the DSS that is **managed by another USS**, fetch the full details (including the OVN and exact volumes):

Endpoint (on the peer USS): `GET {uss_base_url}/uss/v1/operational_intents/{entityid}` **Scope:** `utm.strategic_coordination` **Token audience:** Domain of the peer USS (extracted from `uss_base_url`)

The response includes the full `OperationalIntentDetails` — volumes, off_nominal_volumes, priority, and the current OVN.

Collect the OVN from each response — you will need all of them when creating your OIR.

> **Note:** Even if local geometry calculation shows a particular OIR does not intersect with your planned volume, you **must still collect its OVN** and include it in the `key` array. The DSS guarantees airspace awareness at the S2 cell level, not the geometric level.

5.2.3 Query for Constraints

Endpoint: `POST http://api.sandbox.brutm.dcta.mil.br/dss/dss/v1/constraint_references/query` **Scope:** `utm.constraint_processing`

Same request format as the OIR query above. The response returns `ConstraintReference` objects.

5.2.4 Fetch Constraint Details from DECEA's USS

For each constraint returned, fetch the full details:

Endpoint (on DECEA's USS): `GET {constraint_uss_base_url}/uss/v1/constraints/{entityid}` **Scope:** `utm.constraint_processing`

Collect the OVN from each constraint. If your planned volume intersects a constraint, **you must not proceed** with creating the OIR in that area — deconflict or abandon the operation.

5.2.5 Local 4D Conflict Detection

After collecting all nearby OIR and Constraint volumes, perform local 4D intersection calculations:

- **Horizontal**: polygon or circle intersection in the lat/lng plane.
- **Vertical**: overlap between altitude ranges (`altitude_lower` and `altitude_upper`).
- **Temporal**: overlap between time windows (`time_start` and `time_end`).

A conflict exists only when **all three dimensions overlap simultaneously**.

If a conflict with another OIR is detected at the same priority (0), deconfliction strategies include:

- Adjusting the route or volume to avoid overlap.
- Adjusting the time window to fly when the conflicting OIR is not active.
- Suggesting the operator wait and retry.

5.3 Pre-Flight: Creating the OIR

Once you have resolved any conflicts and collected all OVNs, create the OIR in the DSS.

Endpoint: `PUT http://api.sandbox.brutm.dcta.mil.br/dss/dss/v1/operational_intent_references/{entityid}` **Scope:** `utm.strategic_coordination`

The `{entityid}` is a UUID generated by **your USS** — you own this identifier.

```
{
  "extents": [
    {
```

```
      "volume": {
        "outline_polygon": {
          "vertices": [
            { "lat": -23.207, "lng": -45.875 },
            { "lat": -23.208, "lng": -45.874 },
            { "lat": -23.209, "lng": -45.876 }
          ]
        },
        "altitude_lower": { "value": 0, "reference": "W84", "units": "M" },
        "altitude_upper": { "value": 120, "reference": "W84", "units": "M" }
      },
      "time_start": { "value": "2026-07-01T10:00:00Z", "format": "RFC3339" },
      "time_end": { "value": "2026-07-01T11:00:00Z", "format": "RFC3339" }
    }
  ],
  "key": ["<OVN_of_nearby_OIR_1>", "<OVN_of_constraint_1>"],
  "state": "Accepted",
  "uss_base_url": "https://uss.yourcompany.com/utm",
  "new_subscription": {
    "uss_base_url": "https://uss.yourcompany.com/utm",
    "notify_for_constraints": true
  },
  "flight_type": "BVLOS"
}
```

Key fields:

- `extents`: The bounding box of your operation (used for S2 indexing). This should cover all your planned volumes.
- `key`: **All OVNs** you collected in steps 5.2.2 and 5.2.4. Can be empty `[]` if the area is clear.
- `state`: Always `Accepted` at creation time.
- `uss_base_url`: Your publicly accessible base URL. Other USSs will use this to call you.
- `new_subscription`: Creates an implicit subscription for the operation's area, so you receive notifications about new OIRs/Constraints appearing in that area.
- `flight_type`: One of `VLOS`, `EVLOS`, `BVLOS`.

Response (HTTP 201):

```
{
  "operational_intent_reference": {
    "id": "2f8343be-6482-4d1b-a474-16847e01af1e",
```

```
    "manager": "your-uss-sub",
    "state": "Accepted",
    "ovn": "9d158f59-80b7-4c11-9c0c-8a2b4d936b2d",
    "uss_base_url": "https://uss.yourcompany.com/utm",
    "subscription_id": "78ea3fe8-...",
    ...
  },
  "subscribers": [
    {
      "subscriptions": [...],
      "uss_base_url": "https://other-uss.com/utm"
    }
  ]
}
```

Save the `ovn` from the response — you will need it for all subsequent updates.

5.4 Pre-Flight: Notifying Subscribers

The DSS response includes a `subscribers` list — the USSs that have subscriptions in your operation's area. You **must notify each of them** immediately after a successful DSS write.

Endpoint (on each subscriber USS): `POST {subscriber_uss_base_url}/uss/v1/operational_intents` **Scope:** `utm.strategic_coordination` **Token audience:** Domain of the subscriber USS

```
{
  "operational_intent_id": "2f8343be-6482-4d1b-a474-16847e01af1e",
  "operational_intent": {
    "reference": { ... },
    "details": {
      "volumes": [ ... ],
      "off_nominal_volumes": [],
      "priority": 0
    }
  },
  "subscriptions": [
```

```
    {
      "subscription_id": "78ea3fe8-...",
      "notification_index": 1
    }
  ]
}
```

Expected response: `204 No Content`

> **Notification timing:** You must send this notification within **5 seconds** in at least 95% of cases. If a subscriber USS is unreachable, proceed — their unavailability does not block your operation.

This same notification pattern applies for **every DSS write** — creation, updates, state changes, and deletion all require notifying the subscribers returned in that write's response.

5.5 Flight Activation

Immediately before the flight begins, activate the OIR. This is the moment the drone is cleared to launch.

5.5.1 Pre-Activation Conflict Re-Check

Before transitioning to `Activated`, perform a fresh conflict check (repeat steps 5.2.1–5.2.5). The airspace may have changed since you created the OIR. If a newly arrived OIR at the same or higher priority conflicts with your volume, **you must not activate** — deconflict first.

The rule: **first to activate wins**. If two USSs with overlapping OIRs at the same priority (0) both attempt activation simultaneously, it is the USS's responsibility to detect this and stand down if necessary.

5.5.2 Update OIR State to Activated

Endpoint: `PUT http://api.sandbox.brutm.dcta.mil.br/dss/dss/v1/operational_intent_references/{entityid}/{ovn}`
Scope: `utm.strategic_coordination`

```
{
  "extents": [ ... ],
  "key": ["<latest OVNs of all nearby entities>"],
  "state": "Activated",
  "uss_base_url": "https://uss.yourcompany.com/utm",
  "subscription_id": "78ea3fe8-...",
  "flight_type": "BVLOS"
}
```

> **Note:** The `{ovn}` in the URL must be the current OVN of your OIR (from the creation response or last update response). If the DSS returns `409 AirspaceConflictResponse`, it means new entities have appeared in the area whose OVNs you haven't acknowledged. Fetch their details, add their OVNs to `key`, and retry.

Response (HTTP 200): Returns the updated reference with a new OVN and a fresh `subscribers` list.

Notify all subscribers (step 5.4) with the `Activated` state.

5.5.3 Create the ISA (Remote ID)

Simultaneously with (or immediately after) OIR activation, register an **ISA** in the Remote ID DSS:

Endpoint: `PUT http://api.sandbox.brutm.dcta.mil.br/dss/dss/identification_service_areas/{isa_id}` **Scope:** `rid.service_provider`

```
{
  "extents": {
    "volume": {
      "outline_polygon": {
        "vertices": [
          { "lat": -23.205, "lng": -45.878 },
          { "lat": -23.205, "lng": -45.872 },
          { "lat": -23.211, "lng": -45.872 },
          { "lat": -23.211, "lng": -45.878 }
        ]
      },
      "altitude_lo": 0,
```

```
      "altitude_hi": 120
    },
    "time_start": "2026-07-01T10:00:00Z",
    "time_end": "2026-07-01T11:00:00Z"
  },
  "flights_url": "https://uss.yourcompany.com/uss/flights"
}
```

The `flights_url` tells display providers (like DECEA's monitoring tools) where to query your telemetry.

5.6 In-Flight: Serving Telemetry

Once the flight is activated and the ISA is registered, your USS must serve **live telemetry** for the drone via the Remote ID endpoints.

DECEA and other display providers will **poll** your telemetry endpoints. You must keep the data fresh.

Required update frequency: At least once every **10 seconds**.

Your USS must implement and serve:

`GET /uss/flights` — Basic flight list for a view area. **`GET /uss/flights/{id}/details`** — Detailed information for a specific flight.

See Chapter 6 — APIs to Implement for the full request/response specification.

5.7 In-Flight: Conformance Monitoring

Your USS must continuously monitor whether the drone is flying within its declared OIR volumes. This is called **conformance monitoring**.

- If the drone's reported position is inside the `volumes` of the Activated OIR → **nominal, no action needed**.
- If the drone's position is outside the `volumes` → trigger the Non-Conforming flow (Section 5.8).

5.8 In-Flight: Handling Incoming Notifications

While your flight is active, you may receive notifications from other USSs about changes in the airspace near you (new OIRs, constraint changes). These arrive as `POST` requests on your own USS:

- `POST /uss/v1/operational_intents` — A new or updated OIR in your subscription area.
- `POST /uss/v1/constraints` — A new or updated Constraint in your subscription area.

When you receive such a notification:

1. Update your internal airspace state.
2. Check whether the new/updated entity conflicts with your active OIR.
3. If a conflict exists and the new entity has **equal or higher priority**, you must deconflict — either modify your route (update the OIR) or, if the flight is active, trigger the Nonconforming flow.

This is how DECEA enforces ATM priority: DECEA's USS may create a high-priority Constraint or OIR in your area, and your USS must react within the 10-second conformance window.

5.9 Emergency Handling: Non-Conforming State

Trigger: The drone's telemetry position is outside the OIR's `volumes`. **Required response time:** Transition to `Nonconforming` **within 10 seconds** of detecting the deviation.

5.9.1 Calculate off_nominal_volumes

Compute a volume that covers the drone's current position and the deviation path. This is your "warning zone" for other operators. The exact calculation is at your USS's discretion — it should be large enough to represent the risk area but not unnecessarily large.

5.9.2 Update DSS (within 5 seconds of detection)

Endpoint: `PUT .../dss/v1/operational_intent_references/{entityid}/{ovn}`

```
{
  "extents": [ ... ],
  "state": "Nonconforming",
  "uss_base_url": "https://uss.yourcompany.com/utm",
  "subscription_id": "78ea3fe8-...",
  "flight_type": "BVLOS"
}
```

Note: `key` (OVNs) **may be omitted** in `Nonconforming` state — the DSS does not require proof of airspace awareness during emergencies.

5.9.3 Expose Telemetry Endpoint

In `Nonconforming` state, you must also expose live drone position at: `GET /uss/v1/operational_intents/{entityid}/telemetry`

This allows DECEA and other USSs to monitor the drone's actual position in real time.

5.9.4 Notify Subscribers

After the DSS update, notify all subscribers (as returned in the DSS response) with:

- The new `Nonconforming` state.
- The updated `off_nominal_volumes` included in the `details`.

5.9.5 Recovery

If the drone returns to its original volume:

1. Update the OIR back to `state: Activated`.
2. Clear `off_nominal_volumes` from the details.
3. Notify subscribers of the recovery.
4. Continue normal telemetry serving.

5.10 Emergency Handling: Contingent State

Trigger: The drone has been continuously in `Nonconforming` state for **more than 60 seconds**.

Transition to Contingent

Endpoint: `PUT .../dss/v1/operational_intent_references/{entityid}/{ovn}`

```
{
  "extents": [ ... ],
  "state": "Contingent",
  "uss_base_url": "https://uss.yourcompany.com/utm",
  "subscription_id": "78ea3fe8-...",
  "flight_type": "BVLOS"
}
```

In `Contingent` state:

- The **original** `volumes` **are no longer active** — only `off_nominal_volumes` define the warning zone.
- There is **no recovery path** — the operation must be terminated.
- The ISA **remains active** as long as the drone is still sharing telemetry.
- The telemetry endpoint `GET /uss/v1/operational_intents/{entityid}/telemetry` must remain available.
- Continue notifying subscribers of the Contingent state.

The USS must guide the operator to land the drone and then close the operation (Section 5.11).

5.11 End of Flight: Cleanup

When the flight concludes normally (or after a Contingent state is resolved), the USS must clean up all DSS records.

5.11.1 Delete the ISA

Endpoint: `DELETE /dss/identification_service_areas/{isa_id}/{version}` **Scope:** `rid.service_provider`

Delete the ISA from the Remote ID DSS. The `{version}` comes from the ISA creation response.

Notify the ISA subscribers if the DSS response includes any.

5.11.2 Delete the OIR

Endpoint: `DELETE /dss/v1/operational_intent_references/{entityid}/{ovn}` **Scope:** `utm.strategic_coordination`

Delete the OIR from the DSS entirely. There is no "Completed" state — deletion is how a flight is closed.

> **Note:** Each OIR represents a single flight. If the same drone flies a second mission (e.g., after a battery change), create a **new OIR with a new UUID** for that second flight.

5.11.3 Notify Subscribers of Deletion

The DSS delete response includes a `subscribers` list. Notify each subscriber:

Endpoint (on each subscriber): `POST {subscriber_uss_base_url}/uss/v1/operational_intents`

When sending a deletion notification, **omit the `operational_intent` field** from the body — its absence signals that the OIR has been deleted:

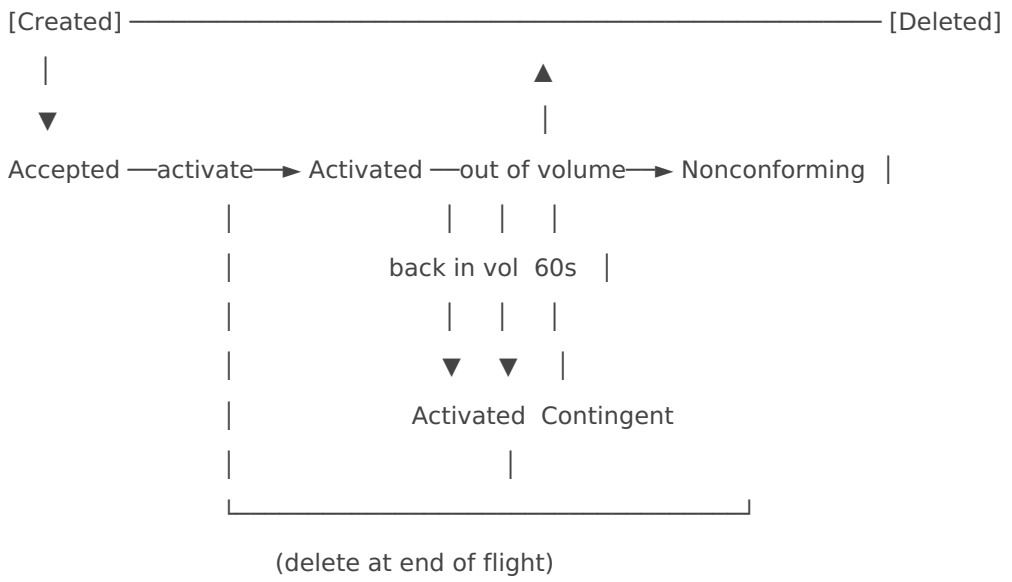
```
{
  "operational_intent_id": "2f8343be-6482-4d1b-a474-16847e01af1e",
  "subscriptions": [
    {
      "subscription_id": "78ea3fe8-...",
      "notification_index": 5
    }
  ]
}
```

5.12 Multi-Volume Operations

A single OIR can contain **multiple `Volume4D` objects** in its `volumes` array. This is the correct approach for complex flight paths. Example structure for a takeoff → route → landing mission:

```
"volumes": [
  {
   "volume": {
     "outline_circle": { "center": {...}, "radius": { "value": 50, "units": "M" }},
     "altitude_lower": { "value": 0, "reference": "W84", "units": "M" },
     "altitude_upper": { "value": 120, "reference": "W84", "units": "M" }
   },
   "time_start": { "value": "2026-07-01T10:00:00Z", "format": "RFC3339" },
   "time_end": { "value": "2026-07-01T10:05:00Z", "format": "RFC3339" }
  },
  {
   "volume": {
     "outline_polygon": { "vertices": [...] },
     "altitude_lower": { "value": 80, "reference": "W84", "units": "M" },
     "altitude_upper": { "value": 120, "reference": "W84", "units": "M" }
   },
   "time_start": { "value": "2026-07-01T10:05:00Z", "format": "RFC3339" },
   "time_end": { "value": "2026-07-01T10:55:00Z", "format": "RFC3339" }
  },
  {
   "volume": {
     "outline_circle": { "center": {...}, "radius": { "value": 50, "units": "M" }},
     "altitude_lower": { "value": 0, "reference": "W84", "units": "M" },
     "altitude_upper": { "value": 120, "reference": "W84", "units": "M" }
   },
   "time_start": { "value": "2026-07-01T10:55:00Z", "format": "RFC3339" },
   "time_end": { "value": "2026-07-01T11:00:00Z", "format": "RFC3339" }
  }
]
```

5.13 Complete State Transition Summary


[Created]
[Deleted]
Accepted
activate
Activated
out of volume
Nonconforming
back in vol 60s
Activated Contingent
(delete at end of flight)

Chapter 6 — APIs Your USS Must Implement

This chapter describes all the HTTP endpoints that your USS must expose publicly. These are the endpoints that DECEA and other USSs will call against your server. They are divided into:

1. **Production endpoints** — required for all live operations.
2. **Testing/Homologation endpoints** — required for the validation process.

All endpoints must be accessible over HTTPS from the public internet. The base domain of your server becomes the `uss_base_url` registered in the DSS.

6.1 Production Endpoints (USS-to-USS)

These are defined in `utm.yaml` and `remoteid.yaml` and must be live during production operations.

6.1.1 `GET /uss/v1/operational_intents/{entityid}`

Purpose: Return the full details of one of your USS's Operational Intents to another USS or DECEA that is querying it.

Required scope (caller must have): `utm.strategic_coordination`

Response body:

```
{
  "operational_intent": {
    "reference": {
      "id": "2f8343be-6482-4d1b-a474-16847e01af1e",
      "manager": "your-uss-sub",
      "uss_availability": "Normal",
```

```
      "version": 3,
      "state": "Activated",
      "ovn": "9d158f59-80b7-4c11-9c0c-8a2b4d936b2d",
      "time_start": { "value": "2026-07-01T10:00:00Z", "format": "RFC3339" },
      "time_end": { "value": "2026-07-01T11:00:00Z", "format": "RFC3339" },
      "uss_base_url": "https://uss.yourcompany.com/utm",
      "subscription_id": "78ea3fe8-..."
    },
    "details": {
      "volumes": [ ... ],
      "off_nominal_volumes": [],
      "priority": 0,
      "flight_type": "BVLOS"
    }
  }
}
```

Key behaviors:

- The `ovn` in the reference is what other USSs collect to include in their `key` array.
- When the OIR is in `Nonconforming` or `Contingent` state, the `off_nominal_volumes` field must be populated.
- In `Contingent` state, `volumes` may be empty (only `off_nominal_volumes` applies).

6.1.2 `POST /uss/v1/operational_intents`

Purpose: Receive notifications from other USSs about Operational Intents in your subscription area. This is how you learn about new, updated, or deleted OIRs near your operations.

Required scope (caller must have): `utm.strategic_coordination`

Request body (OIR created or updated):

```
{
  "operational_intent_id": "2f8343be-6482-4d1b-a474-16847e01af1e",
  "operational_intent": {
    "reference": { ... },
    "details": {
      "volumes": [ ... ],
      "off_nominal_volumes": [],
```

```
    "priority": 0
   }
  },
  "subscriptions": [
   {
    "subscription_id": "78ea3fe8-...",
    "notification_index": 3
   }
  ]
}
```

Request body (OIR deleted):

```
{
  "operational_intent_id": "2f8343be-6482-4d1b-a474-16847e01af1e",
  "subscriptions": [
   {
    "subscription_id": "78ea3fe8-...",
    "notification_index": 5
   }
  ]
}
```

When `operational_intent` is absent, the OIR has been deleted.

Expected response: `204 No Content`

Key behaviors:

- Update your internal airspace state upon receiving this notification.
- Check whether the new/updated OIR or the deletion changes your own conflict situation.
- If a new high-priority OIR or Constraint conflicts with your active operation, begin deconfliction.

6.1.3 `GET /uss/v1/constraints/{entityid}`

Purpose: Return the full details of a Constraint managed by your USS. In Phase 1, only DECEA creates Constraints, so this endpoint is primarily implemented by **DECEA's USS**. However, it must also be implemented by all USSs for completeness and future phases.

Required scope (caller must have): `utm.constraint_processing`

Response body:

```
{
  "constraint": {
    "reference": {
      "id": "c036326c-...",
      "manager": "decea-uss",
      "version": 1,
      "ovn": "a1b2c3d4-...",
      "time_start": { "value": "2026-07-01T08:00:00Z", "format": "RFC3339" },
      "time_end": { "value": "2026-07-01T20:00:00Z", "format": "RFC3339" },
      "uss_base_url": "https://uss.decea.mil.br/utm"
    },
    "details": {
      "volumes": [ ... ],
      "type": "com.decea.restricted_area"
    }
  }
}
```

6.1.4 POST /uss/v1/constraints

Purpose: Receive notifications about new, updated, or deleted Constraints in your subscription area. These come from DECEA's USS (and in future phases, potentially other authorized entities).

Required scope (caller must have): `utm.constraint_management`

Request body (Constraint notification):

```
{
  "constraint_id": "c036326c-...",
  "constraint": {
    "reference": { ... },
    "details": { "volumes": [ ... ], "type": "..." }
  },
  "subscriptions": [
    { "subscription_id": "...", "notification_index": 1 }
  ]
}
```

When `constraint` is absent, the Constraint has been deleted.

Expected response: `204 No Content`

6.1.5 GET /uss/v1/operational_intents/{entityid}/telemetry

Purpose: Serve live telemetry for an OIR in `Nonconforming` or `Contingent` state. DECEA's monitoring systems and other USSs call this to track a drone that has deviated from its plan.

Required scope (caller must have): `utm.conformance_monitoring_sa`

Response body:

```
{
  "telemetry": {
    "time_measured": { "value": "2026-07-01T10:32:15Z", "format": "RFC3339" },
    "position": {
      "longitude": -45.876,
      "latitude": -23.210,
      "altitude": { "value": 115.2, "reference": "W84", "units": "M" },
      "accuracy_h": "HA10m",
      "accuracy_v": "VA10m"
    },
    "velocity": {
      "speed": 8.5,
      "units_speed": "MetersPerSecond",
      "track": 270.0
    }
  },
  "next_telemetry_opportunity": { "value": "2026-07-01T10:32:25Z", "format": "RFC3339" }
}
```

This endpoint must only be available when the OIR is in `Nonconforming` or `Contingent` state. You may return `404` in other states.

6.1.6 GET /uss/flights

Purpose: Return basic flight information for all active drones in a given geographic view area. This is the primary Remote ID polling endpoint.

Required scope (caller must have): `rid.display_provider`

Query parameters:

- `view` — Bounding box as `lat1,lng1,lat2,lng2` (southwest and northeast corners).
- `recent_positions_duration` — How many seconds of recent position history to include (optional).

Response body:


```
{
 "timestamp": { "value": "2026-07-01T10:32:15Z", "format": "RFC3339" },
 "flights": [
  {
   "id": "flight-uuid-here",
   "aircraft_type": "Helicopter",
   "current_state": {
    "timestamp": { "value": "2026-07-01T10:32:15Z", "format": "RFC3339" },
    "timestamp_accuracy": 0.1,
    "position": {
     "lat": -23.2071,
     "lng": -45.8750,
     "alt": 115.2,
     "accuracy_h": "HA10m",
     "accuracy_v": "VA10m",
     "extrapolated": false
    },
    "track": 270.0,
    "speed": 8.5,
    "speed_accuracy": "SA3mps",
    "vertical_speed": 0.0,
    "operational_status": "Airborne"
   },
   "recent_positions": [ ... ]
  }
 ],
 "no_isas_present": false
}
```

Update frequency: The data returned must reflect the drone's actual position, updated at most every **10 seconds**.

6.1.7 `GET /uss/flights/{id}/details`

Purpose: Return detailed information for a specific flight, including UAS identification and operator data.

Required scope (caller must have): `rid.display_provider`

Response body:

```
{
  "details": {
    "id": "flight-uuid-here",
    "uas_id": {
      "registration_id": "PR-XXXX"
    },
    "operator_id": "operator-registration-id",
    "operator_location": {
      "position": { "lat": -23.208, "lng": -45.878 },
      "altitude": { "value": 0, "reference": "W84", "units": "M" }
    },
    "operation_description": "Package delivery - Sector A",
    "auth_data": {
      "format": 0,
      "data": ""
    }
  }
}
```

6.2 Additional Required Endpoints

These endpoints are required by the qualification rules (`uss-qualification-rules.md`) and are called during homologation testing and normal operations.

6.2.1 GET /version

Purpose: Return the current deployed version of your USS software.

No authentication required (or use your standard validation).

Response body:

```
{
  "version": "1.4.2"
}
```

6.2.2 GET /diagnostics/time

Purpose: Return the current system time and NTP synchronization status. Used by DECEA to verify your system clock is properly synchronized.

Response body:

```
{
  "system_time": "2026-07-01T10:32:15.482Z",
  "ntp_sync": {
    "synchronized": true,
    "source": "ntp.decea.gov.br",
    "stratum": 2,
    "offset_ms": 38,
    "last_sync": "2026-07-01T10:32:05.000Z"
  },
  "timestamp_format": "ISO 8601"
}
```

6.2.3 POST /telemetry

Purpose: Accept drone telemetry data injection during DECEA's homologation testing. DECEA uses this to simulate drone position updates and test your conformance monitoring logic.

Request body:

```
{
  "flight_id": "a8c4af2a-6640-41d9-b8e7-719fd19a2fce",
  "aircraft_type": "Helicopter",
  "operational_intent_id": "<id-of-the-operational-intent>",
  "position": {
    "timestamp": { "value": "2026-07-01T10:32:00Z", "format": "RFC3339" },
    "lat": -23.207184,
    "lng": -45.875054,
    "alt": 115.0,
    "accuracy_h": "HA10m",
    "accuracy_v": "VA10m",
    "extrapolated": false
  },
  "operational_status": "Airborne",
  "track": 270.0,
  "speed": 10.0,
  "vertical_speed": 0.0,
  "test_metadata": {
    "scenario_id": "VOL4D-EXIT-REENTRY-59S",
    "event": "exit_volume",
    "t_offset_seconds": 0
  }
}
```

6.3 Homologation/Testing Endpoints

These endpoints are defined in `flights.yaml`, `injection.yaml`, and `versioning.yaml`. They are **only called by DECEA's automated testing framework** during homologation. You implement them, but they are not used in production operations.

6.3.1 Flight Planning Testing (`flights.yaml`)

These endpoints simulate the user experience of creating and managing flight plans, allowing DECEA's test framework to exercise your OIR creation and management logic.

Method	Path	Description
`GET`	`/status`	Return readiness status of your testing interface.
`POST`	`/clear_area_requests`	Instruct your USS to cancel all flight plans in a given area.
`PUT`	`/flight_plans/{flight_plan_id}`	Create or update a flight plan (simulates a user action).
`DELETE`	`/flight_plans/{flight_plan_id}`	Delete a flight plan.
`GET`	`/user_notifications`	Return notifications received by the virtual user.

Scopes:

- `interuss.flight_planning.direct_automated_test` — For test director operations (clear area, delete, status).
- `interuss.flight_planning.plan` — For virtual user operations (create/update, notifications).

6.3.2 Remote ID Data Injection (`injection.yaml`)

These endpoints allow DECEA's test framework to inject simulated drone telemetry directly into your USS, to test your Remote ID serving logic.

Method	Path	Description
`PUT`	`/tests/{test_id}`	Create a test: inject one or more simulated flights with telemetry sequences.
`DELETE`	`/tests/{test_id}/{version}`	Delete a test and remove all injected data.
`GET`	`/user_notifications`	Return notifications received by the virtual user during the test.

Scope: `rid.inject_test_data`

Example injection request:

```
{
  "requested_flights": [
    {
      "injection_id": "test-flight-001",
```

```
    "aircraft_type": "Helicopter",
    "telemetry": [
      {
        "timestamp": { "value": "2026-07-01T10:00:00Z", "format": "RFC3339" },
        "position": { "lat": -23.2071, "lng": -45.8750, "alt": 100.0 },
        "track": 90.0,
        "speed": 10.0,
        "vertical_speed": 0.0,
        "operational_status": "Airborne",
        "accuracy_h": "HA10m",
        "accuracy_v": "VA10m"
      }
    ],
    "details_responses": [
      {
        "effective_after": "2026-07-01T09:59:00Z",
        "details": {
          "id": "test-flight-001",
          "operator_id": "OP-TEST-123",
          "uas_id": { "registration_id": "PR-TEST01" }
        }
      }
    ]
  }
 ]
}
```

6.3.3 Versioning (`versioning.yaml`)

Method	Path	Description
`GET`	`/versions/{system_identity}`	Return the version of a specific system component.

Scope: `interuss.versioning.read_system_versions`

Example:

```
GET /versions/br.mil.decea.brutm.uss.v1
```

```
{
  "system_identity": "br.mil.decea.brutm.uss.v1",
  "system_version": "1.4.2"
}
```

6.4 Summary of All Required Endpoints

Endpoint	Type	When Required
`GET /uss/v1/operational_intents/{entityid}`	Production	Always
`POST /uss/v1/operational_intents`	Production	Always
`GET /uss/v1/constraints/{entityid}`	Production	Always
`POST /uss/v1/constraints`	Production	Always
`GET /uss/v1/operational_intents/{entityid}/telemetry`	Production	When OIR is Nonconforming/Contingent
`GET /uss/flights`	Production (Remote ID)	When any flight is Activated
`GET /uss/flights/{id}/details`	Production (Remote ID)	When any flight is Activated
`GET /version`	Operations	Always
`GET /diagnostics/time`	Operations	Always
`POST /telemetry`	Operations/Testing	During homologation
`GET /status`	Testing	Homologation
`POST /clear_area_requests`	Testing	Homologation
`PUT /flight_plans/{id}`	Testing	Homologation
`DELETE /flight_plans/{id}`	Testing	Homologation
`GET /user_notifications`	Testing	Homologation
`PUT /tests/{test_id}`	Testing (Remote ID)	Homologation
`DELETE /tests/{test_id}/{version}`	Testing (Remote ID)	Homologation
`GET /versions/{system_identity}`	Testing	Homologation

Chapter 7 — Non-Functional Requirements

This chapter defines the non-functional requirements (NFRs) that your USS must satisfy to be approved for the BR-UTM ecosystem. These are enforced during homologation and are continuously expected in production.

Failure to meet these requirements can result in:

- Rejection during homologation.
- Operational incidents (missed emergency notifications, incorrect timestamps, clock drift).
- Potential deauthorization from the ecosystem.

7.1 Time Synchronization (NTP)

Requirement: Your USS's system clock must be synchronized with DECEA's NTP server with a precision of **≤ 5 seconds**.

Parameter	Requirement
NTP Server	`ntp.decea.gov.br`
Maximum offset	≤ 5 seconds
Authentication	Required (NTP authentication, integrity and anti-spoofing protection)

All timestamps generated by your system (for OIR time windows, telemetry, notifications, and audit logs) must be derived from this synchronized clock.

Verification endpoint: `GET /diagnostics/time`

Your `GET /diagnostics/time` endpoint must return current sync status, including the NTP source, stratum, offset in milliseconds, and last synchronization time. DECEA will call this endpoint during homologation to verify compliance.

```
{
  "system_time": "2026-07-01T10:32:15.482Z",
```

```
  "ntp_sync": {
    "synchronized": true,
    "source": "ntp.decea.gov.br",
    "stratum": 2,
    "offset_ms": 38,
    "last_sync": "2026-07-01T10:32:05.000Z"
  },
  "timestamp_format": "ISO 8601"
}
```

7.2 Timestamp Format

Requirement: All dates and times stored or transmitted by your USS must conform to **ISO 8601 (RFC 3339)** with the **UTC timezone** (expressed as `Z` suffix).

Parameter	Requirement
Format	ISO 8601 / RFC 3339
Timezone	UTC (`Z`)
Examples	`"2026-07-01T10:32:15Z"` or `"2026-07-01T10:32:15.482Z"`

Never use local time or non-UTC offsets in any field transmitted to the DSS, to other USSs, or in any API response. This includes:

- OIR time windows (`time_start`, `time_end`)
- Volume4D time bounds
- Telemetry timestamps
- ISA time windows
- Audit log entries

When wrapping timestamps in the BR-UTM `Time` structure:


```
{ "value": "2026-07-01T10:32:15Z", "format": "RFC3339" }
```

7.3 Notification Latency

Requirement: When your USS creates, updates, or deletes an OIR or Constraint in the DSS and receives a list of subscribers to notify, it must send the `POST /uss/v1/operational_intents` (or `POST`

`/uss/v1/constraints`) notifications to each subscriber within:

Metric	Requirement
Target latency	≤ 5 seconds
Required percentile	≥ 95% of cases

This means that in 95% of all notification events, the subscriber USS must receive the notification within 5 seconds of your DSS write completing. Up to 5% of cases may exceed this threshold (e.g., due to transient network conditions).

Implementation guidance:

- Use asynchronous notification dispatch to avoid blocking the main OIR creation flow.
- Implement reasonable timeouts (e.g., 5–10 seconds per subscriber) so a slow subscriber doesn't delay notifications to others.
- Fire notifications in parallel when there are multiple subscribers.

7.4 Conformance Monitoring Response Time

Requirement: When your USS detects that a drone has left its declared OIR volumes, it must:

Action	Time Limit
Detect position out-of-volume	≤ 10 seconds after last telemetry update shows deviation
Update DSS with `Nonconforming` state	≤ 5 seconds after detection

This means the worst-case end-to-end timeline from drone position deviation to DSS update is **15 seconds**.

Additionally:

- After **60 continuous seconds** in `Nonconforming` state, the USS must transition to `Contingent`.
- Both state transitions must include notification to all subscribers in the area.

7.5 Telemetry Update Frequency

Requirement: Your USS must update the telemetry data served at `GET /uss/flights` at an interval of:

Metric	Requirement
Maximum update interval	10 seconds
When required	During any `Activated`, `Nonconforming`, or `Contingent` operation

In other words: the position data returned by `GET /uss/flights` must never be more than 10 seconds stale for an active flight.

7.6 Geospatial Intersection Precision

Requirement: Your USS must be able to calculate 4D intersection between volumes with:

Parameter	Requirement
Horizontal precision	1 centimeter (1 cm)
Dimensions	Latitude, longitude, altitude, time

This precision is required for accurate conflict detection between OIRs and between OIRs and Constraints. Using imprecise intersection algorithms (e.g., simple bounding box checks) is not acceptable and will fail homologation scenarios.

Your intersection logic must handle both **polygon** and **circle** geometries, and combinations thereof.

7.7 Audit Logging

Requirement: Your USS must maintain comprehensive audit logs of all safety-critical operations. All audit log entries must use timestamps derived from the NTP-synchronized clock.

The audit log must capture:

- All OIR state transitions (creation, activation, nonconformance, contingency, deletion).
- All peer-to-peer notifications sent and received.
- All telemetry positions recorded during active operations.

- All conflict detections and deconfliction actions taken.
- All token validation results for inbound requests.

These logs must be retained for traceability and may be requested by DECEA during incident investigations.

7.8 Subscription Continuity

Requirement: Your USS must maintain an active subscription for any area where you have an active OIR. This ensures you receive notifications about changes in the airspace around your operations.

- Subscriptions must remain active for the full duration of the OIR's time window.
- If a subscription is lost (e.g., due to a DSS connectivity issue), your USS must re-create it as soon as connectivity is restored.
- Your USS must process all incoming notifications and re-evaluate its conflict status whenever a new or updated OIR/Constraint arrives in the subscribed area.

7.9 DSS Discoverability

Requirement: An OIR may only transition between states (`Accepted`, `Activated`, `Nonconforming`, `Contingent`) if it is currently discoverable by other USSs via the DSS.

This means:

- Before transitioning state, your OIR must exist in the DSS with the correct `extents`, `uss_base_url`, and `subscription_id`.
- If your DSS write fails, you must not proceed with the state transition.
- If your USS cannot reach the DSS, active flights must not proceed to activation.

7.10 Summary Table

NFR	Requirement
NTP synchronization	`ntp.decea.gov.br`, ≤5s offset, authenticated
Timestamp format	ISO 8601 / RFC 3339, UTC (`Z`)

NFR	Requirement
Notification latency	≤5 seconds in ≥95% of cases
Nonconformance detection	≤10 seconds
Nonconformance DSS update	≤5 seconds after detection
Contingent trigger	After 60 continuous seconds Nonconforming
Telemetry update interval	≤10 seconds for active flights
Geospatial intersection precision	1 cm
Audit logs	All safety-critical events, NTP-derived timestamps
Subscription continuity	Active for full OIR duration
DSS discoverability	Required before any state transition

Chapter 8 — Homologation

This chapter describes the homologation (validation) process that a company must complete before receiving a production API Key and being authorized to operate in the BR-UTM ecosystem.

8.1 What Is Homologation?

Homologation is DECEA's process for validating that a USS implementation:

- Correctly implements all required APIs (as defined in the OpenAPI specifications).
- Behaves correctly in all defined operational scenarios.
- Meets all non-functional requirements (timing, precision, synchronization).
- Is ready to operate safely alongside other USSs in the live ecosystem.

The process is **primarily manual** — conducted by DECEA engineers who run test scenarios against your deployed system. DECEA may also use **internal automated testing tools** (`uss_qualifier` or equivalent) to exercise specific scenarios programmatically.

8.2 Prerequisites for Homologation

Before requesting homologation, your USS must:

1. **Implement all mandatory production endpoints** (see Chapter 6):
   - `GET /uss/v1/operational_intents/{entityid}`
   - `POST /uss/v1/operational_intents`
   - `POST /uss/v1/constraints`
   - `GET /uss/flights`
   - `GET /uss/flights/{id}/details`
   - `GET /version`
   - `GET /diagnostics/time`
   - `POST /telemetry`
2. **Implement all testing endpoints**:
   - All `flights.yaml` endpoints
   - All `injection.yaml` endpoints
   - The `versioning.yaml` endpoint
3. **Be deployed and reachable from the internet** with a valid HTTPS base URL.

4. **Be registered in the Sandbox** with a development API Key and able to interact with the Sandbox DSS.
5. **Satisfy all non-functional requirements** (see Chapter 7), especially NTP synchronization.

8.3 Requesting Homologation

To initiate the homologation process:

1. Contact DECEA through the official support channels:
   - **Mattermost** (developer channel)
   - **Central de Ajuda** (ticketing system)
2. Provide:
   - Your company name and CNPJ.
   - The version of your software being submitted.
   - The publicly accessible base URL of your USS.
   - Contact details for the technical team.
3. DECEA will schedule the homologation session and provide:
   - A testing API Key (granting access to the test ecosystem).
   - The URL of the test ecosystem (if separate from the main Sandbox).
   - The test scenario list to prepare for.

8.4 What DECEA Tests

DECEA validates your USS against 18 defined scenarios. Each scenario tests one or more functional and non-functional requirements.

Scenario Matrix

#	Scenario	Key Requirements Tested
1	Register an operation in an empty area	Auth (8.1), NTP, notifications (8.2), OIR creation (8.3), conflict check (8.4), DSS discoverability
2	Register an operation near a non-intersecting Constraint	+ Constraint query and processing (8.7)

#	Scenario	Key Requirements Tested
3	Register an operation near an intersecting Constraint	+ 4D intersection detection, deconfliction (8.5), operator notification
4	Register an operation near a non-intersecting OIR	+ Peer-to-peer OIR details fetch
5	Register an operation near an intersecting OIR	+ Deconfliction, operator notification
6	Delete an operation (OIR)	+ Deletion notification to subscribers
7	Activate an operation conflicting with another active OIR at the same priority	+ Pre-activation conflict check (8.6), blocking activation
8	Activate an operation conflicting with an active OIR at higher priority	+ Deconfliction before activation
9	Activate an operation conflicting with an active OIR at lower priority	+ Correctly allows activation when lower priority conflict exists
10	DECEA creates a Constraint over an existing Accepted OIR	+ Reaction to constraint notification (8.11), deconfliction or state change
11	DECEA creates a Constraint over an existing Activated OIR	+ Immediate reaction (Nonconforming or deconfliction)
12	DECEA's USS activates a higher-priority OIR over an active OIR	+ Priority-based conflict resolution, deconfliction
13	USS creates an ISA when the OIR is activated	+ Remote ID ISA lifecycle (8.8)
14	USS shares drone position during an active operation	+ Telemetry serving at `/uss/flights` and `/uss/flights/{id}/details`
15	USS updates drone position every 10 seconds	+ Update frequency requirement
16	Drone position exits OIR volume → transition to Nonconforming	+ Conformance monitoring timing (≤10s detection, ≤5s DSS update)
17	Drone position exits OIR → Nonconforming → returns → back to Activated	+ Recovery from non-conformance
18	Drone position exits OIR → Nonconforming for 60s → Contingent	+ Full emergency state machine

Requirement Mapping

Code	Full Requirement
RF 1	4D geospatial intersection calculation, 1 cm precision

Code	Full Requirement
RF 2	DSS discoverability before state transitions
RF 3	Automatic high priority for critical situations
RF 4	Transition to Accepted only if no higher-priority conflict
RF 5	Pre-activation final conflict check
RF 6	Continuous situational awareness via subscriptions
RF 7	Automatic conflict notification to operators
RF 8	CMSA-only role for Nonconforming/Contingent transitions
RF 9	Constraint intersection analysis before OIR creation
RF 10	Continuous constraint monitoring during operations
RF 11	Automatic propagation of constraint notifications to affected operators
RF 12	Full telemetry recording during conformance monitoring
RF 13	Aircraft position updated and available within ≤10 seconds
RNF 1	NTP synchronization to `ntp.decea.gov.br`
RNF 2	Consistent timestamps from synchronized clock
RNF 3	Notification latency ≤5s in ≥95% of cases
RNF 4	ISO 8601 / RFC 3339 with UTC timezone
RNF 5	Audit logs with NTP-derived timestamps

8.5 How the Automated Testing Works

For scenarios involving Remote ID (scenarios 13–15), DECEA's test framework calls your `injection.yaml` endpoints to inject simulated telemetry:

1. DECEA's framework calls `PUT /tests/{test_id}` with a sequence of simulated aircraft positions.
2. Your USS processes these as if they were real drone telemetry and makes them available via `GET /uss/flights`.
3. DECEA's framework queries `GET /uss/flights` and `GET /uss/flights/{id}/details` to verify the data is correct and timely.
4. After the test, `DELETE /tests/{test_id}/{version}` clears the injected data.

For scenarios involving flight planning (scenarios 1–12), DECEA's framework uses your `flights.yaml` endpoints:

1. DECEA's framework calls `PUT /flight_plans/{id}` to simulate a user creating a flight plan.
2. Your USS translates this into an OIR creation in the DSS and notifies subscribers.
3. DECEA validates the DSS state and your subscriber notifications.
4. `POST /clear_area_requests` may be used to reset the test environment between scenarios.

For conformance monitoring scenarios (16–18), DECEA combines both `injection.yaml` (to inject positions outside the volume) and `flights.yaml` (to set up the OIR), then monitors whether your USS correctly transitions states and notifies within the required time limits.

8.6 Testing Environment Access

During homologation, DECEA provides a dedicated test environment:

- A test API Key with access to the test ecosystem.
- Possibly a separate test DSS instance.
- Access credentials for DECEA's monitoring systems (Interface UTM) to observe your operations during the test.

This testing key and environment are separate from your normal development (Sandbox) key.

8.7 After Successful Homologation

Upon passing all scenarios:

1. DECEA grants your software the **U1 permission level** for the **production environment**.
2. You receive a **production API Key** tied to your validated software version.
3. You can now:
   - Create UTM Zones in the production Portal UTM.
   - Begin operating flights in production.
   - Generate sub-keys for third-party operators using your software.

Third-Party Sub-Keys

If your USS software is a platform sold to third-party drone operators:

- Your company (as the validated software owner) creates **sub-API Keys** for each third-party client.
- The third-party client uses their sub-key to authenticate and to create their own UTM Zones.
- All operations under sub-keys are still associated with your validated software version.

8.8 Re-Homologation

If you release a **new major version** of your USS software with significant architectural changes, a new homologation process may be required. Contact DECEA to determine whether re-validation is needed for your specific changes.

Minor updates and bug fixes that do not affect the API contract or safety-critical behaviors typically do not require re-homologation.

8.9 Homologation Checklist

Use this checklist before requesting homologation:

APIs

- ☐ `GET /uss/v1/operational_intents/{entityid}` implemented and tested
- ☐ `POST /uss/v1/operational_intents` implemented (correctly handles creation, update, deletion)
- ☐ `GET /uss/v1/constraints/{entityid}` implemented
- ☐ `POST /uss/v1/constraints` implemented
- ☐ `GET /uss/v1/operational_intents/{entityid}/telemetry` implemented (active in Nonconforming/Contingent)
- ☐ `GET /uss/flights` implemented with correct data structure
- ☐ `GET /uss/flights/{id}/details` implemented
- ☐ `GET /version` implemented
- ☐ `GET /diagnostics/time` implemented with NTP status
- ☐ `POST /telemetry` implemented
- ☐ All `flights.yaml` endpoints implemented
- ☐ All `injection.yaml` endpoints implemented

- [ ] `versioning.yaml` endpoint implemented

Functional

- [ ] OIR creation → DSS → subscriber notification flow works end-to-end
- [ ] OIR state transitions (Accepted → Activated → Nonconforming → Contingent) work correctly
- [ ] Pre-activation conflict check is performed
- [ ] Constraint queries and processing are implemented
- [ ] ISA is created on activation and deleted on flight closure
- [ ] Incoming `POST /uss/v1/operational_intents` notifications trigger conflict re-evaluation
- [ ] OIR deletion triggers subscriber notification (with `operational_intent` omitted)
- [ ] Nonconforming detected within 10 seconds
- [ ] DSS updated within 5 seconds of Nonconforming detection
- [ ] Contingent triggered after 60 seconds of Nonconforming

Non-Functional

- [ ] NTP synchronized to `ntp.decea.gov.br` with ≤5s offset
- [ ] All timestamps in ISO 8601 / RFC 3339 / UTC
- [ ] Subscriber notifications sent within 5 seconds in ≥95% of cases
- [ ] Telemetry data updated at most every 10 seconds
- [ ] 4D intersection precision at 1 cm
- [ ] Audit logs in place with NTP-derived timestamps
- [ ] All inbound JWT tokens validated (signature, expiry, audience, scope)
- [ ] System deployed publicly on HTTPS