# Radiocall

# Schema Reference

Data Model & Grading System Documentation

| 8 | 9 | 500 |
| --- | --- | --- |
| Collections | Relations | Radiocalls |

Aevoli Training Platform

Version 1.0 · January 2026

# Contents

# Conceptual Overview

The radiocall system enables structured practice of ATC readback skills. Understanding **why** the data is structured this way is key to using it effectively.

## The Core Problem

When a pilot receives an ATC instruction, they must read back critical elements accurately. Grading this requires:

**① Decomposition**

Break transmission into individual instructions

**② Classification**

Know which elements are critical

**③ Weighting**

Score based on severity

## The Solution: Hierarchical Data

**💡 Key Insight**

A single transmission contains multiple instructions. Each instruction has a **type** that determines its grading weight. This hierarchy enables granular, fair scoring.

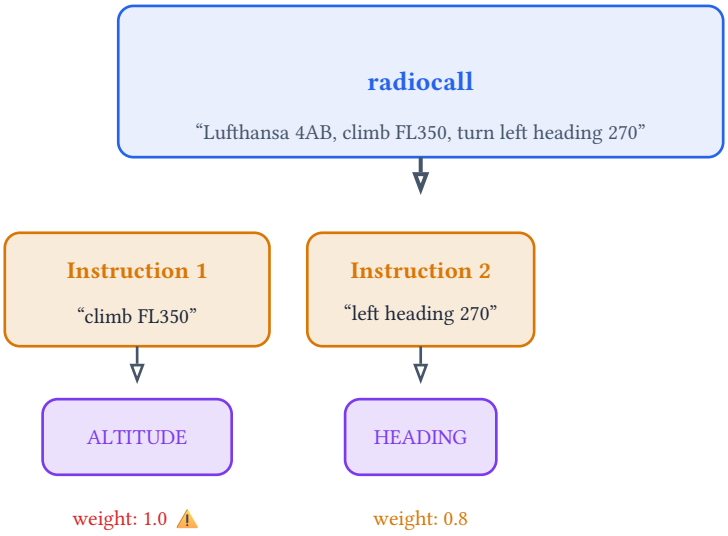



Figure 1: Hierarchical decomposition of a transmission

# Entity Reference

## Reference Data (populated once, used by all radiocalls)

These collections define the "vocabulary" of the system.

### ◆ instruction_type

| | |
|---|---|
| `code` | Unique identifier (e.g., `CLIMB`, `DESCEND`) |
| `category` | Grouping: altitude, heading, speed… |
| `is_critical` | Must be read back correctly? |
| `grading_weight` | 0.0–1.0 scoring multiplier |

### ◆ callsign_format

| | |
|---|---|
| `airline_code` | ICAO code (e.g., `DLH`) |
| `airline_callsign` | Spoken name (e.g., `Lufthansa`) |
| `difficulty` | Complexity level |
| `phonetic_template` | How to speak the callsign |

## Core Data (generated content)

### ● radiocall

**Identification**

| | |
|---|---|
| `aircraft_callsign` | e.g., "DLH4AB" |
| `callsign_phonetic` | e.g., "Lufthansa 4 Alpha Bravo" |
| `airport` | → linked airport |

**Classification**

| | |
|---|---|
| `category` | ground, departure, enroute… |
| `difficulty` | super_easy → hard |
| `flight_phase` | taxi, takeoff, cruise… |

**Content**

| | |
|---|---|
| `full_transmission` | Complete ATC message as spoken |
| `expected_readback` | What pilot should say back |
| `critical_elements` | JSON array of gradable elements |

### ▸ radiocall_instruction

Links to parent `radiocall` and `instruction_type`

### ✓ acceptable_variation

Alternative correct readbacks

| | |
|---|---|
| `radiocall` | → parent radiocall |

| | |
|---|---|
| `sequence` | Order in transmission (1, 2, 3…) |
| `raw_value` | The actual value (e.g., “FL350”) |
| `readback_text` | Expected spoken readback |

| | |
|---|---|
| `variation_text` | Alternative phrasing |
| `notes` | When this is acceptable |

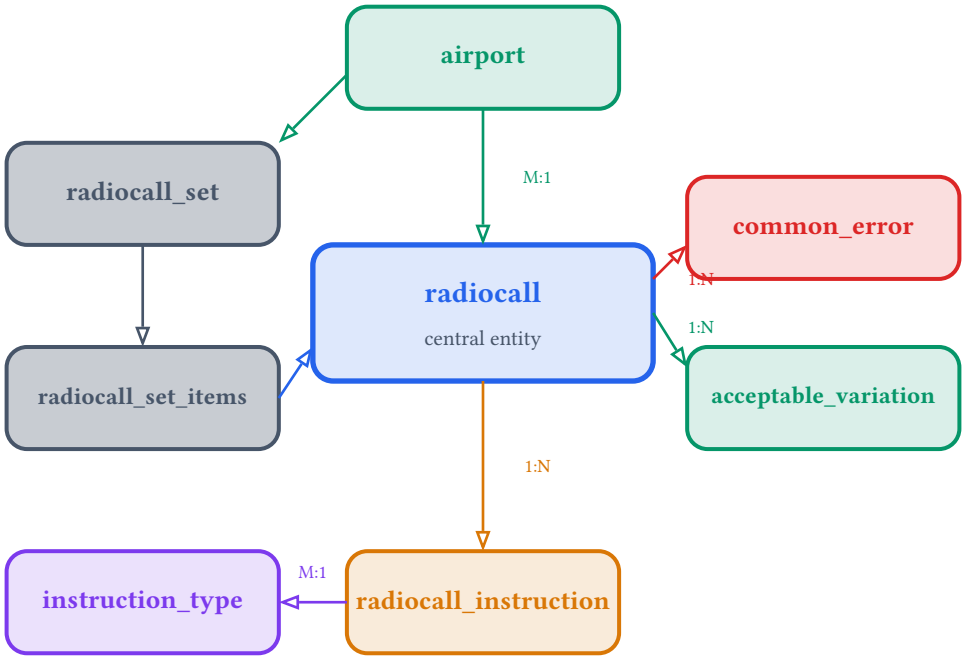
## ✗ common_error

Typical mistakes for targeted feedback

| | | | |
|---|---|---|---|
| `error_code` | e.g., `WRONG_ALTITUDE` | `example` | What users might say wrong |
| `severity` | critical, major, minor | `feedback_text` | Constructive feedback |

# Relationships

## Visual Schema




## Relationship Table

| From | Type | To | Purpose |
|---|---|---|---|
| `radiocall` | M:1 | `airport` | Where the transmission occurs |
| `radiocall_instruction` | M:1 | `radiocall` | Parent transmission (ordered by sequence) |
| `radiocall_instruction` | M:1 | `instruction_type` | Classification for grading weight |
| `acceptable_variation` | M:1 | `radiocall` | Alternative correct answers |
| `common_error` | M:1 | `radiocall` | Expected mistakes for feedback |
| `radiocall_set` | M:1 | `airport` | Airport-specific practice sets |
| `radiocall_set_items` | M:1 | `radiocall_set` | Parent set |
| `radiocall_set_items` | M:1 | `radiocall` | Member radiocall (ordered by sequence) |

# Grading System

The grading system is designed to be **fair**, **educational**, and **realistic**.

## Critical Elements

Each radiocall stores a `critical_elements` JSON array. This is the "answer key" for grading:

```
{
  "critical_elements": [
    {"type": "callsign", "value": "DLH4AB", "display": "Lufthansa 4AB",
"weight": 1.0},
    {"type": "altitude", "value": "FL350", "display": "flight level 350",
"weight": 1.0},
    {"type": "heading", "value": "270", "display": "heading 270", "weight":
0.8}
  ]
}
```

> **Grading Algorithm**
>
> For each critical element, check if it appears in the user's readback.
> Score = Σ(matched elements × weight) / Σ(all weights)

## Instruction Type Weights

Weights reflect real-world consequences of errors:

| Instruction Type | Weight | Critical? | Rationale |
|---|---|---|---|
| Runway Assignment | 1 | ✓ | Wrong runway = potential collision |
| Altitude / Flight Level | 1 | ✓ | Altitude busts cause separation loss |
| Heading | 0.8 | ✓ | Heading deviations affect traffic flow |
| Frequency | 0.7 | ✓ | Wrong frequency = lost communication |
| Speed | 0.6 | — | Usually has margin for correction |
| Squawk Code | 0.5 | — | Incorrect but ATC will notice |
| QNH / Altimeter | 0.4 | — | Can be corrected before departure |
| Taxi Instructions | 0.6 | — | Ground speed allows correction |

## Difficulty Tiers

**SUPER EASY**

- 1 instruction
- Common callsigns
- Clear phraseology
- Standard situations

**EASY**

- 1–2 instructions
- Standard phraseology
- No conditionals
- Common airports

**MEDIUM**

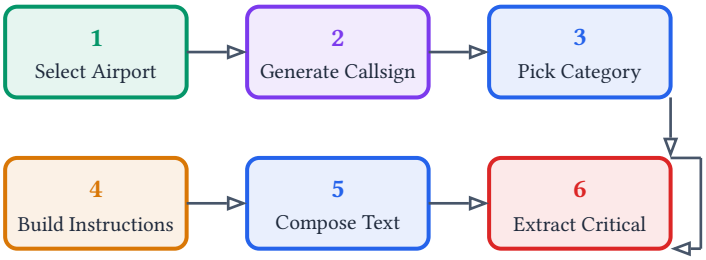
- 2–3 instructions
- Some complexity
- Varied callsigns
- Multiple elements

**HARD**

- 3–4 instructions
- Conditionals
- Amendments
- Rapid delivery

# Data Flow

## Generation Pipeline




### Generation Details

1. **Select Airport** — Random selection from 37 DACH airports
2. **Generate Callsign** — Using `callsign_format` templates matched to difficulty
3. **Pick Category** — Based on difficulty tier distribution
4. **Build Instructions** — 1–4 instructions using `instruction_type` definitions
5. **Compose Text** — Natural ATC phraseology
6. **Extract Critical** — Identify gradable elements with weights

**📦 Output**

Each generation creates:

- 1 × `radiocall` record
- N × `radiocall_instruction` records
- JSON `critical_elements` array

## Consumer Pipeline (Grading)

**🕯️ Integration Pattern**

Your frontend fetches a `radiocall` with nested relations, presents the audio/text, captures user input, then compares against `critical_elements` for scoring.

```
1. Fetch radiocall with instructions
   GET /items/radiocall/{id}?
fields=*,instructions.*,instructions.instruction_type.*

2. Present transmission (audio/text)

3. Capture user readback

4. For each critical_element:
```

```
   - Normalize user input (remove filler words, standardize numbers)
   - Check if element value appears
   - Apply weight to score

5. Calculate final score: matched_weight / total_weight

6. Match errors against common_error for feedback
```

# API Patterns

## Fetching with Relations

Get a radiocall with all data needed for grading:

```
GET /items/radiocall/{id}?fields=
  id,
  aircraft_callsign,
  full_transmission,
  expected_readback,
  critical_elements,
  difficulty,
  airport.icao_code,
  airport.name,
  instructions.sequence,
  instructions.raw_value,
  instructions.readback_text,
  instructions.instruction_type.code,
  instructions.instruction_type.grading_weight,
  instructions.instruction_type.is_critical
```

## Filtering by Difficulty

```
GET /items/radiocall?filter[difficulty][_eq]=medium&limit=10
```

## Random Selection

```
GET /items/radiocall?filter[category][_eq]=departure&sort=rand()&limit=1
```

## Category Distribution

```
GET /items/radiocall?aggregate[count]=id&groupBy[]=category
```

**Questions?** This schema is designed for extensibility.
Add new instruction types, callsign formats, or error patterns as needed.

Aevoli CMS · Schema v1.0