LocalSky Radar

Detailed product and engineering specification for a macOS desktop aircraft viewer with user location, configurable fallback airport, and ADS-B aircraft data.

Prepared for Claude Design, Codex, another AI coding agent, or a human developer.

Field	Value
Document date	June 15, 2026
Primary platform	macOS desktop app plus WidgetKit desktop widget
Primary purpose	Hobby aircraft awareness only. Not for navigation, safety, dispatch, ATC, or operational decisions.
Core range	100 nautical miles from the user's current location, or from a user-selected fallback airport.
MVP emphasis	Design and build the macOS desktop widget experience first, supported by a simple companion app for setup and full details.
Audio	Out of scope for v1 because live ATC stream embedding has legal and terms-of-use risk.

Key decision: the fallback airport must be user-configurable. Each person using the app can choose their own fallback airport by ICAO code, IATA code, airport name, or city search.

1. Brief For Claude Design And AI Agents

This document describes a macOS desktop widget and companion app for hobby aircraft watching. The desired product is not a website, not a marketing page, and not an aviation safety tool. It should feel like a calm, native macOS radar-style widget that helps a user glance at aircraft near their current location.

What we want built:

- A macOS desktop WidgetKit widget named LocalSky Radar that shows nearby aircraft within 100 nautical miles.
- A simple companion macOS app that handles first-time setup, location permission, fallback airport selection, provider configuration, and a larger radar/detail view.
- Current location is the preferred radar center when the user grants permission.
- A user-selected fallback airport is required. This is not hard-coded because the user's dad, brother, or anyone else should be able to choose their own airport.
- If location permission is denied, location fails, or location is stale, the app automatically uses the selected fallback airport.
- The experience should be beginner-friendly for the user: no coding knowledge, no technical API language in normal screens, and clear setup steps.
- The UI should be practical and compact: an aviation/radar look, but still native, readable, and calm on macOS.

Design intent:

- The widget should look like a premium desktop instrument: dark radar surface, subtle range rings, clear aircraft dots/icons, and restrained labels.
- The first view should be the working radar widget concept, not a landing page or product explainer.
- The widget must show data age, such as 'Updated 2 min ago' or 'Stale', because aircraft data can be delayed.
- Use plain user-facing language: 'Using current location', 'Using fallback airport', 'No aircraft nearby', and 'Open full radar'.
- Do not include live ATC audio, emergency language, owner identity features, or anything that implies certified radar.

Plain-English summary for another AI: build a native macOS aircraft-watching widget. It should center on the user's location when allowed, otherwise use an airport the user chose. The widget is a glanceable snapshot; the app handles setup and richer details.

2. Desktop Widget MVP

The MVP is the smallest version that proves the product works as a macOS desktop widget. It should be useful without audio, accounts, paid APIs, historical playback, or complex filters.

MVP area	Required behavior
Widget purpose	Show a quick snapshot of aircraft within 100 NM of the active radar center. The active radar center is current location when available, otherwise the user's fallback airport.
Widget sizes	Small and medium first. Small shows mini radar, aircraft count, closest aircraft, and update age. Medium shows mini radar plus a short nearest-aircraft list.
Companion app	Required for setup and full details. It asks for location permission, lets the user choose fallback airport, shows full radar, and stores settings.
Fallback airport	User chooses it during setup and can change it later. Search by ICAO, IATA, airport name, or city. Store coordinates locally.

MVP area	Required behavior
Aircraft details	Widget shows minimal details. Full app shows callsign, aircraft type, registration if available, altitude, speed, track, distance, source, and data age.
Refresh model	Full app can refresh conservatively, around every 30 seconds. Widget uses cached snapshots and WidgetKit timelines; it must not pretend to be second-by-second live radar.
Failure behavior	If no location or API data is available, show a useful state instead of a blank widget: setup needed, using fallback airport, no aircraft nearby, offline, or stale.
Click behavior	Clicking the widget opens the full app. If feasible, a widget aircraft row can deep-link to that aircraft in the full app.

MVP acceptance criteria:

- A non-coder can open the app, allow or deny location, choose a fallback airport, and add/use the widget.
- The widget never needs the user to know latitude/longitude unless they choose advanced settings.
- The widget displays nearby aircraft or a clear empty/error state on the desktop.
- The full app can show at least one clickable aircraft detail panel when aircraft are available.
- The app works if location is denied by using the selected fallback airport.
- The UI clearly shows whether it is using current location or fallback airport.

3. Executive Summary

LocalSky Radar is a macOS hobby app that shows nearby aircraft within 100 nautical miles. It uses the user's current location when permission is granted. If location is denied, unavailable, or stale, it falls back to an airport selected by the user. The widget gives a quick desktop glance; the full app provides the richer live radar and settings.

- The product should feel like a compact aircraft radar, but it must be labeled as a nearby aircraft viewer because ADS-B data is not complete, certified, or safety-grade.
- The first useful version should prioritize reliability: current location, fallback airport, map/radar, aircraft markers, details panel, refresh status, and clear error states.
- The widget should display a snapshot and open the full app when tapped. The full app can refresh more actively than the widget.
- No audio, owner lookup, private-person tracking, alerts, historical playback, or scraping should be included in v1.

4. User Responsibilities

- Provide the preferred default fallback airport during setup, such as KDFW, KLAX, KORD, KJFK, KATL, or another airport.
- Install and connect Xcode when implementation begins. This should be a native macOS project, not just a VS Code web project.
- Allow location permission during testing if the live location feature should be tested.
- Understand that location may be shared with the selected aircraft-data provider as coordinates for the aircraft search. The app should round or reduce precision where practical.
- Use the app for hobby viewing only. It should not be used for aviation safety, flight planning, navigation, air traffic control, or emergency decisions.

Recommendation for implementation setup: use Xcode for the macOS app and widget. VS Code can be helpful for notes or scripts, but Xcode is the right tool for SwiftUI, Core Location permissions, WidgetKit, signing, and

previewing the widget.

5. Product Scope

In scope for v1:

- macOS SwiftUI app with a full radar screen.
- WidgetKit desktop widget with compact aircraft snapshot.
- User location permission flow through Core Location.
- Fallback airport setting chosen by the user, not hard-coded.
- Airport search by ICAO code, IATA code, airport name, or city.
- Aircraft display within 100 nautical miles.
- Range rings at 25, 50, 75, and 100 nautical miles.
- Aircraft marker rotation by track over ground when available.
- Aircraft details on click: callsign, type, registration, altitude, speed, track, distance, data age, source, and route when available.
- Manual refresh plus conservative automatic refresh.
- Clear states for loading, no aircraft nearby, permission denied, API outage, stale data, and rate limits.

Out of scope for v1:

- Live ATC audio or embedded third-party streams.
- Aircraft owner/person lookup, celebrity/private jet tracking, or persistent tail-number stalking features.
- Commercial API resale, scraping flight-tracker websites, or bypassing provider terms.
- Historical playback, alerts, geofencing, multiple windows, offline maps, and app-store release automation.
- Safety-critical usage or claims that this is real radar.

6. Recommended Architecture

Build a native macOS app plus a WidgetKit extension. The app owns permissions, settings, API calls, airport search, and the richer map. The widget reads a prepared snapshot from shared App Group storage and opens the full app for interaction.

Layer	Responsibility	Implementation notes
macOS app	Full radar, settings, location permission, data fetching.	SwiftUI app target. Use Core Location for permission and MapKit or a custom SwiftUI radar view for display.
Widget extension	Desktop glance view.	WidgetKit timeline provider reads cached aircraft snapshot. Do not rely on the widget as a constantly running live process.
Location service	Choose radar center.	Use current location if authorized and fresh. If not, use selected fallback airport.
Airport service	Resolve fallback airport.	Bundle a trimmed OurAirports database or download/update it during development. Accept ICAO/IATA/name/city search.
Aircraft API client	Fetch and normalize aircraft data.	Start with Airplanes.live. Add OpenSky as a backup adapter. Keep provider-specific fields behind one app model.

Layer	Responsibility	Implementation notes
Shared storage	Share widget snapshot and settings.	Use App Group UserDefaults or a small JSON file in the App Group container.

Suggested internal data flow:

```
Location permission + fallback airport settings -> resolve radar center -> fetch aircraft -> normalize fields -> filter to 100 NM -> cache snapshot -> render full app and widget.
```

7. Location And Fallback Airport Behavior

Condition	Radar center behavior	User-facing state
Location allowed and fresh	Use current user coordinate.	Show 'Using current location' and last location update time.
Location denied	Use selected fallback airport.	Show 'Location off - using fallback airport' with a button to open settings.
Location unavailable or timed out	Use selected fallback airport.	Show 'Location unavailable - using fallback airport'.
No fallback selected yet	Prompt for airport choice before live fetch.	Show setup screen with airport search and examples.
Fallback airport not found	Keep last valid airport if available.	Show validation error and suggestions.

- The fallback airport must be editable at any time from Settings.
- The app should accept ICAO codes such as KDFW, IATA codes such as DFW, airport names, and city searches.
- When both location and airport are available, current location wins unless the user chooses a manual mode.
- Add a manual mode switch: Current Location, Fallback Airport, or Ask Each Launch. Default should be Current Location with fallback.
- Airport coordinates should come from a local or bundled airport dataset so fallback works even when the airport search network is down.

8. Widget Behavior

- The widget is a quick-glance desktop view, not the only interface.
- Widget sizes should include small and medium first. Large can be added later.
- Small widget: show mini radar rings, nearest 3 to 5 aircraft dots, closest callsign or count, and last updated time.
- Medium widget: show mini radar plus a short nearby aircraft list sorted by distance.
- Clicking the widget opens the full app. Clicking a specific aircraft in the widget can open the app focused on that aircraft if supported by the chosen WidgetKit APIs.
- The widget should never hide data age. Always show last updated time or 'stale'.
- Because WidgetKit controls refresh timing, the widget should read a cached snapshot prepared by the app and request reasonable future updates rather than trying to act like a live second-by-second radar.

9. Aircraft Data APIs

Primary recommendation: start with Airplanes.live because it provides a direct point-radius endpoint and fields that match this product. OpenSky is useful as a documented fallback but needs bounding-box math and has stricter quota

considerations. ADS-B Exchange is a paid optional upgrade.

Provider	Use	Important details
Airplanes.live	Primary v1 aircraft source.	Endpoint: https://api.airplanes.live/v2/point/{lat}/{lon}/{radius}. Radius supports up to 250 nautical miles. API guide says no SLA, no uptime guarantee, non-commercial use, and rate limit of 1 request per second.
OpenSky Network	Backup or alternate source.	Endpoint: https://opensky-network.org/api/states/all with lamin/lomin/lamax/lomax bounding box. Uses meters and m/s. OAuth2 client credentials for authenticated access. Anonymous and authenticated credit quotas apply.
ADS-B Exchange Community API	Optional paid richer source.	Developer Hub advertises non-commercial side-project use, 10,000 requests/month for the personal/community API, JSON responses, and location/hex/callsign queries. Use user's own key; do not ship a shared key.
FlightAware AeroAPI or similar	Future route enrichment only.	Use only on click if route/origin/destination becomes important, because route endpoints can be paid and rate-limited.

Airplanes.live field mapping:

App field	Airplanes.live field	Notes
id	hex	Mode S / ICAO identifier. May start with '~' for non-ICAO addresses.
callsign	flight	May be blank or missing.
registration	r	Available when database match exists.
aircraftType	t	ICAO type code such as A320, B738, C172.
latitude/longitude	lat/lon	Only show aircraft with valid position.
altitudeFeet	alt_baro or alt_geom	alt_baro can be 'ground'. Prefer barometric altitude when numeric.
groundSpeedKnots	gs	Already in knots.
trackDegrees	track	Track over ground, not always nose heading.
sourceType	type	Examples include adsb_icao, mlat, tisb_icao, other.
seenSeconds	seen or seen_pos	Use to fade or hide stale aircraft.
distanceNm	computed	Compute from radar center with haversine distance.

OpenSky field mapping:

App field	OpenSky index/property	Notes
id	states[n][0] icao24	Unique transponder address.
callsign	states[n][1] callsign	Can be null.
longitude	states[n][5]	Can be null.
latitude	states[n][6]	Can be null.
altitudeFeet	states[n][7] baro_altitude	OpenSky returns meters; convert to feet.
groundSpeedKnots	states[n][9] velocity	OpenSky returns m/s; convert to knots.
trackDegrees	states[n][10] true_track	True track clockwise from north. Can be null.
verticalRateFpm	states[n][11] vertical_rate	OpenSky returns m/s; convert to feet/minute.

App field	OpenSky index/property	Notes
source	states[n][16] position_source	0 ADS-B, 1 ASTERIX, 2 MLAT, 3 FLARM.

10. Airport Data

- Use OurAirports as the airport lookup source. Their downloads page provides CSV data for airports and states the data is public domain with no guarantee of accuracy or fitness for use.
- Bundle a trimmed airport file in the app for common airports, or bundle the full airports.csv if size is acceptable.
- Fields needed: ident, type, name, latitude_deg, longitude_deg, municipality, region, iso_country, iata_code, gps_code, local_code.
- Search should rank exact ICAO/gps_code matches first, exact IATA matches second, then airport name/city matches.
- Display results as 'KDFW - Dallas Fort Worth International Airport - Dallas-Fort Worth, US'.
- Persist the selected fallback airport by airport identifier and coordinates, not only by display name.

11. Radar Calculations

Calculation	Rule
Nautical miles to meters	meters = nauticalMiles * 1852
100 NM radius	185,200 meters
Distance filtering	Use haversine distance from radar center to aircraft coordinate. Do not trust square bounding boxes as circular range.
Bearing	Compute initial bearing from radar center to aircraft for list/detail display.
Marker rotation	Use trackDegrees when present. If missing, draw neutral aircraft/dot marker.
Stale filtering	Prefer aircraft with seen_pos <= 60 seconds. Fade 60-120 seconds. Hide older than 120 seconds unless provider behavior requires otherwise.
Sorting	Default list sorted by distance ascending. Tie-break by altitude descending or callsign.

12. Data Model

Normalize every provider response into one Aircraft model before the UI touches it. This prevents provider-specific field names from leaking into the app and makes it easier to add sources later.

```
Aircraft { id, callsign?, registration?, aircraftType?, latitude, longitude, altitudeFeet?, groundSpeedKnots?, trackDegrees?, verticalRateFpm?, sourceProvider, sourceType?, seenSeconds?, distanceNm, bearingDegrees, isOnGround?, routeText?, rawUpdatedAt }

RadarCenter { mode: currentLocation | fallbackAirport | manual, latitude, longitude, label, precision, updatedAt }

Airport { ident, icao?, iata?, name, municipality?, region?, country, latitude, longitude }
```

13. Settings

- Fallback airport: searchable field, required before live mode is considered ready.
- Location mode: Current Location with Fallback, Always Use Airport, or Ask Each Launch.
- Range: default 100 NM. Keep 100 NM for v1, but structure code to allow 25/50/100/150 later.

- Refresh interval: default 30 seconds in full app. Do not allow less than provider terms allow. Suggested minimum 15 seconds.
- Data provider: Airplanes.live first. OpenSky can be added later as an alternate option.
- Privacy display: show what coordinate is being used and whether it is current location or airport fallback.
- Reset button: clear saved location mode, fallback airport, and provider settings.

14. Privacy, Legal, And Safety Rules

- Ask for location permission using normal macOS Core Location permission flow. Include a clear location usage string in the app's plist.
- Store user location only locally and only as the latest radar center. Do not store location history in v1.
- Do not upload personal profile data. The only external request should be aircraft search coordinates and optional provider authentication.
- When possible, reduce coordinate precision before sending to APIs if exact home precision is not needed for a 100 NM search.
- Do not scrape Flightradar24, FlightAware website pages, or undocumented endpoints.
- Do not embed LiveATC or other audio streams in v1.
- Do not market as radar, navigation, safety, dispatch, ATC, or emergency tooling.
- Do not include aircraft owner identity, persistent aircraft tracking, or features designed to target private individuals.

15. Error States And Edge Cases

Case	Expected behavior
No location permission	Use fallback airport and show a calm status message.
No fallback airport selected	Show setup prompt before live fetch.
API rate limit / 429	Stop aggressive refresh, show retry message, keep last good snapshot with stale label.
API outage or no network	Show last good snapshot if available; otherwise show offline state.
No aircraft nearby	Show empty radar and 'No aircraft within 100 NM'.
Aircraft missing callsign/type	Show 'Unknown callsign' or 'Type unavailable' rather than failing.
Heading missing	Use a dot or neutral marker instead of a rotated aircraft icon.
Widget stale	Show 'Stale' or last updated age and invite user to open app.
Airport code ambiguity	Show search results and let user choose one.

16. Implementation Milestones

Milestone	Deliverable	Acceptance check
M1	Xcode macOS app with settings screen.	Can save fallback airport and display selected radar center.
M2	Airport search using bundled OurAirports data.	Typing KDFW, DFW, Dallas, or airport name returns usable choices.
M3	Location permission and fallback logic.	Denied location cleanly falls back to chosen airport.

Milestone	Deliverable	Acceptance check
M4	Radar UI with fake aircraft.	Range rings and clickable aircraft details work without API dependency.
M5	Airplanes.live integration.	Live aircraft appear within 100 NM and details are normalized.
M6	Refresh, stale data, and errors.	API failure does not crash the app; last good data remains labeled.
M7	Widget snapshot.	Widget shows aircraft count/nearest aircraft and opens full app.
M8	Polish and packaging preparation.	App is usable by a non-coder from launch through settings and viewing aircraft.

17. AI Implementation Notes

- Do not start by over-building. First make fake aircraft work, then wire live data.
- Keep provider code isolated. A future AI should be able to add OpenSky by creating another adapter without changing the UI.
- Use exact, typed Swift models for Aircraft, Airport, RadarCenter, and AircraftProviderError.
- Make the widget depend on cached snapshots, not direct high-frequency API polling.
- Use App Group storage for data shared between the macOS app and widget extension.
- Put provider URLs, refresh intervals, and radius settings in a configuration layer.
- Show units consistently: nautical miles, feet, knots, degrees.
- Label heading-like values as 'track' unless the data source explicitly provides true heading.
- Keep the user-facing language calm and non-technical: 'Using fallback airport' is better than 'Core Location authorization denied'.

18. Verification Plan

- Unit-test distance and bearing calculations with known coordinates.
- Unit-test airport search ranking for exact ICAO, exact IATA, and city/name matches.
- Unit-test provider normalization with saved sample Airplanes.live and OpenSky responses.
- Test location denied, location allowed, location unavailable, no fallback airport, and invalid fallback airport.
- Test with a busy airport fallback such as KDFW, KLAX, KORD, or KATL so the radar has visible aircraft.
- Test with a quiet rural airport to confirm empty states work.
- Test widget after app close to confirm the cached snapshot appears and stale state is clear.
- Check that no API key is committed to source code.

19. Source Notes

These sources were checked while writing this specification. The implementation should re-check terms and rate limits before release because API rules can change.

Source	URL
Airplanes.live REST API guide	https://airplanes.live/api-guide/
Airplanes.live ADS-B data fields	https://airplanes.live/rest-api-adsb-data-field-descriptions/
OpenSky REST API documentation	https://openskynetwork.github.io/opensky-api/rest.html
ADS-B Exchange Developer Hub	https://www.adsbexchange.com/community/developer-hub/
OurAirports open data downloads	https://ourairports.com/data/
Apple WidgetKit documentation	https://developer.apple.com/documentation/widgetkit
Apple WidgetKit update guidance	https://developer.apple.com/documentation/widgetkit/keeping-a-widget-up-to-date
Apple Core Location documentation	https://developer.apple.com/documentation/corelocation
Apple Core Location authorization	https://developer.apple.com/documentation/corelocation/requesting-authorization-to-use-location-services

Final implementation recommendation: connect Xcode when coding begins. Codex should create or modify the Xcode project, implement the SwiftUI app and WidgetKit extension, and keep the first version scoped to radar, location, fallback airport, and reliable aircraft data.