PARTNER INTEGRATION PLATFORM

partnermax

Technical Documentation

VERSION	v1.5.13
BUILD DATE	2026-06-29
OWNER	Azure S.r.l.
CLASSIFICATION	Confidential — partner review only
CONTACT	support@dealermax.app

DealerMAX is the Dealer Identity Layer — the infrastructure that applies the principles of AI Optimization to give automotive dealers a citable presence in the era of AI agents.

Contents

CHAPTER 01

Executive Brief

partnermax — Partner Integration Platform of the DealerMAX network

In one sentence

`partnermax` lets any automotive SaaS vendor (DMS, digital retailing platform, dealership tools provider) bring its own dealer network into the DealerMAX AI-citation infrastructure — Model Context Protocol server, ChatGPT Custom GPT, NLWeb `/ask`, structured JSON-LD surfaces — and resell the shared Italian long-term rental (NLT) catalog under its own brand, through a single REST API and Stainless-generated SDKs.

Why this exists

The dealer-side automotive market is at the inflection point where buyers no longer start the journey on a dealer website — they start by asking ChatGPT, Claude, Perplexity, or Gemini. Dealers that are not citable inside large language model answers are absent from the new top-of-funnel. Building this citation layer requires:

- A live MCP server compatible with MCP-capable clients that support Streamable HTTP transport
- A published ChatGPT Custom GPT available through the ChatGPT GPT surface, subject to OpenAI platform policies
- An NLWeb-compliant `/ask` endpoint with semantic search and grounded summaries
- Per-dealer structured data (JSON-LD), `llms.txt`, `sitemap-llm.xml`, `agents.md`
- C2PA-signed media for deepfake-resistant attribution

DealerMAX has shipped this stack to production for 27 Italian dealers and now exposes it as a partner-tier platform so software vendors can deliver the same surface to their customer base without rebuilding any of it.

What partners get

Provisioning	Create opaque partner dealer references through `client.dealers.create(...)` in the official SDKs, backed by `POST /api/partner/dealers` on `https://api.dealermax.app`. The partner keeps dealer anagraphics in its own system; DealerMAX stores only the partner-supplied external dealer id and the hidden technical tenant needed by stock/NLT tables.
NLT economics control	Configure per-dealer markup (0–10%) and three down-payment scenarios via `PATCH /v1/dealers/{id}/nlt-settings`. The shared Italian NLT catalog renders with your dealer's pricing applied.
Dealer-aware catalog reads	`GET /v1/dealers/{id}/nlt/offers` returns the network catalog already repriced for that specific dealer — up to 18 priced quotation cells per offer (3 durations × 6 km/year tiers) plus the three dealer-configured down-payment scenario amounts, ready to display.
Used-vehicle inventory	`POST /v1/dealers/{id}/vehicles` pushes a dealer's used-car stock into the network using the Motornet UNI code as canonical identifier. If the partner does not have its own Motornet catalogue access, it must use the paid targa/VIN resolver on `https://api.dealermax.app` first and then post the resolved code.
Cross-network AI surface, automatic	Your dealers and their inventory appear in `mcp.dealermax.app/mcp`, the published ChatGPT Custom GPT, NLWeb `/ask` semantic search, and the network `llms.txt` / `sitemap-llm.xml` with no additional integration effort.
SDKs day one	Python (`pip install partnermax`) and TypeScript / JavaScript (`npm install partnermax`) are generated and maintained via Stainless — the same toolchain used by OpenAI, Anthropic, and Cloudflare. Additional languages can be generated from the PartnerMAX OpenAPI contract during approved onboarding.

Standards posture

Surface	Standard
API specification	OpenAPI 3.1 (single source of truth in `docs/appendix/openapi.json`; shared during approved enterprise onboarding)
Errors	RFC 7807 Problem Details (`application/problem+json`)
Rate limiting	RFC 9239 draft headers + `X-RateLimit-*` alias headers
API lifecycle	RFC 8594 `Sunset` and `Deprecation` headers (180-day minimum window for breaking changes)
Idempotency	Stripe-style `Idempotency-Key` header on every mutating endpoint (24-hour retention)
Distributed tracing	W3C Trace Context (`traceparent`) propagated end-to-end
Transport	TLS 1.3 minimum, HSTS preload
Async notifications	No outbound webhooks in v1; partners poll read endpoints on their own schedule

Security and compliance posture

Data residency	Primary persistent storage is in the European Union — Supabase Frankfurt (AWS `eu-central-1`). LLM subprocessing boundaries and payload exclusions are documented in Security and Compliance.
Encryption	TLS 1.3 in transit, AES-256 at rest, SHA-256 hash of the plaintext for API key resolution (no salt; entropy of about 190 bits per key makes brute-force infeasible without salting).
Sub-processors	Supabase (DB + storage, SOC 2 Type 2), Stripe (billing, PCI-DSS Level 1 + SOC 2 Type 2), Postmark (transactional email, SOC 2), Railway (compute, SOC 2), Cloudflare (CDN + DDoS, ISO 27001 + SOC 2), Sentry (error tracking, SOC 2), OpenAI (LLM inference, zero-retention DPA).
GDPR	Azure S.r.l. is data controller for network operations; partner is data controller for the dealer data it provisions. DPA template available on request. Data subject rights flow documented, 30-day response window.
Audit logging	Every mutating operation recorded with timestamp, partner_id, endpoint, request_id, trace_id, source IP, response status. 12-month retention minimum. Partner-accessible export.
Breach notification	Within 72 hours of awareness, per GDPR Article 33.
Compliance roadmap	SOC 2 Type 1 readiness target Q3 2026. ISO 27001 readiness assessment in roadmap. Independent penetration test cadence: annual third-party review.

Full posture: `docs/11-security-compliance.md`.

Commercial model

Billing target	Partner only. Dealers under the partner do not bill DealerMAX directly.
Plan structure	Flat monthly subscription per partner, currency EUR, billed through DealerMAX. No per-dealer fee, no per-API-call metering in v1.
API access gating	An active DealerMAX subscription is required. Payment failure starts a 14-day grace window; after that, every authenticated call returns `402 Payment Required` until billing is restored.
Dealer cascade	When a partner subscription is canceled, dealers owned by that partner are deactivated automatically across the cross-network AI surfaces. Reactivation is manual via `PATCH /v1/dealers/{id}` after subscription restore.
NLT economics	Partner is free to set agency markup independently per dealer (0–10% over network base canon). DealerMAX does not take commission on NLT contracts closed by partner dealers — partner is responsible for its own commercial relationship with its dealers and with the NLT provider network.

Audience

This document is sized for the first 15 minutes of a technical introduction. It is intentionally not a sales document. Readers wanting deeper detail can jump to:

- **Integration engineers**: `02-getting-started.md` — first API call in under 10 minutes
- **Solution architects**: `01-architecture-overview.md` — system design with C4 and sequence diagrams
- **Security review**: `11-security-compliance.md` — threat model, sub-processors, GDPR
- **Procurement**: `10-sla.md` — uptime SLO, support tiers, escalation matrix
- **Legal**: see [`LICENSE`] for the dual-license model

Network signals

	Source
Live MCP server	`mcp.dealermax.app/mcp` (Streamable HTTP transport, 7 tools, official MCP Registry + server-card discoverable)
Published ChatGPT Custom GPT	`gpt.dealermax.app` -> ChatGPT GPT surface
NLWeb `/ask` (Microsoft 2025 spec)	`apimax.azcore.it/public/ask` (list + summarize modes, gpt-5-mini grounding)
C2PA root certificate	`apimax.azcore.it/.well-known/c2pa-root-ca.pem`
OpenAPI Custom GPT spec	`apimax.azcore.it/.well-known/openapi-custom-gpt.json`
Dealer JSON-LD coverage	Per dealer via `/public/render` — Organization, AutoDealer, Vehicle, Car, Offer, FAQPage, Article, AudioObject, DefinedTerm
Active dealer base	27 production dealers
Monthly MCP queries	Not disclosed in the public brief; available under NDA during enterprise review.
Monthly Custom GPT invocations	Not disclosed in the public brief; available under NDA during enterprise review.

The endpoint rows above point to live, production surfaces today. Usage metrics that are not disclosed in this public brief are provided only during enterprise review with an agreed measurement window.

Getting started

PartnerMAX has two onboarding paths:

Path	Use when	Access model
Standard Trial Path	The partner wants to evaluate the SDK/API before procurement.	Self-service signup at `https://developers.dealermax.app/signup`, verified company email + Partita IVA, 15-day live trial, limited enterprise access, no custom SLA or contractual commitments until conversion.
Enterprise Onboarding Path	The partner needs procurement, security review, custom quotas, DPA review, or commercial terms before production rollout.	Commercial agreement with DealerMAX, support-managed credentials, sandbox + production enablement, custom quota/SLA review, and named technical contact.

Full journey: `12-onboarding-journey.md`.

Document version	1.5.13
Last updated	2026-06-29
Owner	Azure S.r.l.
Classification	Confidential — partner review only
Contact	support@dealermax.app

CHAPTER 02

Architecture Overview

This document describes the technical architecture of `partnermax`, the integration boundary between a third-party SaaS vendor and the DealerMAX network. Audience: solution architects, integration engineers, security reviewers.

Contents

System context (C4 Level 1)

```
graph TB
    Partner["<b>Partner SaaS</b><br/>DMS, digital retailing<br/>automotive platform"]
    PartnerDealer["<b>Partner-owned dealer</b><br/>Italian automotive<br/>retail business"]
    EndUser["<b>End consumer</b><br/>Vehicle buyer<br/>NLT prospect"]
    AIClient["<b>AI client</b><br/>ChatGPT, Claude<br/>Perplexity, Gemini"]

    Partnermax["<b>partnermax</b><br/>Partner Integration Platform<br/>developers.dealermax.app"]
    DealerMaxNetwork["<b>DealerMAX network</b><br/>Shared NLT catalog<br/>AI-citation surfaces"]

    Partner -->|"Provision dealers<br/>Configure NLT<br/>Read catalog"| Partnermax
    Partnermax -->|"Reads + writes<br/>same Supabase tables"| DealerMaxNetwork
    PartnerDealer -->|"Operate sales<br/>Receive leads"| Partner
    EndUser -->|"Browse / inquire"| PartnerDealer
    AIClient -->|"Search via MCP / GPT / NLWeb"| DealerMaxNetwork
    EndUser -->|"Asks AI"| AIClient

    style Partnermax fill:#FDB825,stroke:#000,color:#000
    style DealerMaxNetwork fill:#fff,stroke:#000
```

Key relationships:

- The partner integrates server-to-server with `partnermax` using an API key. No direct partner-to-DealerMAX-network HTTP traffic exists; all interactions route through `partnermax`.
- Partner-owned dealers do not interact with `partnermax` directly. The partner is responsible for the dealer-facing experience and propagates settings via this API.
- End consumers reach the partner's dealers through any channel — the partner's dealer-facing websites, the AI-citation surfaces, or organic discovery.

- AI clients (ChatGPT, Claude Desktop, Cursor, Gemini, Perplexity) discover and cite the partner's dealers through DealerMAX-operated MCP, Custom GPT, and NLWeb endpoints. No partner-side configuration required — indexing happens via background workers (`azurenet-engine`) that read the shared database.

Container view (C4 Level 2)

```
graph TB
    subgraph PartnerEnv["Partner Environment"]
        PartnerBackend["Partner Backend<br/>(server-to-server)"]
        PartnerSDK["partnermax SDK<br/>Python<br/>or raw HTTP"]
        PartnerBackend -->|uses| PartnerSDK
    end

    subgraph PartnerMaxLayer["PartnerMAX public API + docs"]
        Portal["Developer portal<br/>docs, SDK guides, gated contracts"]
        PartnerV1["/v1 stock + NLT<br/>dealer-scoped operations"]
    end

    subgraph SharedData["Shared Data Plane"]
        Supabase[("Supabase PostgreSQL<br/>EU Frankfurt<br/>shared with all services")]
    end

    subgraph DMaxNetwork["DealerMAX Network (independent services)"]
        Apimax["apimax<br/>apimax.azcore.it<br/>(public read SSR)"]
        CoreApi["api.dealermax.app<br/>(selected routes implemented by core_api_v2)"]
        Mcp["MCP server<br/>mcp.dealermax.app/mcp<br/>(mounted on apimax)"]
        Engine["azurenet-engine<br/>(async workers)"]
    end

    subgraph External["External"]
        Stripe["Stripe<br/>(subscription lifecycle)"]
        OpenAI["OpenAI<br/>(NLWeb summarize)"]
        Sentry["Sentry<br/>(error tracking)"]
    end

    PartnerSDK -->|/api/partner signup, credits, dealer refs| CoreApi
    PartnerSDK -->|/v1 vehicle and NLT APIs| PartnerV1
    Portal -->|Publishes docs and SDK guidance| PartnerSDK
    PartnerV1 -.->|"Dealer-scoped stock/NLT writes"| Supabase

    Apimax -.->|Reads same tables/views| Supabase
    CoreApi -.->|Owns platform writes| Supabase
    Mcp -.->|Reads same tables/views| Supabase
    Engine -.->|Writes async results| Supabase

    Portal -.->|Error events| Sentry
    Apimax -.->|"LLM summarize<br/>mode"| OpenAI

    style PartnerMaxLayer fill:#FDB825,stroke:#000,color:#000
    style SharedData fill:#f5f5f5,stroke:#000
```

Service boundaries (enforced by design):

Service	Reads	Writes	HTTP-callable by
`partnermax`	Developer portal content, gated contracts, partner dealer registry reads, vehicle/NLT shared tables	Partner vehicle inventory, photos, accessories, and NLT settings; no identity, billing, credits, or dealer-anagraphic ownership	Partner SaaS, approved onboarding
`apimax`	Public read views + per-dealer SSR data	None (read-only by design)	End-user browsers, AI clients, partner dealers' websites
`core_api_v2`	All SaaS tables	Partner onboarding, subscription/key lifecycle, credits, paid Motornet targa/VIN lookup, and `partner_dealers` technical registry	Internal implementation behind selected `api.dealermax.app` paths
MCP server	Public read views	None	AI clients via Streamable HTTP (mounted on `apimax`)

Services never call each other over HTTP. Partner SDKs call the single public host `https://api.dealermax.app`; that host exposes `/api/partner/*` for partner-registry operations and `/v1/*` for partner vehicle/NLT operations. `core_api_v2` is an internal implementation name in this document, not a partner-facing DNS name. `partnermax` must not proxy or duplicate identity, Stripe, credit, or lookup rules. The shared Supabase database remains the internal data plane.

Data model

Partner accounts live in the existing `utenti` table with `role='partner'`. Dealer identity for PartnerMAX is intentionally opaque: the partner sends an `external_dealer_id`, and `core_api_v2` stores the mapping in `partner_dealers`. The hidden `dealer_user_id` exists only because stock tables need an internal `utenti.id`; its `parent_id` stays in the admin/root tree. PartnerMAX never treats `utenti.parent_id = partner.id` as the new ownership model.

```
erDiagram
    utenti ||--o{ utenti : "parent_id<br/>(self-reference)"
    utenti ||--o{ dealer_public : "owner_user_id"
    utenti ||--o{ dealer_public_api_keys : "dealer_id (= utenti.id)"
    utenti ||--o{ partner_dealers : "partner_user_id"
    utenti ||--o{ partner_dealers : "dealer_user_id"

    utenti {
        bigint id PK
        text email UK
        text role "admin | partner | dealer | dealer_team | ..."
        bigint parent_id FK "hierarchy"
        text dmax_subscription_status "active | trialing | past_due | canceled | ..."
        text stripe_dmax_subscription_id
        boolean dealer_disabled "terminal dealer deletion flag"
        timestamp created_at
        timestamp updated_at
    }
    dealer_public {
        bigint id PK
        bigint owner_user_id FK
        text ragione_sociale
        text brand_name
        text citta
        text provincia
        text telefono
        boolean is_active
        smallint nlt_agency_percent "0-10"
        jsonb nlt_anticipi_config "[{pct, eur}, {pct, eur}, {pct, eur}]"
        text nlt_image_mode "branded | scenario_locked | scenario_seasonal"
        text nlt_image_scenario_locked "mediterraneo | cortina | milano | showroom | null"
        timestamp created_at
        timestamp updated_at
    }
    dealer_public_api_keys {
        bigint id PK
        bigint dealer_id FK "= utenti.id (schema column name; partnermax exposes it as user_id in Python via a SQLAlchemy s
        text key_hash "SHA-256(plaintext) — no salt, byte-compatible with apimax and core_api_v2"
        text key_prefix "for safe logging, e.g. pmk_live_abc"
        text label
        jsonb capabilities "['can_issue_keys']"
        boolean is_active
        timestamp created_at
        timestamp last_used_at
        timestamp expires_at
    }
    partner_dealers {
        bigint id PK
        bigint partner_user_id FK
        text external_dealer_id "partner-supplied opaque id"
        bigint dealer_user_id FK "hidden technical dealer"
        text status "internal: active | suspended | revoked; API: active | inactive | deleted"
        jsonb metadata
        boolean public_surfaces_enabled
        timestamp created_at
        timestamp updated_at
    }
```

Hierarchy invariant: `utenti.parent_id` remains the platform hierarchy, not the PartnerMAX ownership model.

```
admin (id=13, network root)
   |-- partner (id=N, role=partner, parent_id=13)
   |-- dealer technical tenant (id=M1, role=dealer, parent_id=13 or configured admin)
   |-- native DealerMAX dealer (id=Mx, role=dealer, parent_id=13)
   |-- ...

partner_dealers(partner_user_id=N, external_dealer_id="verdicar-001", dealer_user_id=M1)
```

A real-world dealer can appear under multiple partners, and can also register independently in DealerMAX. DealerMAX does not merge those identities from partner data. The partner owns the anagraphic record in its own system; the platform stores only the opaque external id and the technical tenant needed by existing stock tables.

NLT tenant context continues to depend on the admin hierarchy. Technical partner dealers therefore stay under the configured admin/root parent so NLT catalog and pricing logic keep using the production admin context.

API key resolution:

1. Extract the raw key from the `X-Api-Key` header (preferred) or `Authorization: Bearer <key>` header.
2. Compute `key_hash = sha256(key)` — plain SHA-256, **no salt**. Same scheme as `core_api_v2` and `apimax`; the same row in `dealer_public_api_keys` is readable by every service.
3. Look up the active row in `dealer_public_api_keys` by `key_hash`.
4. Load the owning user from `utenti` (joined on `dealer_id = utenti.id`).
5. Verify `dmax_subscription_status` is acceptable: `active` / `trialing` → full access; `past_due` / `unpaid` within the configurable grace window (default 14 days, measured from `utenti.updated_at` as a proxy until a dedicated `subscription_changed_at` column ships); `canceled` / `incomplete_expired` / `incomplete` → reject with `402`.
6. Identify role: `partner` → multi-dealer scope (this is the only role partnermax authenticates today). Other roles are rejected with `401 invalid_api_key`.

Sequence: partner dealer reference creation

Partner dealer provisioning is exposed publicly as `POST /api/partner/dealers` on `https://api.dealermax.app` and implemented by the DealerMAX internal platform. PartnerMAX `/v1` does not create dealer anagraphics; it consumes the opaque `external_dealer_id` returned by that registry.

```
sequenceDiagram
    autonumber
    participant P as Partner Backend
    participant S as api.dealermax.app<br/>(internal partner registry)
    participant D as Supabase
    participant E as azurenet-engine<br/>(async indexers)

    P->>S: POST /api/partner/dealers<br/>{external_dealer_id, activate, metadata}<br/>X-Api-Key, Idempotency-Key
    S->>S: Resolve partner from API key
    S->>S: Verify subscription active
    S->>D: BEGIN TRANSACTION
    S->>D: INSERT partner_dealers (external_dealer_id, status)
    S->>D: INSERT hidden technical utenti row<br/>(role='dealer', parent_id=admin/root)
    S->>D: INSERT minimal technical publication/settings row
    S->>D: COMMIT
    D-->>S: external_dealer_id, status, technical tenant linked
    S-->>P: 201 Created<br/>{dealer_id: external_dealer_id, status, public_surfaces_enabled}
    Note over P,S: Partner receives synchronous confirmation
    par Async indexing (decoupled, no SLA in v1)
        E->>D: Periodic scan: new active dealers
        E->>D: Refresh apimax read views, sitemap.xml, llms.txt
        E->>D: Bootstrap mcp_dealer_public row (gates MCP visibility)
    end
```

Latency targets:

- API response (steps 1-10): p95 < 1000 ms.
- Public/AI surface activation is separate from the technical dealer reference. A partner dealer created with `activate=false` is available in the partner registry but remains invisible to PartnerMAX `/v1` stock/NLT reads until it is activated.

Idempotency: if the partner retries with the same `Idempotency-Key` within 24 hours and an identical request body, the original 201 response is returned. The DB transaction is not re-executed.

Sequence: dealer-aware NLT read

```
sequenceDiagram
    autonumber
    participant P as Partner Backend
    participant S as partnermax
    participant D as Supabase
    participant C as NLT calculator<br/>(in-process)

    P->>S: GET /v1/dealers/{id}/nlt/offers/{offer_id}<br/>X-Api-Key
    S->>S: Resolve partner from API key
    S->>S: Resolve dealer through partner_dealers.external_dealer_id (ACL)
    S->>D: SELECT nlt_offerte WHERE id_offerta=?<br/>JOIN mnet_dettagli, nlt_pneumatici,<br/>nlt_autosostitutiva
    S->>D: SELECT nlt_quotazioni WHERE id_offerta=?<br/>(18 columns: 3 durations × 6 km tiers)
    S->>D: SELECT nlt_agency_percent, nlt_anticipi_config,<br/>nlt_image_mode, nlt_image_scenario_locked<br/>FROM dealer_pu
    S->>D: SELECT mnet_modelli_ai_foto / dealer_branded_images<br/>(per dealer image_mode)
    D-->>S: Offer + quotations + dealer settings + assets
    S->>C: For each (duration, km) cell, apply markup + high tier
    C-->>S: Up to 18 priced cells + 3 down-payment scenarios
    S-->>P: 200 OK<br/>{offer detail with 29+ fields, identical to apimax MCP shape}
```

Computation model (in-process, deterministic, no external dependencies):

```
listino_imponibile  = prezzo_listino / 1.22
provvigione         = listino_imponibile × (agency_markup_percent / 100)
anticipo_tier_eur   = listino_imponibile × (tier.percent_of_list / 100) + tier.fixed_eur
canon_imponibile    = base_canon + provvigione / duration_months
                           − anticipo_tier_eur / duration_months
if offer.solo_privati:
    canon = canon_imponibile × 1.22
else:
    canon = canon_imponibile
```

Implemented in [`app/services/nlt_calculator.py`] as a 1:1 port of `apimax/app/services/nlt/calculator.py` — same formula, same rounding, same result. CLAUDE.md forbids cross-service Python imports, so the calculator is duplicated locally and kept in lockstep by hand.

ACL check is mandatory: step 3 rejects with `403 forbidden_dealer_not_owned` if the requested `dealer_id` does not belong to the authenticated partner. This is the single non-negotiable authorization gate.

Sequence: subscription cancellation cascade

```
sequenceDiagram
    autonumber
    participant SC as Stripe
    participant S as partnermax<br/>/v1/stripe/webhook
    participant D as Supabase
    participant API as partnermax<br/>API endpoints
    participant V as Cross-network<br/>AI views

    SC->>S: POST customer.subscription.deleted<br/>{customer, subscription_id, status: canceled}
    S->>S: Verify Stripe HMAC signature
    S->>D: UPDATE utenti<br/>SET dmax_subscription_status='canceled'<br/>WHERE stripe_dmax_subscription_id=?
    S->>D: UPDATE dealer_public<br/>SET is_active=false<br/>WHERE owner_user_id IN<br/>(SELECT id FROM utenti WHERE parent_
    Note over D: Cascade complete. Partner-owned dealers<br/>are hidden from public/AI surfaces.
    S-->>SC: 200 OK

    Note over API,V: Subsequent partner API calls fail auth
    API->>D: SELECT key + utenti by key_hash
    D-->>API: subscription_status='canceled'
    API-->>API: Reject with 402 subscription_inactive

    Note over V: Cross-network AI surfaces stop returning dealers
    V->>D: SELECT public dealer views<br/>WHERE dealer_public.is_active=true
    Note over V,D: Dealers disappear from MCP, Custom GPT,<br/>NLWeb, llms.txt within the next index refresh
```

Cascade timing:

- Stripe → `partnermax` webhook: typically 1–5 seconds.
- Database cascade (steps 4–5): synchronous within the webhook handler, completes before returning 200 to Stripe.
- API rejection at the next request: immediate (subscription check happens on every authenticated call).
- AI-surface visibility loss: at next view refresh, typically within 5 minutes once the indexing workers pick up the change.

Reactivation: when the partner restores payment, Stripe sends `customer.subscription.updated` with `status: active`. The webhook handler flips `utenti.dmax_subscription_status` back to `active`. **Dealers must be manually re-activated** by the partner via `PATCH /v1/dealers/{id}` with `{"status": "active"}` — between cancellation and restore the partner may have lost some of its underlying business relationships, so re-activation is an explicit per-dealer decision. Reactivation flips `dealer_public.is_active` back to true; `utenti.dealer_disabled` remains reserved for terminal dealer deletion.

Grace period for `past_due` and `unpaid` is configurable (default 14 days). It is measured against `utenti.updated_at` as a proxy until a dedicated `dmax_subscription_status_changed_at` column ships in a future migration; within the window the API responds normally, after the window the API returns `402 grace_period_expired`.

Sequence: AI-citation surface indexing

This is informational — the partner does not interact with this flow directly. It explains why provisioned dealers will appear in AI surfaces once the worker pipelines are enabled.

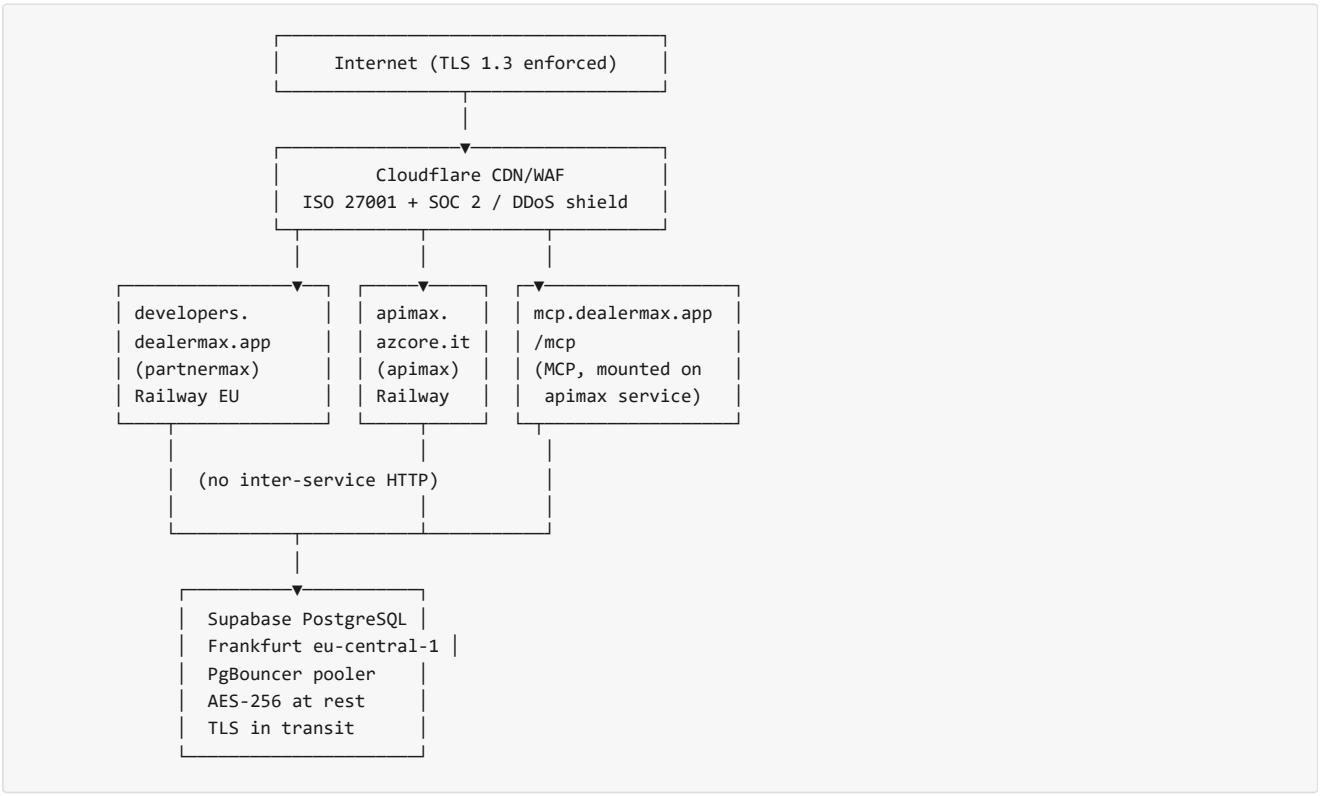
```
sequenceDiagram
    participant Pmax as partnermax<br/>(provisioning endpoint)
    participant DB as Supabase
    participant Eng as azurenet-engine<br/>(workers)
    participant Apimax as apimax service
    participant Mcp as MCP server<br/>(mounted on apimax)
    participant GPT as Custom GPT Actions<br/>(on apimax)
    participant NLW as NLWeb /ask<br/>(per dealer)

    Pmax->>DB: INSERT dealer_public (is_active=true)
    Note over DB: Public read views over base tables<br/>reflect the new dealer immediately.
    Eng->>DB: Periodic scan for new dealers
    Eng->>DB: INSERT mcp_dealer_public (enabled=true)<br/>Refresh sitemap, llms.txt, agents.md fragments
    Eng->>DB: Bootstrap per-dealer NLWeb corpus

    par Per AI client (steady state)
        Mcp->>DB: Tool call: search_vehicles / find_dealer
        DB-->>Mcp: Includes new dealer (where mcp_dealer_public.enabled)
        Mcp-->>Mcp: Returns to AI client (Claude, Cursor, ChatGPT)
    and
        GPT->>DB: Custom GPT Actions (apimax.azcore.it/public/custom-gpt/*)
        DB-->>GPT: Includes new dealer
        GPT-->>GPT: Returns to ChatGPT user
    and
        NLW->>DB: Semantic search over the dealer's corpus
        DB-->>NLW: Includes new dealer
        NLW-->>NLW: Optional gpt-5-mini summarize
    end
```

Why no provisioning hook into AI services: every AI-facing endpoint reads from the same Supabase tables/views, which read from the same base tables. There is no separate “search index” to update. A new row in `dealer_public` is visible to all consumers on the next query. Cached content (1-60 minute TTLs depending on surface) is the only delay once the gating row in `mcp_dealer_public` is present.

Network topology



Additional supporting services (out of the partner's data path):

- `api.dealermax.app` (selected routes implemented by `core_api_v2`) - public PartnerMAX API DNS for partner registry and platform services. Partners call only `https://api.dealermax.app`; `core_api_v2` is the internal implementation name behind selected paths, not a partner-facing service name.
- `gpt.dealermax.app` — 302 redirect to the ChatGPT Custom GPT URL.
- `azurenet-engine` Railway worker — async indexing, no inbound HTTP from partner.

Technology stack

Layer	Technology	Rationale
Language	Python 3.12	Consistent with the rest of the platform (`apimax`, `core_api_v2`). Mature FastAPI ecosystem.
Framework	FastAPI 0.115+	Async, OpenAPI 3.1 native, Pydantic 2 integration.
ORM	SQLAlchemy 2.0	Same metadata-style mappings used across services.
Database	Supabase PostgreSQL (managed)	EU residency, PITR backups, single shared instance for the platform.
Idempotency cache	Postgres table `partnermax_idempotency` (24h TTL, Stripe-compatible)	Same primary-data plane as the rest of the stack — no extra HA story needed.
Rate-limit state	Redis-backed fixed-window counters in `app/middleware/rate_limit.py` with fail-open behavior if Redis is unavailable.	Per-key counters scale horizontally; bucket reset is wall-clock predictable.
Subscription billing	Stripe	Industry standard.
Error tracking	Sentry	SOC 2; same project pattern as `apimax` / `core_api_v2`.
Logging	structlog → stdout → Railway log aggregation	Structured JSON logs, W3C `traceparent` propagation.
Deployment	Docker image on Railway	Same pattern as the rest of the platform.
CI/CD	GitHub Actions	Standard, integrated with Stainless SDK regeneration.
SDK generation	Stainless	Same toolchain as OpenAI, Anthropic, Cloudflare.
Status page	Live	Partner-facing service health at `developers.dealermax.app/status`, backed by independent infrastructure monitoring.

Deployment model

Environment	Host	Key prefix	Billing mode
Local	`127.0.0.1:8080`	`pmk_test_…`	Test
Sandbox	`api.dealermax.app` (same host)	`pmk_sand_…`	Test
Production	`api.dealermax.app`	`pmk_live_…`	Live

There is no separate sandbox host. The environment is encoded in the API-key prefix and resolved server-side. A `pmk_sand_…` key runs against an isolated billing-provider test-mode tenant and against a separate dataset (sandbox-prefixed dealers are not surfaced on production AI clients); everything else (URL, paths, payloads) is identical to production.

Deployment flow:

```
graph LR
    Dev[Developer<br/>local commit] -->|git push| GH[GitHub<br/>main branch]
    GH -->|webhook| CI[Railway<br/>auto-build]
    CI -->|health check OK| Deploy[Replace running<br/>container, zero downtime]
    Dev -->|upload spec via API| ST[Stainless<br/>regenerate SDKs]
    ST -->|auto PRs| SDKRepo[partnermax-python / partnermax-node<br/>release-please]
    SDKRepo -->|on merge| Registries[PyPI + npm<br/>partnermax]
```

Zero-downtime deploys via Railway's blue/green container swap. SDK regeneration is triggered manually by uploading the latest OpenAPI spec to Stainless (`POST /api/spec`); Stainless generates the Python and TypeScript SDKs, opens release-please PRs on `partnermax-python` and `partnermax-node`, and CI publishes `partnermax` to PyPI and npm on merge.

Design principles and constraints

The architecture is informed by these explicit choices:

1. **Shared data plane, isolated services.** Services never call each other over HTTP. The Supabase database is the only integration boundary. This was a foundational decision of the DealerMAX platform and is non-negotiable.
2. **Single internal owner for identity and billing.** `core_api_v2` owns partner onboarding, subscription/key lifecycle, credits, paid Motornet targa/VIN lookup, and the `partner_dealers` registry. `partnermax` owns the partner-facing vehicle/NLT API layer and must reuse those platform primitives instead of duplicating them.
3. **Partner dealer registry via `partner_dealers`.** `utenti.parent_id` stays reserved for the platform admin hierarchy and NLT context. The partner relation is `(partner_user_id, external_dealer_id, dealer_user_id)` in `partner_dealers`.
4. **The DealerMAX subscription is the ground truth for access.** API key validity is derived from subscription status, not stored as a separate flag. There is exactly one place where access is revoked: the billing-provider cancellation webhook.
5. **Cascade deactivation is automatic, reactivation is manual.** Loss of subscription suspends active `partner_dealers`, hides their stock, and revokes partner keys. Recovery is explicit through the partner-registry dealer/key flows because business relationships may have changed between billing events.
6. **OpenAPI as the single source of truth.** SDKs are generated from the PartnerMAX OpenAPI contract, the Postman collection is checked in, and the enterprise spec is shared during approved onboarding. The unrestricted public portal exposes the Dealer Public API contract separately.
7. **Standards over invention.** RFC 7807 (Problem Details), RFC 8594 (Sunset / Deprecation), RFC 9239 (`RateLimit-*` headers), W3C `traceparent`, Stripe-style `Idempotency-Key`. Enterprise integration teams should never need to learn a new convention.
8. **EU persistent data residency.** Customer business data is stored in Frankfurt. Exposure to US-headquartered subprocessors is governed through DPAs, SCCs, payload minimization, encryption, audit logging, and contract-specific controls where required.
9. **No commercial broker positioning.** `partnermax` revenue comes only from the partner subscription. DealerMAX does **not** take commission on NLT contracts closed by partner dealers. The partner sets its own markup (capped at 10%) and runs its own commercial relationships downstream.
10. **No consumer-facing chatbot.** The AI surface is for AI clients (LLMs that consume MCP / Custom GPT / NLWeb), not for embedded chat widgets on partner-dealer sites. This is a brand and product positioning constraint, not just a technical one.

Related documents

- `02-getting-started.md` — First API call walkthrough.
- `03-authentication.md` — API key lifecycle and subscription gating.
- [`04-api-reference/`] — Per-endpoint reference (auth + keys, dealer management, NLT settings, NLT catalog).

- `11-security-compliance.md` — Security posture, sub-processors, GDPR.
- [`appendix/openapi.json`] — Machine-readable specification (the contract Stainless reads to generate the SDK).

CHAPTER 03

Getting Started

> *Make your first API call in under 10 minutes.*
>
> *All partner API calls in this guide go to the single partner base URL `https://api.dealermax.app`. Onboarding, credits, Motornet lookup, dealer registry, and vehicle stock are exposed through this public host. Internal service names are not part of the partner contract.*
>
> *Dealer-reference creation is exposed in the SDK as `client.dealers.create(...)`. Under the hood the generated client calls `/api/partner/dealers` on the same host; you do not need to write raw HTTP for dealer provisioning.*

This guide walks you through provisioning your first dealer on the DealerMAX network, configuring its NLT (long-term rental) economics, and reading the dealer-aware catalog — end to end.

By the end of this guide you will have:

1. Set up your Partner API key.
2. Registered an opaque dealer reference under your partner account.
3. Configured per-dealer NLT mark-up, down-payment tiers, and cover-image rendering mode.
4. Fetched the dealer-aware NLT catalog (listing + one detail).

All examples target the production host `https://api.dealermax.app`. There is **no separate sandbox host**: the environment is encoded in the API-key prefix (`pmk_live_…`, `pmk_sand_…`, `pmk_test_…`) and resolved on the server. A sandbox-prefixed key issued by DealerMAX support runs against an isolated Stripe-test-mode tenant and an isolated dataset; everything else (host, paths, payloads) is identical to production.

Authenticated PartnerMAX reads are served with `Cache-Control: no-store` and CDN no-store headers, with `Vary: Authorization, X-Api-Key`, so listing/detail calls immediately reflect writes such as vehicle-image uploads.

Prerequisites

Requirement	How to obtain
An active partner account on the DealerMAX network	Choose one path. Standard Trial Path: self-service signup at `https://developers.dealermax.app/signup` with verified email and Partita IVA; the trial key is shown once. Enterprise Onboarding Path: signed commercial agreement, support-managed account provisioning, and secure key delivery by DealerMAX.
An active DealerMAX subscription on your partner account	API access is gated on the subscription state, which reduces to three buckets for the integrator: **Active** (full access), **Grace** (14-day recovery window after a payment failure, still full access), **Inactive** (`402 Payment Required` on every call). See Authentication → Subscription lifecycle for the full mapping.
A Motornet identifier for each used vehicle	Every SDK vehicle create needs a `motornet_code`. Get it from your own Motornet data subscription (Motornet banche dati), or send DealerMAX the vehicle plate/targa or VIN/telaio through the paid resolver. The resolver consumes PartnerMAX credits and returns the Motornet UNI code to pass to `client.dealers.vehicles.create(...)`.
`curl` 7.68+ or any HTTP client	Pre-installed on macOS, most Linux distributions, and Windows 10+. Official SDKs are published for Python (`pip install partnermax`) and TypeScript / JavaScript (`npm install partnermax`).

Step 1 — Use the Partner API key

The plaintext Partner API key is shown **exactly once**. In the Standard Trial Path it is returned at the end of the self-service signup flow. In the Enterprise Onboarding Path it is minted by DealerMAX support and delivered through the agreed secure channel. Capture it into your secret manager immediately.

Key shape: `pmk_<env>_<32-base64url-chars>` where `<env>` is `live` (production), `sand` (sandbox), or `test` (local development). The random body is generated from 24 random bytes via `secrets.token_urlsafe(24)`, yielding about 190 bits of entropy.

Set it in your shell for the rest of this guide:

```
export PARTNERMAX_API_KEY="pmk_live_REPLACE_WITH_YOUR_KEY"
```

PartnerMAX v1 allows **one active API key per partner account per environment**. Use the same environment key for your production consumers, CI jobs, and background workers by distributing it through your own secret manager.

If the key is lost, exposed, or due for routine replacement, contact DealerMAX support from the registered technical contact. Support rotates the credential atomically: the old key is deactivated, a new key is delivered once through the secure channel, and only the hash plus 12-character prefix are retained by the platform. There is no multi-key overlap window in v1.

What just happened

DealerMAX generated 24 random url-safe bytes, prepended the environment prefix, computed `SHA-256(plaintext)` for shared key resolution, stored only the hash plus safe metadata, and exposed the plaintext exactly once through the onboarding path you used.

Step 2 — Create an opaque dealer reference

Register the dealer reference in the PartnerMAX dealer registry. Do not send the dealer's commercial anagrafica: keep VAT, email, phone, address, and business-name data in your system. DealerMAX stores only your `external_dealer_id` plus a hidden technical tenant used by stock and NLT calculators.

```python
from partnermax import Partnermax

client = Partnermax()  # reads PARTNERMAX_API_KEY

dealer = client.dealers.create(
    external_dealer_id="acme-001",
    activate=True,
    metadata={"internal_crm_id": "ACME-78432"},
)
print(dealer.dealer_id, dealer.status, dealer.created)
```

```javascript
import Partnermax from "partnermax";

const client = new Partnermax();

const dealer = await client.dealers.create({
  external_dealer_id: "acme-001",
  activate: true,
  metadata: { internal_crm_id: "ACME-78432" },
});
console.log(dealer.dealer_id, dealer.status, dealer.created);
```

Successful response:

```
HTTP/1.1 201 Created
Content-Type: application/json
```

```
{
  "dealer_id": "acme-001",
  "partner_id": "ptn_5858",
  "status": "active",
  "public_surfaces_enabled": false,
  "created_at": "2026-05-17T23:10:15.603295Z",
  "updated_at": "2026-05-17T23:10:15.603295Z",
  "created": true
}
```

Capture the dealer ID:

```
export DEALER_ID="acme-001"
```

What just happened

The API created a partner-dealer registry entry keyed by your `external_dealer_id`, plus the hidden technical tenant needed by the stock and NLT calculators. That tenant is parented to the configured DealerMAX admin/root, not to your partner user, so NLT reads inherit the admin-generated offer catalog correctly. The technical profile stores only pricing and rendering settings; it is not commercial anagrafica and it is not a real DealerMAX dealer claim.

Note the response shape: - `dealer_id` is your opaque external id and is the required value used in every PartnerMAX `/v1/dealers/{dealer_id}/...` stock and NLT path. - `public_surfaces_enabled=false` means DealerMAX public/AI publication is not automatically enabled. Your SDK/API reads still work for the active partner dealer reference.

Keep `activate=true` when the next call is a PartnerMAX `/v1` stock or NLT operation. If you intentionally create the reference with `activate=false`, the partner registry row exists and `GET /v1/dealers/{dealer_id}` reports `status="inactive"`, but the default active listing omits it and stock/NLT operations reject it until you activate that reference.

Step 3 — Configure NLT settings

NLT economics are configured per dealer. You set:

- **Agency mark-up percent** (`0`–`10`): the percentage the dealer adds on top of the network base monthly canon. This is a **mark-up** on the wholesale canon, **not a commission to DealerMAX**. DealerMAX revenue comes only from the partner subscription.
- **Three down-payment tiers** (`low / medium / high`): each carrying `percent_of_list` (`0`–`100`) + `fixed_eur` (`≥0`). The final EUR amount per offer is `listino_imponibile × percent_of_list/100 + fixed_eur`.
- **Image mode**: `branded` (default — per-dealer composite), `scenario_locked` (single AI scenario fixed by the dealer), or `scenario_seasonal` (AI scenario auto-rotated by Italian season).

```
curl -X PATCH "https://api.dealermax.app/v1/dealers/$DEALER_ID/nlt-settings" \
  -H "X-Api-Key: $PARTNERMAX_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "agency_markup_percent": 5,
    "down_payment_tiers": {
      "low":    { "percent_of_list": 0,    "fixed_eur": 0 },
      "medium": { "percent_of_list": 12.5, "fixed_eur": 0 },
      "high":   { "percent_of_list": 25,   "fixed_eur": 0 }
    },
    "image_mode": "scenario_locked",
    "image_scenario_locked": "mediterraneo"
  }'
```

Successful response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```


```
{
  "dealer_id": "acme-001",
  "agency_markup_percent": 5.0,
  "down_payment_tiers": {
    "low":    { "percent_of_list": 0,    "fixed_eur": 0 },
    "medium": { "percent_of_list": 12.5, "fixed_eur": 0 },
    "high":   { "percent_of_list": 25,   "fixed_eur": 0 }
  },
  "image_mode": "scenario_locked",
  "image_scenario_locked": "mediterraneo",
  "currency": "EUR",
  "effective_from": "2026-05-17T23:10:42.180Z"
}
```


What just happened

The platform persisted the mark-up, tiers, and image rendering preferences for that partner dealer. From now on, every `GET /v1/dealers/{id}/nlt/offers` request returns prices with your mark-up amortized over the duration, and the cover image is composed with the selected AI scenario.

For the full reference (response field-by-field, validation rules, worked example), see NLT settings.

Step 4 — Fetch the dealer-aware NLT catalog

Read the listing — offers with the dealer's prices already applied:

```
curl "https://api.dealermax.app/v1/dealers/$DEALER_ID/nlt/offers?limit=3" \
  -H "X-Api-Key: $PARTNERMAX_API_KEY"
```

Successful response (truncated):

```
HTTP/1.1 200 OK
Content-Type: application/json
RateLimit-Limit: 300
RateLimit-Remaining: 297
```

```
{
  "data": [
    {
      "offer_id": "381",
      "slug": "business-volkswagen-t-cross-t-cross-1-0-tsi-edition-plus-115cv-dsg",
      "brand": "Volkswagen",
      "model": "T-Cross",
      "trim": "T-Cross 1.0 tsi Edition Plus 115cv dsg",
      "fuel_type": "Benzina",
      "segment": "SUV piccoli",
      "monthly_canon_from_eur": 244.12,
      "duration_months": 48,
      "km_per_year_at_quote": 10000,
      "dealer_id": "acme-001",
      "vat_treatment": "business",
      "image_url": "https://cdn.azcore.it/storage/v1/render/image/public/modelli-ai/3141/mediterraneo-c536791f.webp?width=1
      "canonical_url": null,
      "has_promo": false
    }
  ],
  "has_more": true,
  "next_cursor": "Mzgx"
}
```


Drill into one offer to retrieve the 18-cell pricing matrix, Motornet technical sheet, partner-network breakdown, and Italian FAQ block:

```
curl "https://api.dealermax.app/v1/dealers/$DEALER_ID/nlt/offers/ofr_381" \
  -H "X-Api-Key: $PARTNERMAX_API_KEY"
```

The detail response (truncated for readability) carries 29+ top-level fields including:

- `quotations[]` — up to 18 cells `{duration_months, km_per_year, monthly_canon_eur}` priced with your mark-up + `high` down-payment tier.
- `down_payment_scenarios_eur` + `down_payment_scenarios_labels` — the three down-payment amounts applied to this offer's list price (e.g. `{"zero": 0, "medium": 3134, "standard": 6269}` for a `25 075 €` list-imponibile).
- `image_url` — **identical** to the listing summary for this offer (Mediterraneo AI scenario, per your `image_mode`).
- `gallery[]` — vehicle photos deduped by view angle and ordered frontale → tre-quarti → laterale.
- `technical_details` — flat dict with up to 65 Motornet-backed fields (cilindrata, kW, dimensioni, bagagliaio, CO_2, pneumatici, …). Keys stay in Italian because they are domain labels from the Italian catalogue.
- `included_services[]` — services bundled in the canone (Assicurazione RCA, Manutenzione, Assistenza Stradale, …).
- `faqs[]` — up to 11 Italian Q&A pairs (dimensioni, motore, canone preset, durate, IVA, anticipi).
- `network_offers[]` — other dealers in your network fulfilling this offer, sorted by canon ascending.

Full payload shape: NLT catalog → detail.

What just happened

The platform looked up your NLT settings, applied the 5% mark-up + `high` tier (25%) to the network base canons, dropped cells whose computed canon falls below the plausibility threshold, composed the cover image against your selected scenario, joined the curated gallery, built the FAQ block, and emitted the response. The same composed image URL is returned in both listing and detail so the partner client renders a single coherent cover per offer.

Step 5 — Verify on the developer portal

Open https://developers.dealermax.app/docs for the public developer index and https://developers.dealermax.app/partner for the PartnerMAX Enterprise overview. The PartnerMAX OpenAPI contract and any interactive console access are shared during approved onboarding rather than published as an unrestricted public surface.

You can also import the Postman collection into Postman or Insomnia for a tabbed workflow.

Troubleshooting

Symptom	HTTP	Likely cause	Fix
`missing_api_key` / `invalid_api_key`	401	The `X-Api-Key` header is missing, the value is wrong, or the key has been revoked or replaced.	Set `X-Api-Key: pmk_<env>_…` (or `Authorization: Bearer …`). If the key is lost or exposed, request a support-managed replacement.
`expired_api_key`	401	The key's `expires_at` has passed.	Contact DealerMAX support for a controlled replacement.
`subscription_inactive` / `grace_period_expired`	402	The DealerMAX subscription is no longer Active, and the 14-day Grace window has expired.	Restore payment in the billing portal. Status flips back within ~30 seconds. See Authentication.
`forbidden_dealer_not_owned`	403	The `dealer_id` in the path is not registered for this partner key.	Verify you are operating on a dealer reference returned by `client.dealers.create(...)` or `GET /v1/dealers`. The error masks both "exists but not authorized" and "does not exist" to prevent cross-partner enumeration.
`not_found`	404	No dealer with this ID, the user has a non-`dealer` role, or the dealer was soft-deleted.	Check the `dealer_id` and that you have not deleted it.
`validation_error`	422	A request body field failed schema validation.	Inspect the `errors[]` array in the RFC 7807 response — each entry carries `{field, code, message}`. Common cases: `agency_markup_percent` outside `[0, 10]`, `image_scenario_locked` set when `image_mode != "scenario_locked"`.

For every error response, the body is RFC 7807 `application/problem+json` and carries a `trace_id` correlated to the W3C traceparent header for support diagnostics. See Error Handling.

SDK patterns

The four patterns below cover every integration shape partnermax needs in production. Copy them once into a thin internal wrapper and the rest of the code stays declarative.

1. Construction and reuse

The `Partnermax` client is HTTP-pool-backed; keep one instance for the lifetime of the process. The constructor reads `PARTNERMAX_API_KEY` from the environment by default — pass `api_key=` explicitly only if you load secrets through a different channel.

```
import os
from partnermax import Partnermax, DefaultHttpxClient

# Module-level singleton — reuse the connection pool across requests.
client = Partnermax(
    api_key=os.environ["PARTNERMAX_API_KEY"],
    timeout=30.0,           # seconds — default
    max_retries=5,          # default; SDK applies decorrelated jitter back-off
)
```

2. Cursor pagination

NLT catalog listing paginates with opaque base64url cursors. After the next Stainless regeneration, SDK list methods are declared as cursor-paginated and can be consumed as iterators:

```
for offer in client.dealers.nlt.offers.list(dealer_id="acme-001", limit=50):
    print(offer.offer_id, offer.brand, offer.model, offer.monthly_canon_from_eur)
```

Raw HTTP clients still use the response envelope manually: read `data`, continue while `has_more=true`, and send `cursor=next_cursor` on the next request. The cursor format is platform-controlled. Don't decode it; just round-trip the string.

3. Idempotent writes

Every mutating endpoint accepts an `Idempotency-Key` header (Stripe-style convention). Pick a stable string per *logical* operation — not per HTTP attempt — and retries become safe:

```
dealer = client.dealers.create(
    external_dealer_id="rossi-001",
    activate=True,
    metadata={"internal_dealer_code": "ROS-001"},
    extra_headers={"Idempotency-Key": "create-dealer-rossi-001-2026-06-27"},
)
```

A retry with the same `key` and same body returns the original response (and the same `dealer_id`) for the retention window. A retry with the same `key` but a *different* body returns `409 idempotency_key_reuse`, which means the operation you're trying to perform is not the one you originally scheduled.

4. Typed error handling

The SDK raises a typed exception per HTTP status family. Catch the narrowest type that matches the recovery action:

```python
from partnermax import (
    Partnermax,
    AuthenticationError,       # 401
    PermissionDeniedError,     # 402 / 403
    NotFoundError,             # 404
    ConflictError,             # 409
    UnprocessableEntityError,  # 422
    RateLimitError,            # 429
    APIConnectionError,        # transport-level
)

try:
    dealer = client.dealers.retrieve("acme-001")
except NotFoundError:
    log.warning("dealer acme-001 has been deleted")
except PermissionDeniedError as exc:
    # 402 (subscription_inactive) or 403 (forbidden_dealer_not_owned).
    # 402 is not retryable — escalate to whoever pays the DealerMAX invoice.
    notify_finance(exc.body)
    raise
except RateLimitError as exc:
    # 429 — honour Retry-After (the SDK already retried `max_retries` times
    # internally with jitter; re-raising means the budget is exhausted).
    sleep_until(exc.retry_after)
    raise
except APIConnectionError:
    # transport-level: DNS, TLS handshake, connection reset. Safe to retry.
    raise
```

Every exception exposes `.status_code`, `.body` (the parsed Problem Details), and the underlying `trace_id` — include the latter when filing a support ticket.

The four patterns above also live in the Error Handling and Rate Limits chapters with deeper context.

Next steps

You have completed the end-to-end happy path. From here:

- Authentication — API key lifecycle, subscription gating, capability inheritance.
- API reference: dealer management — SDK dealer-reference creation plus `/v1` dealer read/update/delete semantics.
- API reference: NLT settings — full schema + worked example for the pricing formula.
- API reference: NLT catalog — full listing + detail shape.
- Error handling — full error-code catalog, `trace_id` propagation, retry semantics.
- Rate limits — current quotas (`300 req/min` for reads, `60 req/min` for writes, per API key) and the `RateLimit-*` response headers.
- Sandbox — sandbox-prefixed keys, billing-provider test mode, isolated dataset.

If you are on the Standard Trial Path, add Stripe before the 15-day trial ends to keep the live key active. If you are on the Enterprise Onboarding Path, coordinate production enablement with your account manager. No code changes are required when switching environments: the host stays the same and routing is decided by the key prefix.

CHAPTER 04

Authentication

The partnermax API uses a single credential type: a **partner API key** for server-to-server traffic. Keys are long-lived, scoped to a partner account, and used for every API call.

Key validity is gated by the partner's DealerMAX subscription status. This document covers how to obtain, send, and replace API keys, plus the subscription state machine that controls their lifecycle.

Architecture boundary: the key identifies the partner account and is accepted on the single public API host, `https://api.dealermax.app`. The generated SDK uses `/api/partner/*` for partner-registry workflows and `/v1/*` for dealer-scoped stock and NLT workflows. Internal service names are infrastructure details, not partner endpoints.

API key model

Partner API keys

A partner API key authenticates an entire partner account. There is exactly one credential type at the partner level — there are no per-dealer, per-user, or per-endpoint sub-keys. Keys are namespaced by environment: a sandbox key cannot authenticate against production, and vice versa.

Property	Sandbox	Production
Prefix	`pmk_sand_`	`pmk_live_`
Length	41 characters for current `live`, `sand`, and `test` keys: 9-character environment prefix (`pmk_<env>_`) + 32-character base64url body, ≈190 bits of entropy	Same
Storage	`sha256(plaintext)` hex digest, no salt; first 12 characters of the full plaintext retained in `key_prefix` for display and log correlation	Same
Lifetime	No expiry; valid until revoked or subscription becomes inactive	Same
subscription state required	`active` or `trialing` (test mode)	`active` or `trialing` (live mode)
Issued via	Standard Trial Path self-service signup, or Enterprise Onboarding Path support-managed secure delivery/replacement	Same
Revoked via	DealerMAX support during atomic replacement or deactivation	Same

The plaintext key value is returned **only once**, at issuance. The platform stores only the SHA-256 hash and a safe 12-character `key_prefix` such as `pmk_live_aBc`; the plaintext itself is never persisted. The 12-character `key_prefix` is not the environment prefix. If you lose the plaintext, request a support-managed replacement — recovery is not possible.

The SHA-256 (no-salt) scheme is deliberate: it lets the same key row be resolved consistently across the DealerMAX platform without each service maintaining its own salt. The entropy floor (about 190 random bits from `secrets.token_urlsafe(24)`) makes brute-forcing the hash computationally infeasible without per-key salting.

How partners obtain their first key

Initial credentials are issued through one of two onboarding paths. In the Standard Trial Path, the self-service signup flow returns one live trial key exactly once after company verification and legal acceptance. In the Enterprise Onboarding Path, DealerMAX support mints the key and delivers it to the partner's named technical contact through the agreed secure channel. PartnerMAX v1

allows **one active API key per partner account per environment**. The key carries `can_issue_keys` for key-lifecycle authorization, but that capability does not create a multi-key self-service quota.

The partnermax surface does not expose a human/browser login endpoint. Any future partner-facing dashboard is a separate web application that manages its own session outside the partnermax API scope.

How to send the key

The API accepts two header conventions. The `X-Api-Key` header is **preferred** because it cannot collide with framework-level authorization middlewares that intercept the `Authorization` header.

Preferred: `X-Api-Key`

```
GET /v1/dealers HTTP/1.1
Host: api.dealermax.app
X-Api-Key: pmk_live_REPLACE_WITH_YOUR_KEY
Accept: application/json
```

Accepted: `Authorization: Bearer`

```
GET /v1/dealers HTTP/1.1
Host: api.dealermax.app
Authorization: Bearer pmk_live_REPLACE_WITH_YOUR_KEY
Accept: application/json
```

If both headers are present, `X-Api-Key` wins and `Authorization` is ignored.

Code samples

```
curl -X GET https://api.dealermax.app/v1/dealers \
  -H "X-Api-Key: $PARTNERMAX_API_KEY"
```

```
import os
from partnermax import Partnermax

client = Partnermax(api_key=os.environ["PARTNERMAX_API_KEY"])
dealers = client.dealers.list()
```

The official TypeScript / JavaScript SDK is published as `partnermax` on npm. Install with `npm install partnermax`; it is generated from the same PartnerMAX OpenAPI contract as the Python SDK and released from CI with signed provenance.

The Python SDK reads `PARTNERMAX_API_KEY` from the environment by default. There is a **single host** for both environments — `https://api.dealermax.app` — and the request is routed to sandbox or production based on the key's `pmk_<env>_` prefix (`pmk_sand_*` → sandbox database and Stripe test mode; `pmk_live_*` → production database and Stripe live mode). See Sandbox for the full isolation model.

Security best practices

The platform enforces what it can — SHA-256 hashing of the plaintext at rest, environment namespacing through the key prefix, automatic subscription-gated revocation. The rest is your responsibility.

Single active key policy

PartnerMAX v1 supports **one active API key per partner account per environment**. Keep that key in a secret manager and distribute it to your production services, CI jobs, and workers from there. Do not create separate PartnerMAX keys per deployment or per engineer.

Rotate production API keys at least every **90 days**, and immediately after any suspected exposure. Replacement is support-managed: the registered technical contact asks DealerMAX support to rotate the key, and the platform deactivates the old key and issues the new one in one atomic operation. The new plaintext is delivered exactly once through the secure channel.

There is no multi-key overlap window in v1. Plan the rollout so every consumer can be updated from your secret manager as soon as the replacement is delivered. If a rotation response is lost before the plaintext reaches your secret manager, the plaintext cannot be recovered; contact DealerMAX support for a new controlled replacement.

Never log keys

API keys grant full partner-scoped access. Do not log them in application logs, exception traces, request dumps, or analytics events. The SDKs scrub the `X-Api-Key` and `Authorization` headers from their own debug logs by default; verify your HTTP client does the same.

Store keys in a secrets manager

Use a dedicated secrets manager — AWS Secrets Manager, Google Secret Manager, HashiCorp Vault, Doppler, Infisical, or equivalent. Do not commit `.env` files to version control. Do not embed keys in container images. Do not pass keys as command-line arguments (they show up in `ps`).

Restrict by IP allowlist (optional)

Production API keys may be bound to a list of source IPs or CIDR ranges via support request. Requests originating from outside the allowlist receive `403 Forbidden` with `code: "ip_not_allowlisted"`. This is recommended for production keys; sandbox keys can be left unrestricted for development convenience.

Sandbox vs production isolation

The two environments are fully isolated at the data layer (separate Supabase database, separate billing account, separate dealer/audit tables) even though they share the same hostname. A `pmk_sand_*` key cannot read or mutate production data, and a `pmk_live_*` key cannot read or mutate sandbox data — the auth resolver enforces the environment band on every request. Use the sandbox for all development, integration testing, and CI. Use production only for live traffic. The prefix (`pmk_sand_` vs `pmk_live_`) makes the environment obvious in any log line.

Subscription-gated lifecycle

Every API key’s validity is tied to your DealerMAX subscription. The lifecycle reduces to three states from the integrator’s perspective:

State	HTTP behavior	How you got here	How to leave it
Active	`200 OK` (full access)	Subscription is paid up, or on a free trial	n/a — keep paying
Grace	`200 OK` for 14 days from the first failed payment, then `402`	Payment failed but the platform is honoring a recovery window	Update the payment method in the billing portal; the grace clock stops the moment payment clears
Inactive	`402 Payment Required` on every authenticated call	Grace expired, or you cancelled outright	Restore payment in the billing portal; access is re-enabled within 60 seconds of the billing provider's confirmation webhook

The transition from grace to inactive happens at the **start of day 15** in Europe/Rome time — a calendar day, not a rolling 14-day timestamp. Integration teams can therefore monitor a fixed date rather than a moving target.

Underneath, the platform tracks the full state machine of the billing provider it uses (Stripe). Mapping is mechanical (`active` / `trialing` → Active; `past_due` / `unpaid` → Grace; everything else terminal → Inactive). The mapping is an implementation detail of the platform and is not exposed in the public API contract; partners should only branch on the three states above and on the HTTP status code.

`402` response shape

When a request is rejected for subscription reasons, the response includes a `WWW-Authenticate` header so HTTP-aware clients can branch on `error="subscription_inactive"`:

```
HTTP/1.1 402 Payment Required
Content-Type: application/problem+json
WWW-Authenticate: PartnerSubscription realm="partnermax", error="subscription_inactive"

{
  "type": "https://developers.dealermax.app/errors/subscription_inactive",
  "title": "Partner subscription inactive",
  "status": 402,
  "detail": "Restore payment in the billing portal to resume API access.",
  "code": "subscription_inactive",
  "instance": "/v1/dealers",
  "trace_id": "00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01"
}
```

For requests rejected after the grace window the `code` flips to `grace_period_expired` and the `WWW-Authenticate` header tracks it: `error="grace_period_expired"`. Same body shape, same recovery action.

See Error Handling → 402 Payment Required for the full error catalog.

Recommended client branching

```python
import partnermax

client = partnermax.Partnermax(api_key=os.environ["PARTNERMAX_API_KEY"])

try:
    dealers = client.dealers.list()
except partnermax.AuthenticationError:
    # 401 — rotate the key.
    rotate_key()
except partnermax.PermissionDeniedError as exc:
    # 402 — surface the billing-portal link from your records to ops/finance,
    # then exit. Don't auto-retry; the recovery is a human action.
    notify_finance("DealerMAX API access suspended; restore billing.")
    raise
except partnermax.RateLimitError as exc:
    # 429 — honour Retry-After, exponential back-off.
    ...
```

The 402 family is not retryable. Treat it as “stop, escalate to whoever pays the DealerMAX invoice.”

Cascade on subscription cancellation

When your DealerMAX subscription enters a terminal state (cancellation or unrecoverable payment failure past the grace window), the platform performs the following actions within 60 seconds of the billing-provider webhook:

1. **API keys remain in the database** but every request returns `402 Payment Required`. Keys are not deleted — your audit log stays intact.
2. **All dealer references registered for the partner transition to `inactive` status.** This is recorded in the dealer audit log with `reason: "partner_subscription_canceled"`.
3. **Public consumer surfaces stop serving the dealer’s data.** Dealer websites return `503 Service Unavailable`. MCP server entries are removed within the next index cycle. Custom GPT and NLWeb stop returning the dealer.
4. **The dealer’s domain configuration is preserved.** DNS settings, JSON-LD configuration, NLT settings, content snapshots — all retained for restoration. Nothing is deleted.
5. **A summary email is sent** to the billing contact on file, listing every dealer that was deactivated and providing the reactivation flow.

The cascade is intentional. A partner subscription that lapses indicates a billing or business-continuity problem; serving stale dealer data through the network’s AI-citation surfaces would create downstream confusion for end customers.

Reactivation flow

Reactivation is a three-step process. There is no API endpoint to “reactivate a partner” — the trigger is in the billing portal.

Step 1 — Restore payment

Log into the billing portal at the link emailed at cancellation. Update the payment method and clear any outstanding invoices. Your subscription flips back to Active typically within 30 seconds of payment confirmation, and the platform re-enables API key access via the billing-provider webhook. A confirmation email is sent to the billing contact.

Step 2 — Re-activate dealers manually

Dealer reactivation is not automatic. This is deliberate: between cancellation and reactivation, your dealer roster may have changed (a dealer may have churned, been sold, or entered a different partnership), and we do not want to silently restore dealers that should have stayed inactive.

For each dealer you want to restore, issue:

```
curl -i -X PATCH https://api.dealermax.app/v1/dealers/$DEALER_ID \
  -H "X-Api-Key: $PARTNERMAX_API_KEY" \
  -H "Content-Type: application/json" \
  -H "Idempotency-Key: reactivate-$DEALER_ID-2026-05-17" \
  -d '{
    "status": "active"
  }'
```

The dealer's NLT settings, content snapshots, and indexing position are restored to their pre-deactivation state. The `indexed_in_surfaces` flags transition back to `true` once the next index cycle completes — typically five minutes for MCP and llms.txt, longer for Custom GPT and NLWeb which require per-dealer bootstrap.

Step 3 — Verify

```
for dealer in client.dealers.list().data:
    print(dealer.dealer_id, dealer.status)
```

Use `client.dealers.list(status="inactive")` to surface any dealer still pending reactivation.

Audit logging

Every credential lifecycle event is recorded in the partner audit log. Partners may request a CSV or JSONL export of their account's audit log for any date range by emailing `support@dealermax.app`; exports are provided within five business days at no charge. Logged events include:

Event	Fields captured
`api_key.issued`	key_id, label, capabilities, environment, source_ip, user_agent, actor (issuing `api_key_id` or `system:onboarding`), timestamp
`api_key.revoked`	key_id, revoked_by (`api_key_id`), reason, source_ip, timestamp
`api_key.used_first_time`	key_id, source_ip, user_agent, timestamp
`api_key.cascade_disabled`	key_id, reason: `partner_subscription_canceled`, billing_event_id, timestamp
`dealer.cascade_disabled`	dealer_id, reason: `partner_subscription_canceled`, billing_event_id, timestamp
`dealer.reactivated`	dealer_id, reactivated_by (`api_key_id`), timestamp

Audit logs are retained for **24 months** in the EU data plane (Supabase Frankfurt region). They are not exposed via the API at this time; export is available on request through your account manager.

Common authentication errors

HTTP	`code`	When	Recovery
401	`missing_api_key`	No `X-Api-Key` or `Authorization` header sent.	Add the header.
401	`invalid_api_key`	Key not found, revoked, or hash mismatch.	Check you are not mixing sandbox/production. If the key is lost or exposed, request a support-managed replacement.
401	`expired_api_key`	The key passed its `expires_at` timestamp.	Request a support-managed replacement.
402	`subscription_inactive`	The DealerMAX subscription has entered a terminal state.	Restore payment in the billing portal.
402	`grace_period_expired`	Payment failed and the 14-day Grace window has expired.	Restore payment in the billing portal.
403	`capability_required`	The calling key lacks the capability gating the endpoint (currently only `can_issue_keys`).	Use the active key issued by DealerMAX support, or request support if the credential needs replacement.
409	`key_rotation_conflict`	A support-supervised key replacement raced with another rotation or the authenticating key is no longer active.	Contact DealerMAX support; do not retry blindly with stale key material.

IP allowlisting and environment-mismatch detection are reserved for future releases — v1 routes by key prefix at the auth resolver, returning `invalid_api_key` for any cross-environment attempt. `429 rate_limited` is active when the Redis-backed per-key budget is exhausted (see Rate Limits → v1 enforcement state).

See Error Handling for the complete error catalog with examples.

Next steps

- Getting Started — End-to-end walkthrough using a sandbox API key.
- API Reference — Endpoint-by-endpoint PartnerMAX reference.
- Error Handling — RFC 7807 problem details, error codes, retry guidance.
- Rate Limits — Per-endpoint quotas and headers.
- Sandbox — Sandbox environment specifics and Stripe test mode.

CHAPTER 05

API Reference — Overview

This section documents the current PartnerMAX enterprise contract used by the official `partnermax` SDKs. The public API host is always `https://api.dealermax.app`; the SDK maps methods to the correct namespace:

- `/api/partner/*` for partner-registry workflows, such as opaque dealer-reference creation.
- `/v1/*` for PartnerMAX operational workflows: dealer-reference reads, NLT settings/catalog reads, used-vehicle stock, images, and accessories.

Partner integrations should not call internal service hostnames directly and should not reimplement platform behavior. Use the SDK method surface as the contract. The machine-readable PartnerMAX OpenAPI file is gated to approved enterprise onboarding; the unrestricted public OpenAPI on the Developers portal is the read-only Dealer Public API contract.

This reference assumes you have completed Getting Started and have a working API key issued against the sandbox environment.

Reference index

Section	Summary
API key lifecycle management	List the active partner key and understand the support-managed replacement flow.
Dealer management	Create opaque dealer references through the SDK, then list, retrieve, update, suspend, or delete those references.
NLT settings	Configure the agency mark-up, down-payment tiers, and VAT treatment for each dealer.
NLT catalog	Read the rental offer catalog with the calling dealer's pricing already applied.
Used-vehicle inventory	Create, bulk-import, update, soft-delete, and attach images to a dealer's used-vehicle stock.

Common conventions

This section is a one-screen refresher. Each topic links to a dedicated page when you need the full contract.

Base URL

Environment	Base URL	Selection mechanism
Production	`https://api.dealermax.app`	Key prefix `pmk_live_*`
Sandbox	`https://api.dealermax.app` (same host)	Key prefix `pmk_sand_*`

There is exactly one host for the partnermax API. Production and sandbox traffic share the same DNS name, the same TLS endpoint, and the same FastAPI service — the partner's API key prefix is what the auth resolver uses to decide which data plane (production or sandbox Supabase project, live or test-mode Stripe account) to route the request against.

Most operational endpoints are versioned under `/v1/`. Dealer-reference creation is exposed as `POST /api/partner/dealers`; generated SDK users call `client.dealers.create(...)`. Breaking changes to the stable PartnerMAX contract ship under `/v2/`; non-breaking additions are made in place. See versioning policy.

Dealer scoping

Every stock and NLT operation names a dealer:

```
/v1/dealers/{dealer_id}/vehicles...
/v1/dealers/{dealer_id}/nlt-settings
/v1/dealers/{dealer_id}/nlt/offers...
```

`dealer_id` is the partner-supplied `external_dealer_id` returned by `client.dealers.create(...)`. There is no global stock listing in PartnerMAX: each vehicle always belongs to exactly one registered dealer reference, and cross-dealer access is hidden behind the documented `404 vehicle_not_found` / `403 forbidden_dealer_not_owned` problem shapes.

Identifier meanings:

Name	Public?	Meaning
`external_dealer_id`	request field	Partner-owned opaque dealer reference sent to `client.dealers.create(...)`, for example `verdicar-001`.
`dealer_id`	response/path field	The same partner-owned value. Use it in every `/v1/dealers/{dealer_id}/...` SDK call.
Technical tenant id	no	Internal DealerMAX row used to attach stock and NLT settings. It is never sent as an API identifier.

Authenticated PartnerMAX reads are served with `Cache-Control: no-store`, `Pragma: no-cache`, CDN no-store headers, and `Vary: Authorization, X-Api-Key` so a GET immediately reflects preceding writes such as image uploads.

Authentication

Every request is authenticated with a partner API key:

Mechanism	Header	Used for
API key (preferred)	`X-Api-Key: pmk_live_...`	Every endpoint, including key management
API key (alternate)	`Authorization: Bearer pmk_live_...`	Same as above. Useful for HTTP clients that auto-inject `Authorization`.

The partnermax surface does not expose a human/browser login endpoint. Initial credentials are issued through the Standard Trial Path self-service signup flow or through Enterprise Onboarding Path support-managed secure delivery. PartnerMAX v1 allows one active API key per partner account per environment; rotation and emergency replacement are handled with DealerMAX support.

API keys carry capabilities. The `can_issue_keys` capability gates key-lifecycle operations, but it does not create a multi-key quota or per-deployment key model in v1.

See auth-keys.md for the full key lifecycle.

Idempotency

Every mutating endpoint (`POST`, `PATCH`, `DELETE`) accepts the `Idempotency-Key` request header. The value should be a UUID v4 unique to the logical operation you intend to perform.

- Keys are retained for **24 hours**.
- A retry of the same key with the **same request body** replays the original response (including the original status code).
- A retry of the same key with a **different request body** returns `409 idempotency_key_reused` and the original response is preserved.

We strongly recommend wrapping all mutating calls in your retry layer with a stable idempotency key per logical operation, especially dealer provisioning, NLT settings updates, vehicle imports, image uploads, and support-supervised key replacement.

Pagination

List endpoints use opaque-cursor pagination:

```
GET /v1/dealers?limit=50&cursor=eyJhZnRlciI6ImRlYWxcl8wMUhCWFlaIn0
```

The response envelope:

```
{
  "data": [ /* ... */ ],
  "has_more": true,
  "next_cursor": "eyJhZnRlciI6ImRlYWxcl8wMUhCWFlaIn0"
}
```

Rules:

- `limit` is bounded per endpoint (typically 1–100). The default is documented per endpoint.
- `next_cursor` is null when `has_more` is false.
- Cursors are opaque; do not decode them. They remain valid for at least 24 hours after issuance.
- The list order is stable on the primary sort key (typically `created_at` ascending) so pagination is deterministic even while new records are being inserted.

Errors

All errors are encoded as RFC 7807 `application/problem+json` documents.

```
HTTP/1.1 422 Unprocessable Entity
Content-Type: application/problem+json

{
  "type": "https://developers.dealermax.app/errors/validation_error",
  "title": "Validation failed",
  "status": 422,
  "code": "validation_error",
  "detail": "primary_domain must be a valid hostname",
  "errors": [
    {
      "field": "primary_domain",
      "code": "format_invalid",
      "message": "must be a valid hostname"
    }
  ],
  "trace_id": "4bf92f3577b34da6a3ce929d0e0e4736"
}
```

Field reference:

Field	Type	Description
`type`	string (URI)	Stable URL identifying the error class.
`title`	string	Short human-readable title.
`status`	integer	Equal to the HTTP status code.
`code`	string	Stable machine-readable code; safe to switch on.
`detail`	string	Human-readable explanation specific to this occurrence.
`errors`	array	Optional per-field validation breakdown. Only present on `422`.
`trace_id`	string | null	32-char hex from the inbound W3C `traceparent` header (when the caller sent one). Include it when contacting support; the response also echoes the request identifier in `X-Request-Id`.

Switch on `code`, not on `title`. Titles may change with translations; codes are part of the API contract.

The `request_id` field is **not** part of the v1 Problem body — request correlation uses the `X-Request-Id` response header (a UUID per request) and the `X-Trace-Id` response header (the 32-char hex from W3C `traceparent` when the caller propagates one). Both are surfaced together so support can correlate.

Rate limits

Scope	Default limit
All requests per API key	**300 requests / minute**
Mutating requests (`POST` , `PATCH` , `DELETE`) per API key	**60 requests / minute**

Limits are enforced as fixed 60-second windows. Every protected `/v1/` response carries:

```
RateLimit-Limit: 300
RateLimit-Remaining: 287
RateLimit-Reset: 42
```

`X-RateLimit-*` alias headers carry the same values. When you exceed the limit, the API returns `429 rate_limited` with a `Retry-After` header (seconds). Higher quotas are available on contractual request; reach out to your DealerMAX account manager.

Timestamps and identifiers

- All timestamps are ISO 8601 in UTC: `2026-05-18T14:32:08Z`.
- Resource identifiers are opaque strings. Dealer IDs are partner-owned external IDs such as `verdicar-001`; NLT offers use stable offer IDs such as `ofr_30591` or the bare numeric offer ID; key records carry UUIDs in the `key_id` field. Treat all identifier values as opaque strings — do not parse them.
- The `trace_id` returned on every response is the W3C `traceparent` value attached to the request. Include it when contacting support.

Content type

Request bodies are `application/json; charset=utf-8` unless the endpoint explicitly declares another media type. Vehicle image upload is `multipart/form-data` with a single `file` field. All responses are `application/json` except errors, which are `application/problem+json`.

CHAPTER 06

API Reference — API Key Lifecycle

This page documents the endpoints used to manage the lifecycle of the API keys that authenticate every programmatic call against partnermax. API keys are the only credential type exposed by the partnermax surface.

Overview

partnermax exposes a single credential type:

- **Partner API key** — sent on every request as `X-Api-Key: pmk_{env}_{base64url}` (preferred) or `Authorization: Bearer pmk_{env}_{base64url}`. Obtained from Standard Trial Path self-service signup, or from Enterprise Onboarding Path support-managed secure delivery/replacement.

API keys follow the format `pmk_{env}_{base64url}`. Current `live`, `sand`, and `test` keys are 41 characters: a 9-character environment prefix (`pmk_live_`, `pmk_sand_`, or `pmk_test_`) plus a 32-character random body. The body carries ≈190 bits of entropy. The full secret is shown **once** at issuance and is never retrievable afterward; only the first 12 characters of the full plaintext are retained as `key_prefix` for display and log correlation.

Initial credentials are issued through one of two onboarding paths. In the Standard Trial Path, the self-service signup flow returns one live trial key exactly once after company verification and legal acceptance. In the Enterprise Onboarding Path, DealerMAX support mints the key and delivers it through the agreed secure channel. PartnerMAX v1 supports **one active API key per partner account per environment**. The lifecycle endpoints below exist for controlled credential management, but partners should not build automated multi-key issuance or overlap-based rotation on top of them.

Capability model

partnermax v1 ships with a single named capability — `can_issue_keys`. The bootstrap key handed to a partner at onboarding holds it; replacement keys keep the same capability set (a key can never widen its own scope). Endpoint authorization is therefore decided at two layers:

1. **Authentication** — does the inbound key resolve to an active partner record under an active DealerMAX subscription?
2. **Capability gate** — does the resolved key carry `can_issue_keys`? Only the key-management endpoints (`POST /v1/keys/issue`, `DELETE /v1/keys/{key_id}`) require it. The dealer, NLT-settings, and NLT-catalog endpoints are scoped only by the partner's ownership of the dealer record they reference, not by a separate capability bit.

There are no `dealers:read`, `dealers:write`, `nlt:read`, or `nlt:write` capability strings — those scoping decisions are made by partner / dealer ownership at the row level, not by a capability whitelist.

`POST /v1/keys/issue`

Rotate the authenticated partner API key during a controlled DealerMAX support flow. The plaintext secret is returned **exactly once**; persist it in your secret manager before discarding the response.

PartnerMAX v1 allows **one active key per partner account per environment**. This endpoint preserves that invariant atomically: the key that authenticated the request is deactivated in the same transaction that creates the replacement. Do not use this endpoint to create extra credentials for separate deployments, CI pipelines, or engineers.

Authentication

An existing API key that carries the `can_issue_keys` capability, used only when DealerMAX support instructs you during a replacement. The initial key delivered at onboarding has this capability by default. The newly minted key inherits the parent's full capability set, and the authenticating key is revoked as soon as the rotation commits.

Request

Headers

Header	Required	Description
`X-Api-Key`	yes	An API key that holds `can_issue_keys`.
`Content-Type`	yes	`application/json`.
`Idempotency-Key`	recommended	Any string ≤256 characters. Prevents accidental duplicate keys on retry.

Body

```
{
  "label": "production-replacement",
  "expires_at": "2027-05-18T00:00:00Z"
}
```

Field	Type	Required	Description
`label`	string	yes	Free-form human-readable label, 1–128 characters. Use a replacement-oriented label such as `production-replacement-2026-06-26`. Surfaced in audit logs and `GET /v1/keys`.
`expires_at`	string (ISO 8601)	no	Absolute expiration timestamp. If omitted, the key never expires (valid until revoked or until the partner subscription enters a terminal state).

The request body does **not** accept a `capabilities` field. The minted key always inherits the caller's capability set. This is a security invariant: a holder of `can_issue_keys` can never accidentally hand out a key with a broader scope than its own.

Response

`201 Created`

```
{
  "key_id": "key_42",
  "key": "pmk_live_REPLACE_WITH_YOUR_KEY",
  "key_prefix": "pmk_live_aBc",
  "label": "production-replacement",
  "created_at": "2026-05-18T14:32:08Z",
  "expires_at": "2027-05-18T00:00:00Z"
}
```

Field	Type	Description
`key_id`	string	Prefixed identifier (`key_<n>`) used to revoke the key later.
`key`	string	**The full plaintext secret. Shown once; never retrievable again.**
`key_prefix`	string	First 12 characters of the plaintext (e.g. `pmk_live_aBc`). Safe to log; surfaced on `GET /v1/keys`.
`label`	string	Echo of the request.
`created_at`	string (ISO 8601)	Issuance timestamp in UTC.
`expires_at`	string (ISO 8601) | null	Absolute expiration, or `null` if non-expiring.

The `capabilities` of the new key are not returned in the response — by design, they are always identical to the caller's capabilities.

Error responses

Status	`code`	Cause
`401`	`missing_api_key` / `invalid_api_key`	The caller did not authenticate, or the key is unknown/revoked.
`402`	`subscription_inactive` / `grace_period_expired`	Partner DealerMAX subscription is not active (and past any grace window).
`403`	`capability_required`	The caller's key does not carry `can_issue_keys`.
`409`	`idempotency_key_reuse`	The same `Idempotency-Key` was used with a different request body.
`409`	`key_rotation_conflict`	The key that authenticated this request is no longer the active key for the partner. Refresh key metadata with `GET /v1/keys` or contact DealerMAX support.
`422`	`validation_error`	`label` empty/too long, or `expires_at` cannot be parsed.

Example — Python SDK (recommended)

```python
import os
from partnermax import Partnermax

client = Partnermax(api_key=os.environ["PARTNERMAX_API_KEY"])
issued = client.keys.issue(
    label="production-replacement",
    expires_at="2027-05-18T00:00:00Z",
)
# Persist issued.key to your secret manager NOW; it cannot be fetched again.
secret_manager.store("PARTNERMAX_API_KEY", issued.key)
```

Example — curl

```bash
curl -X POST https://api.dealermax.app/v1/keys/issue \
  -H "X-Api-Key: $PARTNERMAX_API_KEY" \
  -H "Idempotency-Key: $(uuidgen)" \
  -H "Content-Type: application/json" \
  -d '{
    "label": "production-replacement",
    "expires_at": "2027-05-18T00:00:00Z"
  }'
```

The official TypeScript / JavaScript SDK is published as `partnermax` on npm; JS consumers can use `npm install partnermax` or call the REST endpoint directly.

Notes

- The plaintext is hashed with `sha256(plaintext)` (no salt) and stored alongside the 12-character prefix in `dealer_public_api_keys`. The same row is resolvable by every service in the platform that authenticates partner traffic, which is why the scheme is salt-free. Brute-force resistance comes from the ≈190 bits of entropy in the random body — not from per-key salting.
- Newly minted replacement keys are valid immediately after the controlled replacement completes; there is no propagation delay.
- The previous key is invalid immediately after the rotation commits. If the response is lost before the plaintext is stored, the plaintext cannot be recovered; contact DealerMAX support for a new controlled rotation.

`DELETE /v1/keys/{key_id}`

Revoke an API key. The key stops working within seconds and cannot be reactivated.

Authentication

An active API key that carries `can_issue_keys`. A key cannot revoke itself through this endpoint; use `POST /v1/keys/issue` during the support-managed replacement flow to atomically replace the current key.

Request

Path parameters

Name	Type	Required	Description
`key_id`	string	yes	The identifier returned from `POST /v1/keys/issue`.

Headers

Header	Required	Description
`X-Api-Key`	yes	An active API key holding `can_issue_keys`.
`Idempotency-Key`	recommended	A retry with the same key returns `204` even if the key was already revoked.

Response

`204 No Content`

Empty body.

Error responses

Status	`code`	Cause
`403`	`capability_required`	Caller does not hold `can_issue_keys`.
`403`	`cannot_revoke_self`	The `key_id` is the same key that authenticated this request. Request a support-managed replacement instead.
`404`	`api_key_not_found`	No key with this ID exists for the calling partner. Same response is returned when the key belongs to a different partner (deliberate, to prevent cross-partner enumeration).

Example — Python SDK (recommended)

```
client.keys.revoke(key_id="key_42")
```

Example — curl

```
curl -X DELETE https://api.dealermax.app/v1/keys/key_42 \
  -H "X-Api-Key: $PARTNERMAX_API_KEY"
```

Notes

- Revocation is synchronous at the auth resolver: the very next request authenticated with the revoked key returns `401 invalid_api_key`. There is no edge-cache propagation delay because the resolver reads the live `is_active` flag on each request.
- Audit-log entries for the revoked key are retained indefinitely with the prefix, label, and revocation timestamp.

`GET /v1/keys`

List the API keys minted under the calling partner, with metadata only. The plaintext is never returned by this endpoint — it is only available at issuance time.

Authentication

An active API key for the calling partner.

Request

Headers

Header	Required	Description
`X-Api-Key`	yes	An active API key for the calling partner.

Response

`200 OK`

```
{
  "data": [
    {
      "key_id": "key_42",
      "key_prefix": "pmk_live_aBc",
      "label": "production-replacement",
      "created_at": "2026-05-18T14:32:08Z",
      "last_used_at": "2026-05-18T18:04:51Z",
      "expires_at": "2027-05-18T00:00:00Z",
      "is_active": true
    },
    {
      "key_id": "01970eaa-2b15-7c4f-a3d0-1e2f4b5c6d7e",
      "key_prefix": "pmk_live_2Nb",
      "label": "previous-production-key",
      "created_at": "2026-04-22T09:15:42Z",
      "last_used_at": "2026-05-18T05:11:09Z",
      "expires_at": null,
      "is_active": false
    }
  ]
}
```

Field	Type	Description
`key_id`	string	Identifier of the key.
`key_prefix`	string	First 12 characters of the plaintext. Use this to correlate logs with a specific key without exposing the secret.
`label`	string	Free-form label set at issuance.
`created_at`	string (ISO 8601)	Issuance timestamp.
`last_used_at`	string (ISO 8601) | null	Timestamp of the most recent successful authentication. `null` if the key has never been used. Updated at most once per minute per key.
`expires_at`	string (ISO 8601) | null	Absolute expiration, or `null` if non-expiring.
`is_active`	boolean	`false` if the key has been revoked or has passed `expires_at`.

`GET /v1/keys` returns every key the caller's partner has ever minted, including revoked and expired ones. Filter client-side on `is_active` if you only care about live credentials. Pagination is not currently exposed on this endpoint because v1 allows only one active key per environment, and the historical list is expected to stay small.

Error responses

Status	`code`	Cause
`401`	`invalid_api_key`	Calling key is unknown or revoked.
`402`	`subscription_inactive`	Partner subscription has lapsed past the grace window.

Example — Python SDK (recommended)

```python
for key in client.keys.list().data:
    print(key.key_prefix, key.label, key.last_used_at, key.is_active)
```

Example — curl

```bash
curl https://api.dealermax.app/v1/keys \
  -H "X-Api-Key: $PARTNERMAX_API_KEY"
```

Notes

- `last_used_at` is updated at most once per minute per key, so very recent calls may not be reflected immediately.
- Use this endpoint together with the `key_prefix` to correlate keys with the prefix logged in your application's outgoing request headers. The prefix is the *only* part of the key that is safe to surface in observability tooling.

CHAPTER 07

API Reference — Dealer Management

Dealer-reference endpoints for partner-owned opaque dealer IDs. SDK integrations create dealer references through `client.dealers.create(...)`, which calls `POST /api/partner/dealers` on `https://api.dealermax.app`, then use the returned `dealer_id` in every `/v1/dealers/{dealer_id}/...` stock and NLT path.

The partner keeps dealer commercial data in its own system. DealerMAX stores only the partner-supplied external dealer id plus the hidden technical tenant needed to attach stock and NLT settings.

Overview

A dealer reference is the unit used to scope partner stock and NLT operations. It is not the dealer's full commercial anagrafica. Public/AI publication is a separate activation/claim flow and is represented by `public_surfaces_enabled`. `public_surfaces_enabled=false` does not block SDK reads or writes for an active dealer reference.

The endpoints in this section:

- `POST /api/partner/dealers` - create a dealer reference through the SDK.
- `POST /api/partner/dealers/{external_dealer_id}/activate` - activate a registry reference.
- `POST /api/partner/dealers/{external_dealer_id}/suspend` - temporarily suspend a registry reference.
- `DELETE /api/partner/dealers/{external_dealer_id}` - terminally revoke a registry reference.
- `GET /v1/dealers` — list.
- `GET /v1/dealers/{dealer_id}` — fetch one.
- `PATCH /v1/dealers/{dealer_id}` — update or (de)activate.
- `DELETE /v1/dealers/{dealer_id}` — soft-delete.

`GET /v1/dealers` and `GET /v1/dealers/{dealer_id}` read the partner dealer registry and return the partner-supplied external id.

`POST /api/partner/dealers`

Create or retrieve an opaque dealer reference for the calling partner. This is the endpoint behind `client.dealers.create(...)` in both official SDKs.

Dealer creation is idempotent by `external_dealer_id`: if the same active reference already exists for the partner, the API returns the existing reference with `created=false`.

Authentication

`X-Api-Key` (recommended) or `Authorization: Bearer`.

Request

Headers

Header	Required	Description
`X-Api-Key`	conditional	Required if not using `Authorization: Bearer`.
`Authorization`	conditional	`Bearer <key>` alternative.
`Content-Type`	yes	Must be `application/json`.
`Idempotency-Key`	strongly recommended	Stable key per logical dealer provisioning operation.

Body

```python
from pydantic import BaseModel, Field

class PartnerDealerCreateRequest(BaseModel):
    external_dealer_id: str = Field(min_length=1, max_length=128)
    activate: bool = True
    metadata: dict[str, str | int | float | bool | None] = {}
```

```json
{
  "external_dealer_id": "verdicar-001",
  "activate": true,
  "metadata": {
    "internal_dealer_code": "VERDI-001",
    "tier": "premium"
  }
}
```

Field	Type	Required	Constraints
`external_dealer_id`	string	yes	Partner-owned opaque dealer id, max 128 chars. This becomes the public `dealer_id` used in every SDK path.
`activate`	boolean	no	Default `true`. When `false`, the registry row exists but `/v1` stock and NLT operations reject it until activation.
`metadata`	object	no	Free-form scalar partner-side correlation metadata.

`metadata` field

Open-ended key/value bag for **partner-side** annotation. Keys and scalar values are stored verbatim, never displayed on consumer-facing surfaces, and returned untouched in subsequent reads. Use it to correlate the PartnerMAX `dealer_id` with your internal CRM ID, sales rep, tier label, onboarding source, or DMS identifier.

Response

`201 Created`

```
HTTP/1.1 201 Created
Content-Type: application/json
```

```
{
  "dealer_id": "verdicar-001",
  "partner_id": "ptn_5858",
  "status": "active",
  "public_surfaces_enabled": false,
  "created_at": "2026-05-17T23:10:15.603295Z",
  "updated_at": "2026-05-17T23:10:15.603295Z",
  "created": true
}
```

Field	Type	Description
`dealer_id`	string	The partner-owned `external_dealer_id`. Use this in all downstream calls.
`partner_id`	string	Opaque partner account id.
`status`	string enum	Public API status: `active`, `inactive`, or `deleted`. Internal registry states such as `suspended` and `revoked` are mapped before the SDK response is returned.
`public_surfaces_enabled`	boolean	Whether the dealer reference is published to public/AI surfaces. Does not gate SDK reads/writes.
`created_at` / `updated_at`	string	ISO 8601 UTC.
`created`	boolean	`true` only when this request inserted the registry row.

Error responses

Every non-2xx response follows RFC 7807 Problem Details.

Status	`code`	Cause
`401`	`missing_api_key` / `invalid_api_key` / `expired_api_key`	Auth checks.
`402`	`subscription_inactive` / `grace_period_expired`	Partner subscription gating.
`409`	`dealer_reference_conflict`	The external id exists for this partner in a revoked/conflicting state.
`422`	`validation_error`	Field-level validation failed. Body includes `errors[]` with per-field `{field, code, message}`.

```
HTTP/1.1 409 Conflict
Content-Type: application/problem+json
```

```
{
  "type": "https://developers.dealermax.app/errors/dealer_reference_conflict",
  "title": "Dealer reference conflict",
  "status": 409,
  "code": "dealer_reference_conflict",
  "detail": "Dealer reference 'verdicar-001' exists in a conflicting state.",
  "instance": "/api/partner/dealers",
  "trace_id": "4bf92f3577b34da6a3ce929d0e0e4736"
}
```

SDK behavior

Generated SDKs expose dealer provisioning as `client.dealers.create(...)`. Use the returned `dealer_id` with PartnerMAX `/v1` vehicle and NLT endpoints.

Example - curl

```
curl -X POST https://api.dealermax.app/api/partner/dealers \
  -H "X-Api-Key: $PARTNERMAX_API_KEY" \
  -H "Idempotency-Key: $(uuidgen)" \
  -H "Content-Type: application/json" \
  -d '{
    "external_dealer_id": "verdicar-001",
    "activate": true,
    "metadata": {
      "internal_dealer_code": "VERDI-001"
    }
  }'
```

Official SDKs are published for Python (`pip install partnermax`, repo `DealerMax-app/partnermax-python`) and TypeScript / JavaScript (`npm install partnermax`, repo `DealerMax-app/partnermax-node`).

Notes

- Public/AI surface publication is intentionally separate from dealer-reference creation. The SDK stock/NLT flow works for active dealer references even when `public_surfaces_enabled=false`.
- The `metadata` map is reserved for partner-side correlation and is never surfaced consumer-side.

`POST /api/partner/dealers/{external_dealer_id}/activate`

Reactivate a non-terminal opaque dealer reference. This route is useful when a partner manages the registry lifecycle directly. The operational SDK equivalent is `PATCH /v1/dealers/{dealer_id}` with `{ "status": "active" }`.

Authentication

`X-Api-Key` (recommended) or `Authorization: Bearer`.

Request

Path parameters

Name	Type	Required	Description
`external_dealer_id`	string	yes	Partner-owned opaque dealer id returned as `dealer_id` by `POST /api/partner/dealers`.

Body

No request body.

Idempotency

This endpoint is row-idempotent by final state. Repeating the call for an already-active reference returns `200 OK` with `status: "active"`. It does not require an `Idempotency-Key` header.

Response

`200 OK`

```
{
  "dealer_id": "verdicar-001",
  "partner_id": "ptn_5858",
  "status": "active",
  "public_surfaces_enabled": false,
  "created_at": "2026-05-17T23:10:15.603295Z",
  "updated_at": "2026-05-18T08:22:10.100000Z",
  "created": false
}
```

Error responses

Status	`code`	Cause
`401`	`missing_api_key` / `invalid_api_key` / `expired_api_key`	Auth checks.
`402`	`subscription_inactive` / `grace_period_expired`	Partner subscription gating.
`404`	`dealer_not_found`	No registry reference with this id exists for the calling partner.
`409`	`dealer_deleted`	The reference is terminally deleted/revoked and cannot be activated without DealerMAX support.
`422`	`validation_error`	The path value is malformed.

Example - curl

```
curl -X POST https://api.dealermax.app/api/partner/dealers/verdicar-001/activate \
  -H "X-Api-Key: $PARTNERMAX_API_KEY"
```

`POST /api/partner/dealers/{external_dealer_id}/suspend`

Temporarily suspend an opaque dealer reference. Suspended references map to the public SDK/API status `inactive`, stop accepting stock/NLT operations, and are removed from public/AI surfaces. The operational SDK equivalent is `PATCH /v1/dealers/{dealer_id}` with `{ "status": "inactive" }`.

Authentication

`X-Api-Key` (recommended) or `Authorization: Bearer`.

Request

Path parameters

Name	Type	Required	Description
`external_dealer_id`	string	yes	Partner-owned opaque dealer id returned as `dealer_id` by `POST /api/partner/dealers`.

Body

No request body.

Idempotency

This endpoint is row-idempotent by final state. Repeating the call for an already-suspended reference returns `200 OK` with `status: "inactive"`. It does not require an `Idempotency-Key` header.

Response

`200 OK`

```
{
  "dealer_id": "verdicar-001",
  "partner_id": "ptn_5858",
  "status": "inactive",
  "public_surfaces_enabled": false,
  "created_at": "2026-05-17T23:10:15.603295Z",
  "updated_at": "2026-05-18T08:22:10.100000Z",
  "created": false
}
```

Error responses

Status	`code`	Cause
`401`	`missing_api_key` / `invalid_api_key` / `expired_api_key`	Auth checks.
`402`	`subscription_inactive` / `grace_period_expired`	Partner subscription gating.
`404`	`dealer_not_found`	No registry reference with this id exists for the calling partner.
`409`	`dealer_deleted`	The reference is terminally deleted/revoked and cannot be suspended.
`422`	`validation_error`	The path value is malformed.

Example - curl

```
curl -X POST https://api.dealermax.app/api/partner/dealers/verdicar-001/suspend \
  -H "X-Api-Key: $PARTNERMAX_API_KEY"
```

`DELETE /api/partner/dealers/{external_dealer_id}`

Terminally revoke an opaque dealer reference. From the partner's perspective this is equivalent to `DELETE /v1/dealers/{dealer_id}`: the public status becomes `deleted`, stock/NLT operations stop, and reactivation requires DealerMAX support.

Authentication

`X-Api-Key` (recommended) or `Authorization: Bearer`.

Request

Path parameters

Name	Type	Required	Description
`external_dealer_id`	string	yes	Partner-owned opaque dealer id returned as `dealer_id` by `POST /api/partner/dealers`.

Body

No request body.

Idempotency

This endpoint is row-idempotent by final state. Repeating the call for an already-revoked reference returns `204 No Content`.

Response

`204 No Content`

Empty body. Use `GET /v1/dealers?status=deleted` to reconcile terminally deleted references.

Error responses

Status	`code`	Cause
`401`	`missing_api_key` / `invalid_api_key` / `expired_api_key`	Auth checks.
`402`	`subscription_inactive` / `grace_period_expired`	Partner subscription gating.
`404`	`dealer_not_found`	No registry reference with this id exists for the calling partner.
`422`	`validation_error`	The path value is malformed.

Example - curl

```
curl -X DELETE https://api.dealermax.app/api/partner/dealers/verdicar-001 \
  -H "X-Api-Key: $PARTNERMAX_API_KEY"
```

`GET /v1/dealers`

List dealer references registered for the calling partner.

Authentication

Same API-key authentication model.

Request

Query parameters

Name	Type	Required	Description
`status`	string enum	no	`active` (default), `inactive`, `deleted`, or `all`. `deleted` returns terminally soft-deleted dealer references only; `all` includes every lifecycle state.
`cursor`	string	no	Opaque base64url cursor returned in the previous response's `next_cursor`.
`limit`	integer	no	`1` – `100`. Default `50`.

Headers

Header	Required	Description
`X-Api-Key`	conditional	Required if not using `Authorization: Bearer`.

Response

`200 OK`

```
HTTP/1.1 200 OK
Content-Type: application/json
```


```
{
  "data": [
    {
      "dealer_id": "verdicar-001",
      "business_name": "Partner dealer verdicar-001",
      "primary_domain": "",
      "province_code": "IT",
      "status": "active",
      "nlt_enabled": true,
      "created_at": "2026-05-17T14:32:08.412Z",
      "last_active_at": "2026-05-17T19:11:42.180Z"
    },
    {
      "dealer_id": "autocity-002",
      "business_name": "Partner dealer autocity-002",
      "primary_domain": "",
      "province_code": "IT",
      "status": "active",
      "nlt_enabled": false,
      "created_at": "2026-04-22T09:15:42.180Z",
      "last_active_at": null
    }
  ],
  "has_more": true,
  "next_cursor": "NTg5Mg"
}
```

Field	Type	Description
`dealer_id`	string	Partner-owned external dealer id.
`business_name`	string	Technical display label for the hidden DealerMAX tenant. The partner remains owner of the real dealer anagrafica.
`primary_domain`	string	Hostname (no scheme), empty until a separate publication/claim flow exists.
`province_code`	string	Italian province code.
`status`	string enum	`active`, `inactive`, or `deleted`.
`nlt_enabled`	boolean	`true` if NLT settings have been configured at least once.
`created_at`	string	Original provisioning timestamp.
`last_active_at`	string | null	Most recent visit on any DealerMAX surface. `null` until the first hit.
`has_more`	boolean	`true` when a `next_cursor` was issued.
`next_cursor`	string | null	Opaque cursor for the next page.

The summary record does **not** include `metadata`, `contact_*`, `vat_number`, `indexed_in_surfaces`, or embedded `nlt_settings`. Use `GET /v1/dealers/{dealer_id}` to retrieve the full record.

Error responses

Status	`code`	Cause
`401`	`missing_api_key` / `invalid_api_key` / `expired_api_key`	Auth checks.
`402`	`subscription_inactive` / `grace_period_expired`	Partner subscription gating.
`422`	`validation_error`	The `cursor` is malformed or one of the query parameters is out of range.

Example — Python SDK (recommended)

```python
listing = client.dealers.list(status="active", limit=100)
for dealer in listing.data:
    print(dealer.dealer_id, dealer.business_name, dealer.last_active_at)
```

Example — curl

```bash
curl "https://api.dealermax.app/v1/dealers?status=active&limit=100" \
  -H "X-Api-Key: $PARTNERMAX_API_KEY"
```

Notes

- Results are sorted by newest `partner_dealers` registry row first; pagination is stable across price refreshes and dealer updates.
- Deleted dealers are excluded by the default `status=active` filter. Use `status=deleted` to reconcile only terminally deleted references, or `status=all` for every lifecycle state.

`GET /v1/dealers/{dealer_id}`

Fetch the full record for one dealer.

Authentication

Same API-key authentication model. The `dealer_id` must belong to the calling partner.

Request

Path parameters

Name	Type	Required	Description
`dealer_id`	string	yes	Partner-owned external dealer id, e.g. `verdicar-001`.

Response

`200 OK`

Returns a full `DealerDetail` payload, including the embedded `nlt_settings` object **only when** `nlt_enabled: true`.


```
{
  "dealer_id": "verdicar-001",
  "business_name": "Partner dealer verdicar-001",
  "primary_domain": "",
  "province_code": "IT",
  "status": "active",
  "nlt_enabled": true,
  "created_at": "2026-05-17T14:32:08.412Z",
  "last_active_at": "2026-05-17T19:11:42.180Z",
  "partner_id": "ptn_5858",
  "vat_number": "",
  "contact_email": "partner-5858-verdicar-001@example.invalid",
  "contact_phone": "-",
  "postal_code": "-",
  "metadata": {},
  "indexed_in_surfaces": {
    "mcp": true,
    "custom_gpt": true,
    "nlweb": true,
    "llms_txt": true
  },
  "nlt_settings": {
    "dealer_id": "verdicar-001",
    "agency_markup_percent": 5.0,
    "down_payment_tiers": {
      "low":    { "percent_of_list": 0,    "fixed_eur": 0 },
      "medium": { "percent_of_list": 12.5, "fixed_eur": 0 },
      "high":   { "percent_of_list": 25,   "fixed_eur": 0 }
    },
    "image_mode": "scenario_locked",
    "image_scenario_locked": "mediterraneo",
    "currency": "EUR",
    "effective_from": "2026-05-17T14:48:22.180Z"
  }
}
```


The embedded `nlt_settings` carries the same shape returned by `GET /v1/dealers/{dealer_id}/nlt-settings` — refer to that page for the per-field reference.

Error responses

Status	`code`	Cause
`401`	`missing_api_key` / `invalid_api_key` / `expired_api_key`	Auth checks.
`402`	`subscription_inactive` / `grace_period_expired`	Partner subscription gating.
`403`	`forbidden_dealer_not_owned`	`dealer_id` is not registered for this partner key.
`404`	`not_found`	No dealer with this ID, or the dealer was soft-deleted.

Example — Python SDK (recommended)

```
dealer = client.dealers.retrieve("verdicar-001")
if dealer.nlt_enabled:
    print(dealer.nlt_settings.agency_markup_percent, dealer.nlt_settings.image_mode)
```

Example — curl

```
curl https://api.dealermax.app/v1/dealers/verdicar-001 \
  -H "X-Api-Key: $PARTNERMAX_API_KEY"
```

`PATCH /v1/dealers/{dealer_id}`

Update a subset of the dealer's fields, or (de)activate the dealer.

What can be changed

Only the following fields are mutable through this endpoint:

Field	Effect
`business_name`	Replaces the legal commercial name. Surfaces on consumer-facing pages within ~5 minutes.
`contact_email`	Updates the operational email.
`contact_phone`	Updates the contact phone (free-format, max 32 chars).
`province_code`	Italian province code, 2 uppercase letters.
`postal_code`	Italian CAP, 5 digits.
`metadata`	Replaces the metadata map **wholesale** — send the full desired map (existing keys not present in the request are removed).
`status`	`"active"` or `"inactive"`. Use `"inactive"` to deactivate the dealer (removed from cross-network surfaces within ~5 minutes); use `"active"` to reactivate. Cannot be used to undo a `DELETE` — soft-deleted dealers stay deleted.

`vat_number`, `primary_domain`, and `activate` are **not** mutable through this endpoint. To change a VAT number, soft-delete the dealer and re-provision (subject to VAT reservation rules — contact DealerMAX support). To rotate `primary_domain`, contact support; the operation requires coordinated DNS + TLS work outside this API.

Authentication

Same API-key authentication model. The `dealer_id` must belong to the calling partner.

Request

Path parameters

Name	Type	Required	Description
`dealer_id`	string	yes	Prefixed dealer ID.

Headers

Header	Required	Description
`X-Api-Key`	conditional	Required if not using `Authorization: Bearer`.
`Content-Type`	yes	Must be `application/json`.

Body

```
class UpdateDealerRequest(BaseModel):
    business_name:  str | None
    contact_email:  EmailStr | None
    contact_phone:  str | None
    province_code:  str | None
    postal_code:    str | None
    status:         Literal["active", "inactive"] | None
    metadata:       dict[str, str] | None
```

All fields are optional. Fields not present in the body are left unchanged. To deactivate:

```
{ "status": "inactive" }
```

To rotate metadata + reactivate:

```
{
  "status": "active",
  "metadata": {
    "internal_dealer_code": "ROS-001",
    "tier": "premium",
    "region": "lombardia"
  }
}
```

Response

`200 OK`

Same shape as `GET /v1/dealers/{dealer_id}` — the full updated `DealerDetail` is returned.

Error responses

Status	`code`	Cause
`401`	`missing_api_key` / `invalid_api_key` / `expired_api_key`	Auth checks.
`402`	`subscription_inactive` / `grace_period_expired`	Partner subscription gating.
`403`	`forbidden_dealer_not_owned`	`dealer_id` is not registered for this partner key.
`404`	`not_found`	No dealer with this ID, or the dealer was soft-deleted.
`422`	`validation_error`	Field-level validation failed (e.g. invalid province code).

Example — Python SDK (recommended)

```
client.dealers.update(
    "verdicar-001",
    contact_phone="+390287654321",
    metadata={"internal_dealer_code": "VERDI-001", "tier": "premium", "region": "lombardia"},
)
```

Example — curl

```
curl -X PATCH https://api.dealermax.app/v1/dealers/verdicar-001 \
  -H "X-Api-Key: $PARTNERMAX_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"status": "inactive"}'
```

Notes

- Deactivation is **not** deletion: the dealer record, NLT settings, and audit history are preserved. The dealer can be reactivated by sending `{"status": "active"}`.
- `metadata` replacement is **wholesale** — to remove a single key, fetch the current map, drop the key client-side, and `PATCH` the new map. There is no partial-update / `$unset` semantic.

`DELETE /v1/dealers/{dealer_id}`

Soft-delete a dealer. The platform marks the technical dealer as deleted and disables public/AI visibility. The resource is retained for audit purposes, but is removed from all cross-network AI-citation surfaces (within ~5 minutes once the indexing workers pick up the change) and the `status` field of every subsequent read transitions to `"deleted"`. The dealer cannot be reactivated through `PATCH` — re-creation requires DealerMAX support.

Authentication

Same API-key authentication model. The `dealer_id` must belong to the calling partner.

Request

Path parameters

Name	Type	Required	Description
`dealer_id`	string	yes	Partner-owned external dealer id, e.g. `verdicar-001`.

Response

`204 No Content`

Empty body. The dealer's `status` is flipped to `"deleted"` in the same transaction. Subsequent `GET /v1/dealers/{dealer_id}` calls return `404 dealer_not_found` because soft-deleted rows are excluded from single-resource reads by default; a `?status=deleted` filter on `GET /v1/dealers` can still surface them for reconciliation.

Error responses

Status	`code`	Cause
`401`	`missing_api_key` / `invalid_api_key` / `expired_api_key`	Auth checks.
`402`	`subscription_inactive` / `grace_period_expired`	Partner subscription gating.
`403`	`forbidden_dealer_not_owned`	`dealer_id` is not registered for this partner key.
`404`	`dealer_not_found`	No dealer with this ID under the calling partner.
`409`	`dealer_already_deleted`	The dealer is already soft-deleted; the call is a no-op.

Example — Python SDK (recommended)

```
client.dealers.delete("verdicar-001")
```

Example — curl

```
curl -X DELETE https://api.dealermax.app/v1/dealers/verdicar-001 \
  -H "X-Api-Key: $PARTNERMAX_API_KEY"
```

Notes

- This is the right operation when the partner has lost the commercial relationship with a dealer permanently. For a temporary pause, use `PATCH /v1/dealers/{dealer_id}` with `{"status": "inactive"}`.
- A soft-deleted VAT number remains reserved against the deleted record. Re-provisioning a dealer with the same VAT requires DealerMAX support intervention to release the VAT for re-use.

CHAPTER 08

API Reference — NLT Settings

> *`dealer_id` is the partner-supplied external dealer id created through `client.dealers.create(...)` / `POST /api/partner/dealers`.*

Per-dealer configuration of NLT (Italian *Noleggio Lungo Termine*, long-term car rental) economics. Setting these values is what makes a dealer visible on the NLT catalog endpoints; until then, the dealer is treated as not enrolled.

The pricing model in one paragraph

Every NLT offer in the DealerMAX network carries a **base canon** (monthly rental fee) negotiated between DealerMAX and the rental network for a baseline configuration: one of three durations (36, 48, or 60 months), one of six yearly km tiers (10 000 / 15 000 / 20 000 / 25 000 / 30 000 / 40 000), and a zero-EUR down payment. The displayed canon for a given dealer is computed from this base as:

```
listino_imponibile  = prezzo_listino / 1.22
provvigione         = listino_imponibile × (agency_markup_percent / 100)
anticipo_tier_eur   = listino_imponibile × (tier.percent_of_list / 100) + tier.fixed_eur
canon               = base_canon + provvigione / duration_months
                            − anticipo_tier_eur / duration_months
if offer.private_only:
    canon = canon × 1.22
```

VAT is a **per-offer** property (`NltOfferSummary.vat_treatment` / `NltOfferDetail.private_only`), derived from `nlt_offerte.solo_privati`, **never** a dealer setting. `vat_treatment="private"` (B2C) means the canon is displayed inclusive of 22% Italian VAT; `"business"` (B2B) displays it net.

Every offer is exposed as an **18-quotation matrix** (3 durations × 6 km tiers). For each cell the headline canon uses the dealer's `high` down-payment tier (the most aggressive trade-in, lowest visible price); the three `down_payment_tiers` are also returned separately so consumer-facing UIs can let the customer toggle between them.

`PATCH /v1/dealers/{dealer_id}/nlt-settings`

Configure or update the NLT economics — and image rendering preferences — for one of the partner's dealers. The endpoint is `PATCH` rather than `PUT` because partial-update semantics rarely apply: in practice partners send the full settings object every time. The first successful call enrolls the dealer into the NLT catalog. Subsequent calls re-price the catalog within ~5 minutes.

Authentication

`X-Api-Key` (recommended) or `Authorization: Bearer <key>`. The `dealer_id` must belong to the calling partner.

Request

Path parameters

Name	Type	Required	Description
`dealer_id`	string	yes	Partner-owned opaque dealer id created through `client.dealers.create(...)`, e.g. `verdicar-001`.

Headers

Header	Required	Description
`X-Api-Key`	conditional	Required if not using `Authorization: Bearer`.
`Authorization`	conditional	`Bearer <key>` alternative to `X-Api-Key`. When both are present, `X-Api-Key` wins.
`Content-Type`	yes	Must be `application/json`.

Body

```python
from typing import Literal
from pydantic import BaseModel, Field

class NltDownPaymentTier(BaseModel):
    percent_of_list: float = Field(ge=0, le=100)   # % of IVA-excluded list price
    fixed_eur:       int   = Field(ge=0)           # flat EUR added on top

class DownPaymentTiers(BaseModel):
    low:    NltDownPaymentTier
    medium: NltDownPaymentTier
    high:   NltDownPaymentTier

class UpdateNltSettingsRequest(BaseModel):
    agency_markup_percent: float = Field(ge=0, le=10)
    down_payment_tiers:    DownPaymentTiers
    image_mode:            Literal["branded", "scenario_locked", "scenario_seasonal"] = "branded"
    image_scenario_locked: Literal["mediterraneo", "cortina", "milano", "showroom"] | None = None
    currency:              Literal["EUR"] = "EUR"
```

```json
{
  "agency_markup_percent": 5,
  "down_payment_tiers": {
    "low":    { "percent_of_list": 0,    "fixed_eur": 0 },
    "medium": { "percent_of_list": 12.5, "fixed_eur": 0 },
    "high":   { "percent_of_list": 25,   "fixed_eur": 0 }
  },
  "image_mode": "branded",
  "currency": "EUR"
}
```

Field	Type	Required	Constraints
`agency_markup_percent`	number	yes	Inclusive `[0, 10]`. Stored as integer rounded from the input. Used to compute the dealer's *provvigione* per offer, amortized over the contract duration.
`down_payment_tiers.{low,medium,high}`	object	yes	Three tiers — `low / medium / high` — each carrying `percent_of_list` (0–100) and `fixed_eur` (≥0).
`down_payment_tiers.*.percent_of_list`	number	yes	Percentage of the IVA-excluded list price. Typical defaults: `0` / `12.5` / `25`.
`down_payment_tiers.*.fixed_eur`	integer	yes	Flat EUR added on top of the percentage (e.g. `0% + 500€ fissi`). Whole euros only.
`image_mode`	string enum	no	`branded` (default), `scenario_locked`, or `scenario_seasonal`. Controls how NLT cover images are rendered for this dealer's offers.
`image_scenario_locked`	string enum	conditional	Required **only** when `image_mode = "scenario_locked"`; **must be null** otherwise. One of `mediterraneo`, `cortina`, `milano`, `showroom`.
`currency`	string enum	no	Only `"EUR"` accepted in v1.

Down-payment tiers

The three tiers do **not** need to be strictly ascending: the final EUR amount each tier produces is offer-dependent (`listino_imponibile × pct + eur`), so a tier that looks larger by percentage can produce a smaller EUR on cheap vehicles. The position labels (`low / medium / high`) are advisory — apimax/DealerMAX UI treat them as opaque slots ordered by intent, not by numeric ordering.

A typical Italian NLT preset is:

Slot	`percent_of_list`	`fixed_eur`	Meaning
`low`	`0`	`0`	Zero anticipo, headline canon
`medium`	`12.5`	`0`	Medium trade-in
`high`	`25`	`0`	Standard vetrina anticipo (matches the canon returned by `/nlt/offers` and `monthly_canon_from_eur`)

Image mode

Mode	Behavior
`branded`	Per-dealer composite (silver vehicle + dealer-branded background, baked at provisioning by the `azurenet-engine` workers). Falls back to `nlt_offerte.default_img` when the composite is not yet generated for that vehicle model.
`scenario_locked`	A single AI scenario fixed by the dealer. **Requires** `image_scenario_locked` to be one of the four valid scenarios. Same scenario applies to every NLT offer of this dealer.
`scenario_seasonal`	AI scenario auto-rotated by Italian season: `mediterraneo` (jun–aug), `milano` (sep–nov), `cortina` (dec–feb), `showroom` (mar–may).

The selected mode applies identically to the listing `image_url` and to the detail `image_url` for the same offer — the partner SDK sees a single coherent cover per offer.

Response

`200 OK`

```
HTTP/1.1 200 OK
Content-Type: application/json
```


```
{
  "dealer_id": "verdicar-001",
  "agency_markup_percent": 5.0,
  "down_payment_tiers": {
    "low":    { "percent_of_list": 0,    "fixed_eur": 0 },
    "medium": { "percent_of_list": 12.5, "fixed_eur": 0 },
    "high":   { "percent_of_list": 25,   "fixed_eur": 0 }
  },
  "image_mode": "branded",
  "image_scenario_locked": null,
  "currency": "EUR",
  "effective_from": "2026-05-17T23:10:42.180Z"
}
```


Field	Type	Description
`dealer_id`	string	Partner-owned opaque dealer id, echoed from the path.
`agency_markup_percent`	number	Echo of the persisted markup.
`down_payment_tiers`	object	Echo of the persisted tiers, percent + flat EUR per slot.
`image_mode`	string	Persisted mode.
`image_scenario_locked`	string | null	Persisted scenario (only set when mode = `scenario_locked`).
`currency`	string	Always `"EUR"` in v1.
`effective_from`	string	Timestamp at which the settings became authoritative. Cross-network AI surfaces re-price within ~5 minutes; the API itself returns the new prices immediately.

Error responses

Every non-2xx response follows RFC 7807 Problem Details (`Content-Type: application/problem+json`).

Status	`code`	Cause
`401`	`missing_api_key`	Neither `X-Api-Key` nor `Authorization: Bearer` was supplied.
`401`	`invalid_api_key`	Key not recognised, revoked, or attached to a non-partner account.
`401`	`expired_api_key`	Key past its `expires_at`.
`402`	`subscription_inactive`	Partner's Stripe subscription is not `active` / `trialing`.
`402`	`grace_period_expired`	Partner was in `past_due` / `unpaid` longer than the grace window.
`403`	`forbidden_dealer_not_owned`	`dealer_id` is not registered for this partner key.
`404`	`not_found`	No dealer with this ID, the user is not a `dealer` role, or the dealer was deleted.
`422`	`validation_error`	Field-level validation failed. Body includes `errors[]` with per-field `{field, code, message}` — typical paths: `down_payment_tiers.medium.percent_of_list`, `agency_markup_percent`, `image_scenario_locked`.

Example validation error:

```
HTTP/1.1 422 Unprocessable Entity
Content-Type: application/problem+json
```

```
{
  "type": "https://developers.dealermax.app/errors/validation_error",
  "title": "Request validation failed",
  "status": 422,
  "code": "validation_error",
  "detail": "One or more request fields failed validation. See `errors` for per-field issues.",
  "instance": "/v1/dealers/verdicar-001/nlt-settings",
  "errors": [
    {
      "field": "image_scenario_locked",
      "code": "value_error",
      "message": "Value error, image_scenario_locked_required_when_locked"
    }
  ]
}
```

Example — Python SDK (recommended)

```python
import os
from partnermax import Partnermax

client = Partnermax(api_key=os.environ["PARTNERMAX_API_KEY"])
settings = client.dealers.nlt_settings.update(
    dealer_id="verdicar-001",
    agency_markup_percent=5,
    down_payment_tiers={
        "low":    {"percent_of_list": 0,    "fixed_eur": 0},
        "medium": {"percent_of_list": 12.5, "fixed_eur": 0},
        "high":   {"percent_of_list": 25,   "fixed_eur": 0},
    },
    image_mode="branded",
)
print(settings.effective_from)
```

Example — curl

```
curl -X PATCH https://api.dealermax.app/v1/dealers/verdicar-001/nlt-settings \
  -H "X-Api-Key: $PARTNERMAX_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "agency_markup_percent": 5,
    "down_payment_tiers": {
      "low":    { "percent_of_list": 0,    "fixed_eur": 0 },
      "medium": { "percent_of_list": 12.5, "fixed_eur": 0 },
      "high":   { "percent_of_list": 25,   "fixed_eur": 0 }
    },
    "image_mode": "branded"
  }'
```

Official SDKs are published for Python (`pip install partnermax`, repo `DealerMax-app/partnermax-python`) and TypeScript / JavaScript (`npm install partnermax`, repo `DealerMax-app/partnermax-node`).

Notes

- The first successful call to this endpoint enrolls the active partner dealer reference into the PartnerMAX NLT API catalog (`GET /v1/dealers/{id}/nlt/offers`). It does not publish a real DealerMAX dealer site or consumer-facing AI surface.
- Subsequent calls update prices and image mode. Edge caches on consumer-facing surfaces refresh within ~5 minutes; the API itself returns the new prices immediately.
- To remove a dealer from the NLT catalog, deactivate the public dealer reference with `PATCH /v1/dealers/{dealer_id}` and body `{ "status": "inactive" }`. The direct registry equivalent is `POST /api/partner/dealers/{external_dealer_id}/suspend`, documented in Dealer Management. There is no `DELETE /nlt-settings` endpoint.

`GET /v1/dealers/{dealer_id}/nlt-settings`

Fetch the current NLT settings for a dealer. The response shape is identical to the `PATCH` response. A dealer that has never been configured returns the catalog defaults — `agency_markup_percent: 2`, tiers `low {percent_of_list: 0, fixed_eur: 0}` / `medium {percent_of_list: 12.5, fixed_eur: 0}` / `high {percent_of_list: 25, fixed_eur: 0}`, `image_mode: "branded"` — rather than a 404, so partners can call this endpoint to discover the starting point before sending their first `PATCH`. (Defaults are owned by `app/services/nlt_calculator.py::DEFAULT_ANTICIPI` and replicate the pre-2026-05-14 hardcoded scenario.)

Authentication

Same as the `PATCH`. The `dealer_id` must belong to the calling partner.

Request

Path parameters

Name	Type	Required	Description
`dealer_id`	string	yes	Partner-owned opaque dealer id, e.g. `verdicar-001`.

Response

`200 OK`

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
  "dealer_id": "verdicar-001",
  "agency_markup_percent": 5.0,
  "down_payment_tiers": {
    "low":    { "percent_of_list": 0,    "fixed_eur": 0 },
    "medium": { "percent_of_list": 12.5, "fixed_eur": 0 },
    "high":   { "percent_of_list": 25,   "fixed_eur": 0 }
  },
  "image_mode": "branded",
  "image_scenario_locked": null,
  "currency": "EUR",
  "effective_from": "2026-05-17T23:10:42.180Z"
}
```

Error responses

Status	`code`	Cause
`401`	`missing_api_key` / `invalid_api_key` / `expired_api_key`	Same auth rules as the `PATCH`.
`402`	`subscription_inactive` / `grace_period_expired`	Partner subscription gating.
`403`	`forbidden_dealer_not_owned`	`dealer_id` is not registered for this partner key.
`404`	`not_found`	No dealer with this ID, or the user is not a `dealer` role.

Example — Python SDK (recommended)

```
settings = client.dealers.nlt_settings.retrieve("verdicar-001")
print(settings.agency_markup_percent, settings.image_mode)
```

Example — curl

```
curl https://api.dealermax.app/v1/dealers/verdicar-001/nlt-settings \
  -H "X-Api-Key: $PARTNERMAX_API_KEY"
```

Worked example

A partner runs:

```
PATCH /v1/dealers/verdicar-001/nlt-settings
```

```
{
  "agency_markup_percent": 5,
  "down_payment_tiers": {
    "low":    { "percent_of_list": 0,    "fixed_eur": 0 },
    "medium": { "percent_of_list": 12.5, "fixed_eur": 0 },
    "high":   { "percent_of_list": 25,   "fixed_eur": 0 }
  },
  "image_mode": "scenario_locked",
  "image_scenario_locked": "mediterraneo"
}
```

Take a real offer — Volkswagen T-Cross 1.0 TSI 115cv DSG, list price `30 591,78 €` IVA inclusa, base canon `400 €/mese` at 48 months × 10 000 km/year. The dealer's headline canon (high tier) is computed as:

```
listino_imponibile  = 30 591,78 / 1,22       = 25 075,23 €
provvigione         = 25 075,23 × 5 / 100    =  1 253,76 €
anticipo_high       = 25 075,23 × 25 / 100   =  6 268,81 €
canon_imponibile    = 400 + 1 253,76 / 48
                          − 6 268,81 / 48     =      311,52 €
```

The T-Cross offer carries `private_only = false`, so no further 22% VAT is added — the consumer-facing canon displayed in the listing and detail is **311,52 €/mese (IVA esclusa)**. A `private_only = true` offer (e.g. the Fiat Pandina shown in NLT catalog) would be displayed as `311,52 × 1,22 = 380,06 €/mese (IVA inclusa)`.

The same offer with `medium` and `low` tiers yields the matrix:

Tier	`anticipo_tier_eur`	Canon imponibile	Canon esposto (B2B)
`high`	6 268,81 €	311,52 €/mese	311,52 €
`medium`	3 134,40 €	376,82 €/mese	376,82 €
`low`	0,00 €	426,12 €/mese	426,12 €

All three values are returned in the offer detail under `quotations[]` (for the headline tier — `high`) and `down_payment_scenarios_eur` (for the three downpayment amounts the consumer can choose from). The image rendered in both listing and detail for this offer will be the *Mediterraneo* AI scenario from `mnet_modelli_ai_foto` because the dealer set `image_mode = "scenario_locked"`.

CHAPTER 09

API Reference — NLT Catalog

> *`dealer_id`* *is the partner-supplied external dealer id created through* *`client.dealers.create(...)`* / *`POST /api/partner/dealers`*.

Dealer-aware read endpoints for the NLT (Italian *Noleggio Lungo Termine*, long-term car rental) catalog. Both endpoints are scoped to a specific `dealer_id` so the prices reflect that dealer's NLT settings: the agency mark-up is amortized over the duration, each of the dealer's three down-payment tiers is applied, and the cover image is rendered according to the dealer's `image_mode`.

Overview

The NLT catalog is the network-wide pool of long-term rental offers negotiated between DealerMAX and the rental networks (Leasys, Athlon, Arval, ALD, …). Every offer is normalized into a uniform shape:

- An **18-quotation matrix** — 3 durations (`36`, `48`, `60` months) × 6 km tiers (`10 000`, `15 000`, `20 000`, `25 000`, `30 000`, `40 000` km/year). Each cell is priced with the dealer's **high** down-payment tier (the most aggressive trade-in, lowest visible canon).
- The dealer's **3 down-payment scenarios** (`zero`, `medium`, `standard`) at the offer's list price, returned separately so consumer UIs can let the customer toggle between them.
- A stable `offer_id`, a stable `slug` (used in canonical URLs), an `image_url` composed for the dealer's `image_mode`, and a `canonical_url` pointing at the dealer's consumer site.

The two endpoints:

- `GET /v1/dealers/{dealer_id}/nlt/offers` — paginated listing with summary records.
- `GET /v1/dealers/{dealer_id}/nlt/offers/{offer_id}` — full detail for one offer, including the 18-cell pricing matrix, the full Motornet technical sheet, the network of partner dealers fulfilling this offer, the Italian FAQ block, and the offer-level VAT flag.

The shape of the detail endpoint mirrors apimax MCP `_tool_get_nlt_offer_details` bit-for-bit so the partner SDK consumer sees the same payload that Custom GPT and NLWeb agents already consume.

`GET /v1/dealers/{dealer_id}/nlt/offers`

List NLT offers available to a specific dealer, with the dealer's prices already applied.

Authentication

`X-Api-Key` (recommended) or `Authorization: Bearer <key>`. The `dealer_id` must belong to the calling partner.

Request

Path parameters

Name	Type	Required	Description
`dealer_id`	string	yes	Partner-owned opaque dealer id, e.g. `verdicar-001`.

Query parameters

Name	Type	Required	Description
`cursor`	string	no	Opaque base64url-encoded cursor returned in the previous response's `next_cursor`.
`limit`	integer	no	`1` – `50`. Default `25`.
`brand`	string	no	Case-insensitive `ILIKE` match on `nlt_offerte.marca`. Example: `?brand=fiat`.
`fuel_type`	string	no	Raw Italian label, case-insensitive `ILIKE %value%`. Examples: `Benzina`, `Diesel`, `Ibrido benzina`, `Ibrido diesel`, `Elettrica`, `GPL`, `Metano`. **No enum normalization** — apimax-aligned.
`segment`	string	no	Raw Italian label, case-insensitive `ILIKE %value%`. Examples: `SUV piccoli`, `SUV medi`, `Superiori`, `Medie`, `Utilitarie`.
`canone_max_eur`	integer	no	Filter to offers whose displayed monthly canon `monthly_canon_from_eur` is at or below this value (EUR).
`duration_months`	integer	no	Restrict the headline canon to a specific duration. One of `36`, `48`, `60`.
`km_per_year`	integer	no	Restrict the headline canon to a specific km tier. One of `10000`, `15000`, `20000`, `25000`, `30000`, `40000`.

When `duration_months` or `km_per_year` are supplied, the `monthly_canon_from_eur` in each summary is computed against the matching cell of the 18-quotation matrix. When omitted, it reflects the cheapest cell across all 18.

Headers

Header	Required	Description
`X-Api-Key`	conditional	Required if not using `Authorization: Bearer`.
`Authorization`	conditional	`Bearer <key>` alternative.

Response

`200 OK`

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
  "data": [
    {
      "offer_id": "381",
      "slug": "business-volkswagen-t-cross-t-cross-1-0-tsi-edition-plus-115cv-dsg",
      "brand": "Volkswagen",
      "model": "T-Cross",
      "trim": "T-Cross 1.0 tsi Edition Plus 115cv dsg",
      "fuel_type": "Benzina",
      "segment": "SUV piccoli",
      "monthly_canon_from_eur": 244.12,
      "duration_months": 48,
      "km_per_year_at_quote": 10000,
      "dealer_id": "verdicar-001",
      "vat_treatment": "business",
      "image_url": "https://cdn.azcore.it/storage/v1/render/image/public/modelli-ai/3141/mediterraneo-c536791f.webp?width=1
      "canonical_url": "https://rossiautomobili.it/noleggio-lungo-termine/business-volkswagen-t-cross-t-cross-1-0-tsi-editi
      "has_promo": false
    },
    {
      "offer_id": "394",
      "slug": "privati-fiat-pandina-pandina-1-0-firefly-hybrid-s-s-70cv",
      "brand": "Fiat",
      "model": "Pandina",
      "trim": "Pandina 1.0 firefly hybrid s&s 70cv",
      "fuel_type": "Ibrido benzina",
      "segment": "Superutilitarie",
      "monthly_canon_from_eur": 191.48,
      "duration_months": 48,
      "km_per_year_at_quote": 30000,
      "dealer_id": "verdicar-001",
      "vat_treatment": "private",
      "image_url": "https://cdn.azcore.it/storage/v1/render/image/public/modelli-ai/3267/mediterraneo-01766c08.webp?width=1
      "canonical_url": "https://rossiautomobili.it/noleggio-lungo-termine/privati-fiat-pandina-pandina-1-0-firefly-hybrid-s
      "has_promo": false
    }
  ],
  "has_more": true,
  "next_cursor": "Mzk0"
}
```

Field	Type	Description
`offer_id`	string	Numeric `nlt_offerte.id_offerta` as string. Use as the path parameter for the detail endpoint.
`slug`	string	Offer slug used in canonical URLs (`/noleggio-lungo-termine/{slug}`).
`brand`	string	Manufacturer brand.
`model`	string	Model name.
`trim`	string | null	Trim/version label as marketed by the manufacturer.
`fuel_type`	string | null	Raw Italian label from `nlt_offerte.alimentazione` (e.g. `Benzina` , `Ibrido diesel`). No enum normalization — apimax-aligned.
`segment`	string | null	Raw Italian label from `nlt_offerte.segmento` (e.g. `SUV piccoli` , `Superiori`).
`monthly_canon_from_eur`	number	Lowest displayed monthly canon across the 18-cell matrix, computed at the dealer's **high** down-payment tier with the dealer's mark-up applied.
`duration_months`	integer	The duration of the headline cell. One of `36` , `48` , `60` .
`km_per_year_at_quote`	integer	The km tier of the headline cell.
`dealer_id`	string	Partner-owned dealer id (echo of the path parameter).
`vat_treatment`	string enum	`"private"` or `"business"` . Property **of the offer** (sourced from `nlt_offerte.solo_privati`), not of the dealer. When `"private"` , `monthly_canon_from_eur` is VAT-inclusive (×1.22).
`image_url`	string | null	Absolute URL to the cover image. Composed for the dealer's `image_mode` (see NLT settings). The same URL is returned by the detail endpoint for the same offer.
`canonical_url`	string | null	Absolute URL to the consumer-facing page on the dealer's site: `https://{primary_domain}/noleggio-lungo-termine/{slug}` . `null` when the dealer has no `dealer_site_public` row yet.
`has_promo`	boolean	`true` if the offer carries a network promotion (presence of an `nlt_offerta_tag` row).
`has_more`	boolean	`true` if a `next_cursor` was issued — more offers exist beyond this page.
`next_cursor`	string | null	Opaque base64url cursor for the following page. Pass back as `?cursor=` .

Error responses

Every non-2xx response follows RFC 7807 Problem Details.

Status	`code`	Cause
`401`	`missing_api_key` / `invalid_api_key` / `expired_api_key`	Auth checks.
`402`	`subscription_inactive` / `grace_period_expired`	Partner subscription gating.
`403`	`forbidden_dealer_not_owned`	`dealer_id` is not registered for this partner key.
`404`	`not_found`	No dealer with this ID, or the user is not a `dealer` role.
`422`	`validation_error`	One of the query parameters is malformed (e.g. non-integer `limit` , `cursor` not parseable).

Example — Python SDK (recommended)

```python
import os
from partnermax import Partnermax

client = Partnermax(api_key=os.environ["PARTNERMAX_API_KEY"])
listing = client.dealers.nlt.offers.list(
    dealer_id="verdicar-001",
    segment="SUV piccoli",
    canone_max_eur=300,
    duration_months=48,
    km_per_year=15000,
    limit=25,
)
for offer in listing.data:
    print(offer.brand, offer.model, offer.monthly_canon_from_eur, offer.canonical_url)
```

Example — curl

```bash
curl "https://api.dealermax.app/v1/dealers/verdicar-001/nlt/offers?segment=SUV%20piccoli&canone_max_eur=300&duration_months
  -H "X-Api-Key: $PARTNERMAX_API_KEY"
```

Official SDKs are published for Python (`pip install partnermax` , repo `DealerMax-app/partnermax-python`) and TypeScript / JavaScript (`npm install partnermax` , repo `DealerMax-app/partnermax-node`).

Notes

- `canonical_url` is `null` until the dealer has provisioned a public site (`dealer_site_public.primary_domain`). Once provisioned, the URL is stable across price refreshes.
- The summary record does **not** include the full pricing matrix or technical sheet — call the detail endpoint when the consumer drills into a specific offer.
- Geographic and brand-exclusivity restrictions are **not** enforced in v1: every active offer is returned for every owned dealer.

`GET /v1/dealers/{dealer_id}/nlt/offers/{offer_id}`

Fetch the full detail of one NLT offer, priced for a specific dealer.

The response shape mirrors apimax MCP `_tool_get_nlt_offer_details` 1:1 (with field names translated to American-English snake_case for the SDK contract — values stay in raw Italian). Partner SDK consumers see the same payload that the Custom GPT and NLWeb agents already receive across the DealerMAX network.

Authentication

Same as the listing endpoint. The `dealer_id` must belong to the calling partner.

Request

Path parameters

Name	Type	Required	Description
`dealer_id`	string	yes	Partner-owned opaque dealer id, e.g. `verdicar-001`.
`offer_id`	string	yes	Numeric `nlt_offerte.id_offerta` as string. Accepts the bare integer or the `ofr_<n>` prefix returned by the listing.

Response

`200 OK`

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
  "found": true,
  "offer_id": "381",
  "slug": "business-volkswagen-t-cross-t-cross-1-0-tsi-edition-plus-115cv-dsg",
  "title": "Volkswagen T-Cross T-Cross 1.0 tsi Edition Plus 115cv dsg",
  "description_short": "Tutti i colori disponibili",
  "description_full": "La **Volkswagen T-Cross** è un SUV compatto …",
  "image_url": "https://cdn.azcore.it/storage/v1/render/image/public/modelli-ai/3141/mediterraneo-c536791f.webp?width=1200&
  "gallery": [
    {
      "url": "https://cdn.azcore.it/storage/v1/render/image/public/mnet-immagini/A104331/11218.jpg?width=1200&height=800&qu
      "is_cover": true
    },
    {
      "url": "https://cdn.azcore.it/storage/v1/render/image/public/mnet-immagini/A104331/11220.jpg?width=1200&height=800&qu
      "is_cover": false
    }
  ],
  "brand": "Volkswagen",
  "model": "T-Cross",
  "trim": "T-Cross 1.0 tsi Edition Plus 115cv dsg",
  "fuel_type": "Benzina",
  "transmission": "Automatico sequenziale",
  "segment": "SUV piccoli",
  "private_only": false,
  "total_price_eur": 30591.78,
  "quotations": [
    { "duration_months": 48, "km_per_year": 10000, "monthly_canon_eur": 223.0 },
    { "duration_months": 48, "km_per_year": 15000, "monthly_canon_eur": 244.0 },
    { "duration_months": 48, "km_per_year": 20000, "monthly_canon_eur": 264.0 },
    { "duration_months": 60, "km_per_year": 10000, "monthly_canon_eur": 241.0 }
  ],
  "min_monthly_canon_eur": 223.0,
  "primary_dealer_name": "Rossi Automobili",
  "primary_dealer_city": "Milano",
  "primary_dealer_province": "MI",
  "down_payment_scenarios_eur":    { "zero": 0, "medium": 3134, "standard": 6269 },
  "down_payment_scenarios_labels": { "zero": "Senza anticipo", "medium": "Anticipo 12,5%", "standard": "Anticipo 25%" },
  "vat_included": false,
  "tags": [
    { "name": "Pronta Consegna", "icon": "fa-truck-fast", "color": "#20C997" }
  ],
  "included_accessories": [],
  "available_addons": {
    "tires":               { "diameter_in": 17, "set_cost_eur": 864.0, "replacement_rule": "1 treno (4 pneumatici) ogni 30.
    "replacement_vehicle": { "default_category": "B", "category_description": "Utilitaria (per fermo macchina)", "monthly_c
  },
  "network_offers": [
    {
      "dealer_id": "verdicar-001",
      "dealer_name": "Rossi Automobili",
      "city": "Milano",
      "province": "MI",
      "min_monthly_canon_eur": 223.0,
      "rating_value": 4.8,
      "review_count": 312,
      "contact_url": "https://rossiautomobili.it/",
      "phone": "+390298765432",
      "google_maps_url": "https://www.google.com/maps/place/?q=place_id:ChIJExampleLatitudeLongitudeOpaqueIdentifier"
    }
  ],
  "network_dealer_count": 1,
  "technical_details": {
    "alimentazione": "Benzina",
    "cilindrata": 999,
    "hp": 115,
    "kw": 85,
    "cambio": "Automatico sequenziale",
```

```
    "lunghezza": 414,
    "larghezza": 176,
    "altezza": 160,
    "passo": 255,
    "peso": 1305,
    "bagagliaio": "455/1281",
    "trazione": "Anteriore",
    "euro": "6"
  },
  "standard_equipment": [],
  "included_services": [
    { "name": "Assicurazione RCA", "description": "Responsabilità Civile Auto" },
    { "name": "Assicurazione Danni", "description": "Kasco - Atti Vandalici - Eventi Naturali - Cristalli" },
    { "name": "Manutenzione", "description": "Manutenzione ordinaria e Straordinaria" },
    { "name": "Assistenza Stradale", "description": "Soccorso e recupero 24 / 7" }
  ],
  "faqs": [
    {
      "question": "Quanto è lunga la Volkswagen T-Cross?",
      "answer": "La Volkswagen T-Cross ha una lunghezza di 4.14 m – larghezza 1.76 m, altezza 1.60 m."
    },
    {
      "question": "Quanto è grande il bagagliaio della Volkswagen T-Cross?",
      "answer": "Il bagagliaio della Volkswagen T-Cross ha una capacità di 455/1281 litri (con sedili in posizione standard
    }
  ],
  "last_modified": "2026-05-04T15:49:16.806390"
}
```


The `quotations` array contains up to 18 entries (3 durations × 6 km tiers). Cells whose underlying base canon is `NULL` in the catalog, or whose computed monthly canon falls below the plausibility threshold of €50, are dropped — partial coverage is honest data, not a bug. The example above is truncated for readability.

Field reference

Field	Type	Description
`found`	boolean	Always `true` on a 200 response. Reserved for future “soft 404” payloads.
`offer_id`	string	Numeric `id_offerta` as string. Same value returned by the listing.
`slug`	string	Offer slug.
`title`	string	`"{brand} {model} {trim}"` joined and trimmed.
`description_short`	string | null	Free-form Italian short copy from `nlt_offerte.descrizione_breve`.
`description_full`	string | null	AI-generated long-form Italian description (markdown).
`image_url`	string | null	Cover image, **identical to the listing summary** for this offer.
`gallery[]`	array	Image list from `mnet_immagini` for this `codice_motornet_uni`, deduped by `codice_visuale` keeping the H-resolution row, ordered by visual priority (frontale → tre-quarti → laterale → posteriore → interni). First entry has `is_cover: true`.
`brand` / `model` / `trim`	string | null	Raw `nlt_offerte.marca / modello / versione`.
`fuel_type`	string | null	Raw Italian label from `nlt_offerte.alimentazione`.
`transmission`	string | null	Raw Italian label from `nlt_offerte.cambio` (e.g. `Automatico sequenziale`).
`segment`	string | null	Raw Italian label from `nlt_offerte.segmento`.
`private_only`	boolean | null	`true` for B2C-only offers (VAT-inclusive canons). Sourced from `nlt_offerte.solo_privati`.
`total_price_eur`	number | null	List price IVA-inclusive (vehicle + accessories + MSS).
`quotations[]`	array	Each entry: `{duration_months, km_per_year, monthly_canon_eur}`. Computed with the dealer’s mark-up + high down-payment tier. VAT-inclusive when `private_only=true`.
`min_monthly_canon_eur`	number | null	Lowest canon for the addressed dealer in this offer’s quotation matrix.
`primary_dealer_name` / `city` / `province`	string | null	Technical display label and location for the addressed dealer. Partner-created technical dealers use the external id as their safe label unless enriched by an explicit future claim flow.
`down_payment_scenarios_eur`	object | null	The three EUR amounts of the dealer’s tiers applied to this offer’s list price. Keys: `zero`, `medium`, `standard`.
`down_payment_scenarios_labels`	object | null	Human-readable Italian labels paired with the EUR amounts: `"Senza anticipo"`, `"Anticipo 12,5%"`, `"Anticipo 25%"` for the default tiers, or dynamic strings derived from the dealer’s custom `percent_of_list` + `fixed_eur` values.
`vat_included`	boolean	Convenience mirror of `private_only` (`true` when the displayed canon is VAT-inclusive).

Field	Type	Description
`tags[]`	array	Network category tags (e.g. `Pronta Consegna`, `Promo`, `GreenChoice`). Each carries `{name, icon, color}`.
`included_accessories[]`	array	Accessories bundled with the offer (entries from `nlt_offerta_accessori`): `{code, description, extra_price_eur}`.
`available_addons`	object	Optional add-ons the partner can quote on top. `tires` carries `{diameter_in, set_cost_eur, replacement_rule}`; `replacement_vehicle` carries `{default_category, category_description, monthly_cost_eur}`. Either is `null` if upstream data is missing.
`network_offers[]`	array	Up to 100 of the partner's active dealers fulfilling this offer, sorted by `min_monthly_canon_eur` ascending. Each carries `{dealer_id, dealer_name, city, province, min_monthly_canon_eur, rating_value, review_count, contact_url, phone, google_maps_url}`.
`network_dealer_count`	integer	Total eligible active partner dealers for this offer; it may be greater than `network_offers.length` when the network exceeds the response cap.
`technical_details`	object	Flat dictionary of every non-null Motornet column for this `codice_motornet_uni`. **Keys are raw SQL column names in Italian** (`cilindrata`, `kw`, `lunghezza`, `bagagliaio`, `emissioni_co2`, `pneumatici_anteriori`, …) — they reflect the underlying `mnet_dettagli` schema. Up to ~90 keys available; ~30–40 typically populated per row. Native units preserved (cc, mm, kg, kW, CV, s, g/km).
`standard_equipment[]`	array	Free-text equipment list from `mnet_dettagli.equipaggiamento`. Currently empty on every live offer (upstream column unpopulated); will auto-fill when the data flows in.
`included_services[]`	array	Services normally bundled in the canone (apimax `nlt_services` is_active). Each entry: `{name, description}`.
`faqs[]`	array	Italian Q&A generated from the offer payload + Motornet specs. Up to ~11 entries: dimensioni, bagagliaio, CO2/Euro, motore, posti/porte, prestazioni, canone preset 48mo/15k, canone minimo, durate disponibili, IVA, anticipi.
`last_modified`	string | null	ISO timestamp of the most recent change to the underlying `nlt_offerte` row.

Error responses

Status	`code`	Cause
`401`	`missing_api_key` / `invalid_api_key` / `expired_api_key`	Auth checks.
`402`	`subscription_inactive` / `grace_period_expired`	Partner subscription gating.
`403`	`forbidden_dealer_not_owned`	`dealer_id` is not registered for this partner key.
`404`	`not_found`	No dealer with this ID, or the user is not a `dealer` role.
`404`	`offer_not_found`	No active offer with this ID, or the offer is not priceable (no priceable quotation cell).

Example — Python SDK (recommended)

```python
offer = client.dealers.nlt.offers.retrieve(
    dealer_id="verdicar-001",
    offer_id="ofr_381",
)

# Pick the 48-month / 10 000 km cell.
cell = next(
    q for q in offer.quotations
    if q.duration_months == 48 and q.km_per_year == 10000
)
print(f"{offer.brand} {offer.model}: {cell.monthly_canon_eur} €/mese ({'IVA inclusa' if offer.vat_included else 'IVA esclus

# Show the three down-payment scenarios at the offer's list price.
sc = offer.down_payment_scenarios_eur
lb = offer.down_payment_scenarios_labels
print(f"{lb.zero}: {sc.zero}€ — {lb.medium}: {sc.medium}€ — {lb.standard}: {sc.standard}€")
```

Example — curl

```bash
curl https://api.dealermax.app/v1/dealers/verdicar-001/nlt/offers/ofr_381 \
  -H "X-Api-Key: $PARTNERMAX_API_KEY"
```

Notes

- The detail response is typically 6–20 KB depending on how many Motornet columns and quotations the offer has populated.
- `image_url` is the **same** URL the listing summary returns for the same offer, composed against the dealer's `image_mode`. Use it directly in galleries / hero crops without recomputing.
- `technical_details` keys stay Italian because they are raw `mnet_dettagli` column names — they are documented in appendix/glossary.md for partner client developers.
- Offers can be withdrawn from the network catalog at any time. A consumer-facing URL that resolved last week may return `404` today; refetch the offer detail before rendering high-stakes UI.

CHAPTER 10

API Reference — Used Vehicles

> *Used-vehicle endpoints are part of the PartnerMAX public SDK contract. The partner calls `https://api.dealermax.app` only; internal service names are implementation details.*

Provision and manage used-vehicle stock through the partner API. The endpoints expose the full lifecycle, scoped to an external dealer id registered for the authenticated partner through `client.dealers.create(...)`:

- `POST /v1/dealers/{dealer_id}/vehicles` — create a vehicle in stock.
- `GET /v1/dealers/{dealer_id}/vehicles` — list, with filters and cursor pagination.
- `GET /v1/dealers/{dealer_id}/vehicles/{vehicle_id}` — full detail with derived Motornet specs.
- `PATCH /v1/dealers/{dealer_id}/vehicles/{vehicle_id}` — partial update; explicit `null` clears nullable fields.
- `DELETE /v1/dealers/{dealer_id}/vehicles/{vehicle_id}` — soft-delete; the plate becomes reusable on the network.
- `GET/POST/PUT /v1/dealers/{dealer_id}/vehicles/{vehicle_id}/accessories` — read Motornet optionals/packages/equipment and replace the partner-managed selections.

Why Motornet is the canonical key

The partner identifies every vehicle by its **Motornet UNI code** (`motornet_code`). The partnermax surface does **not** accept free-text `brand`, `model`, `trim`, `fuel_type` and so on — those are derived server-side from DealerMAX's licensed Motornet-backed catalogue. This eliminates an entire class of data-quality bugs (two partners spelling "Volkswagen" / "VolksWagen" / "VW" differently) and matches a published platform rule: *"Motornet is authoritative, dealers do not fill in catalogue fields"*.

In practice this means the SDK can create used vehicles only after the partner has a Motornet UNI code for that vehicle. There are two supported ways to get that code:

1. Use the partner's own Motornet data access. Motornet publishes the catalogue products at motornet.it/prodotti/banchedati. Partners with a direct Motornet agreement may send v2 auto codes or v3/VCOM commercial-vehicle codes directly as `motornet_code`.
2. Use the DealerMAX paid resolver. Send the vehicle plate/targa or VIN/telaio to `https://api.dealermax.app`; the resolver consumes PartnerMAX credits and returns the Motornet UNI code to use in `client.dealers.vehicles.create(...)`.

DealerMAX does not expose full Motornet catalogue browsing through PartnerMAX. The resolver is per vehicle: it accepts an identifier, returns candidate versions, and leaves the final version choice to the partner.

Paid resolver endpoints:

```
GET https://api.dealermax.app/v1/vehicles/resolve/by-targa/{targa}?catalog=auto
GET https://api.dealermax.app/v1/vehicles/resolve/by-targa/{targa}?catalog=vcom
GET https://api.dealermax.app/v1/vehicles/resolve/by-vin/{vin}?catalog=auto
GET https://api.dealermax.app/v1/vehicles/resolve/by-vin/{vin}?catalog=vcom
```

Use the same Partner key as the SDK (`X-Api-Key` or `Authorization: Bearer`). `catalog=auto` resolves against Motornet v2 auto data; `catalog=vcom` resolves against Motornet v3/VCOM commercial-vehicle data. The response contains `versioni[].codice_motornet_uni`; pass the selected value as `motornet_code` to `POST /v1/dealers/{dealer_id}/vehicles`. The resolver is per-vehicle, paid, and does not expose catalogue browsing or search.

You send (~12 vehicle-specific fields)	You get back (50+ fields derived from Motornet)
`motornet_code`, `plate`, `vin`, `registration_year`, `certified_km`, `enabled_channels`, prices, condition/service fields, editorial fields	`brand`, `model`, `trim`, `fuel_type`, `cilindrata`, `hp`, `kw`, `cavalli_fiscali`, `consumo_medio`, `emissioni_co2`, `lunghezza`, `bagagliaio`, `accelerazione`, full `technical_details` dict + the AI-generated `ai_content` block (descriptions, highlights, FAQ, SEO) once the worker has processed the vehicle

Once a code is in hand, either from the partner's own Motornet access or from the paid targa/VIN lookup, it is stable across the vehicle's lifetime.

A `motornet_code` not available in the DealerMAX auto/VCOM vehicle catalogue returns `422 motornet_code_not_in_catalogue` at creation time — there is no upstream Motornet API call from partnermax (no per-call cost, no upstream network failure mode in the request path).

The async generation pipeline

After a successful `POST`, the stock record is committed and the AI-content worker takes over:

1. Within **60 seconds**, `azurenet-engine` picks up the new vehicle.
2. A single gpt-5 call with structured JSON output generates the full SEO body (title, short description, long description, key features, FAQs, social captions, alt texts, hashtags) and a pgvector embedding.
3. The result is persisted in the shared AI-content layer. From that point onward the vehicle is **discoverable on every AI surface**: - **MCP server** (`mcp.dealermax.app/mcp`) — semantic search via `_tool_search_vehicles`, single-vehicle lookup via `_tool_get_vehicle_details`. - **Custom GPT "DealerMAX Italia"** — Actions `search_vehicles_network` and `get_vehicle_details`. - **NLWeb** `/ask` — list + summarize modes. - **Dealer site JSON-LD** — `Vehicle` / `Car` Schema.org per page. - **Sitemap.xml** + **llms.txt** of the dealer site — refreshed within 5–15 minutes.

End-to-end SLO from `POST` to indexed-on-every-AI-surface: typically **under 2 minutes**. The partner's `POST` returns immediately — no synchronous wait on the generation worker.

Dealer reference, identity and ACL

Every vehicle endpoint enforces three layers of ACL in this order:

1. The partner API key authenticates (see Authentication).
2. The path-parameter `dealer_id` is the external dealer id registered for the calling partner.
3. PartnerMAX writes stock against the hidden technical dealer linked by the partner dealer registry. That technical dealer stays under the admin/root hierarchy, not under the partner.
4. For vehicle-scoped operations (GET-by-id / PATCH / DELETE), the path `vehicle_id` must be stocked by that resolved technical dealer; otherwise `404 vehicle_not_found`. The same 404 is returned for missing-row and cross-partner access to prevent enumeration.

Vehicle IDs are opaque strings of the shape `vhc_<uuid>`. Treat the full string as opaque and round-trip it without parsing.

`POST /v1/dealers/{dealer_id}/vehicles`

Provision a new vehicle in a dealer's stock.

Authentication

Standard partner API key (`X-Api-Key` or `Authorization: Bearer`). No special capability is required — capability gates only apply to key-lifecycle endpoints.

Request

Path parameters

Name	Type	Required	Description
`dealer_id`	string	yes	External dealer id registered for this partner, for example `verdicar-001`. Every vehicle belongs to exactly one dealer and the dealer id is always supplied in the path.

Headers

Header	Required	Description
`X-Api-Key`	conditional	Required if not using `Authorization: Bearer`.
`Content-Type`	yes	`application/json`.
`Idempotency-Key`	strongly recommended	Any string ≤256 chars. A retry with the same key and identical body replays the original 201 response; same key with a different body returns `409 idempotency_key_reuse`.

Body


```
{
  "motornet_code": "VW0035478",
  "plate": "FA123XY",
  "vin": "WVWZZZ1KZJW123456",
  "registration_year": 2022,
  "registration_month": 6,
  "certified_km": 28500,
  "enabled_channels": ["rewind"],
  "sale_price_eur": 18900.0,
  "cost_price_eur": 17000.0,
  "color": "Bianco perla",
  "base_color": "Bianco",
  "alloy_wheel_size": 17,
  "service_history_available": true,
  "last_service_date": "2025-12-01",
  "last_service_km": 24000,
  "inspection_due_date": "2027-06-30",
  "last_inspection_date": "2025-06-30",
  "last_inspection_km": 22000,
  "property_tax_due_date": "2026-12-31",
  "ownership_transfer_date": "2025-06-15",
  "double_keys_available": true,
  "vehicle_damaged": null,
  "damage_repaired": null,
  "previous_owner_count": 1,
  "fuel_type_override": null,
  "power_kw_override": null,
  "wltp_consumption_combined_l_100km": 5.7,
  "co2_emissions_g_km_override": 126,
  "vat_displayed": false,
  "trim_alias": "Edition Plus",
  "description": "Tagliando appena fatto. Garanzia 24 mesi inclusa.",
  "notes": "Mai usata in pista; trasferimento di proprietà di un anno fa."
}
```

Field	Type	Required	Constraints
`motornet_code`	string	yes	Must exist in the DealerMAX auto/VCOM vehicle catalogue: Motornet v2 auto codes or v3/VCOM commercial-vehicle codes. The configuration's technical specs (brand, model, fuel type, dimensions, CO2) are derived from this code.
`plate`	string	yes	Italian plate (5–8 alphanumeric chars). Uppercased server-side. Unique across active vehicles on the network.
`vin`	string | null	no	ISO 3779 (17 chars, no I/O/Q). Strongly recommended for B2B traceability.
`registration_year`	integer	yes	`[1960, current+1]`.
`registration_month`	integer | null	no	`[1, 12]`.
`certified_km`	integer	yes	`[0, 999_999]`.
`service_history_available`	bool	no	Default `false`. Dealer declaration that certified service history is available.
`last_service_date`	date | null	no	ISO date of the last service.
`last_service_km`	integer | null	no	`[0, 999_999]`. Odometer at last service.
`last_service_notes`	string | null	no	Free-form service notes, <=2000 chars.
`inspection_due_date`	date | null	no	Next inspection/revision due date.
`last_inspection_date`	date | null	no	Last inspection/revision date.
`last_inspection_km`	integer | null	no	`[0, 999_999]`. Odometer at last inspection/revision.
`property_tax_due_date`	date | null	no	Vehicle tax/bollo due date.
`ownership_transfer_date`	date | null	no	Last ownership-transfer date.
`double_keys_available`	bool	no	Default `false`.
`color`	string | null	no	Free text, max 64 chars.
`base_color`	string | null	no	Normalized base color, max 64 chars.
`alloy_wheel_size`	integer | null	no	`[13, 22]` inches.
`extended_warranty_enabled`	bool	no	Default `false`. Must be coherent with `extended_warranty_months` (both set or both unset).
`extended_warranty_months`	integer | null	conditional	Required when `extended_warranty_enabled=true`. Max 120 months.

Field	Type	Required	Constraints
`vehicle_damaged`	bool | null	no	Tri-state: `true` yes, `false` no, `null` unknown.
`damage_repaired`	bool | null	no	Tri-state repaired-damage declaration: `true` yes, `false` no, `null` unknown.
`previous_owner_count`	integer | null	no	`[0, 10]`.
`fuel_type_override`	string | null	no	Dealer override for fuel type, max 64 chars. Use only when Motornet needs a display correction.
`power_kw_override`	integer | null	no	`[1, 2000]`. Dealer override for kW.
`wltp_consumption_combined_l_100km`	number | null	no	`[0, 30]`.
`wltp_consumption_urban_l_100km`	number | null	no	`[0, 30]`.
`wltp_consumption_extraurban_l_100km`	number | null	no	`[0, 30]`.
`co2_emissions_g_km_override`	number | null	no	`[0, 500]`.
`enabled_channels`	array	no	Default `["rewind"]`. Allowed values: `rewind`, `nos`. At least one, no duplicates.
`sale_price_eur`	number | null	conditional	Required and `>=100` when `enabled_channels` contains `rewind`. May be omitted or `0` for NOS-only vehicles. Max `5_000_000`.
`cost_price_eur`	number | null	no	Dealer cost basis, `[0, 5_000_000]`. If omitted on create, defaults to `sale_price_eur`.
`vat_displayed`	bool	no	Default `false`. If `true`, the public price is displayed VAT-exposed (B2B).
`is_visible`	bool	no	Default `true`. When `false` the row exists in stock but is excluded from AI surfaces. Different from soft-delete — reversible via PATCH.
`trim_alias`	string | null	no	Dealer-facing trim/allestimento alias, max 255 chars.
`description`	string	no	Long description for the dealer site detail page, ≤5000 chars.
`notes`	string | null	no	Free-form short notes, ≤2000 chars.

Fields not in this surface. Motornet-derived catalogue fields such as `brand`, `model`, `trim`, `fuel_type`, displacement, dimensions and CO2 are still not accepted in the create body. DealerMax derives them from `motornet_code`. Optional, package and equipment selections are handled by the dedicated accessories endpoints below. Sending any unsupported field returns `422 validation_error` from the `extra="forbid"` guard.

Note: `metadata` is **not accepted in v1.x** — vehicle-side annotation storage ships in a later minor. Sending it returns `422 validation_error`.

Response

`201 Created` returns a `VehicleDetail` with `partner_id`, `dealer_id`, `vehicle_id`, and the full Motornet-derived `technical_details` dict.

Error responses

Status	`code`	Cause
`401`	`missing_api_key` / `invalid_api_key` / `expired_api_key`	Auth checks.
`402`	`subscription_inactive` / `grace_period_expired`	DealerMAX subscription not active.
`403`	`forbidden_dealer_not_owned`	`dealer_id` is not registered for this partner key.
`404`	`dealer_not_found`	No dealer with this ID under the calling partner.
`409`	`idempotency_key_reuse`	Idempotency key reused with a different body.
`409`	`vehicle_plate_already_registered`	Another active vehicle on the network already holds this plate.
`422`	`motornet_code_not_in_catalogue`	The supplied code is not available in the DealerMAX auto/VCOM vehicle catalogue.
`422`	`validation_error`	A body field failed validation; see `errors[]` for per-field detail.

Example — Python SDK (recommended)

```python
import os
from partnermax import Partnermax

client = Partnermax(api_key=os.environ["PARTNERMAX_API_KEY"])
vehicle = client.dealers.vehicles.create(
    dealer_id="verdicar-001",
    motornet_code="VW0035478",
    plate="FA123XY",
    vin="WVWZZZ1KZJW123456",
    registration_year=2022,
    registration_month=6,
    certified_km=28_500,
    enabled_channels=["rewind"],
    sale_price_eur=18_900.0,
    cost_price_eur=17_000.0,
    color="Bianco perla",
    base_color="Bianco",
    service_history_available=True,
    last_service_date="2025-12-01",
    last_service_km=24_000,
    vehicle_damaged=None,
    damage_repaired=None,
    previous_owner_count=1,
    description="Tagliando appena fatto. Garanzia 24 mesi inclusa.",
    idempotency_key=f"create-{partner_internal_sku}",
)
print(vehicle.vehicle_id, vehicle.brand, vehicle.model)
```

Example — curl

```
curl -X POST https://api.dealermax.app/v1/dealers/verdicar-001/vehicles \
  -H "X-Api-Key: $PARTNERMAX_API_KEY" \
  -H "Idempotency-Key: $(uuidgen)" \
  -H "Content-Type: application/json" \
  -d '{
    "motornet_code": "VW0035478",
    "plate": "FA123XY",
    "registration_year": 2022,
    "certified_km": 28500,
    "sale_price_eur": 18900.0
  }'
```

`POST /v1/dealers/{dealer_id}/vehicles/bulk`

Provision up to **100 vehicles** in a single synchronous call. The intended use is a partner's initial migration (where a several-thousand-vehicle inventory is chunked client-side) or a daily reconciliation push from the partner's DMS.

Why synchronous (and not background-job)

partnermax v1 deliberately uses a synchronous bulk endpoint instead of a file-upload + worker pattern. The trade-offs that drove the decision:

Aspect	Synchronous (this surface)	Async worker + tracking table
Response time for 100 rows	~20s	Immediate 202 + polling
Per-row error visibility	Inline in the 207 response	Stored separately, partner polls
Orphaned imports on container restart	impossible	"processing" forever until watchdog
Cross-repo worker complexity	none	requires changes in `azurenet-engine`
Predictability for the partner SDK	High — same call → same response	Low — polling loop, race on completion

If you regularly need to import thousands of vehicles at once, the chunking pattern below is the canonical approach. partnermax does not impose concurrency limits on the partner's parallel calls — the bottleneck is the partner's HTTP client.

Authentication

Same as the single-create POST. The dealer in the path must belong to the calling partner.

Request

Headers

Header	Required	Description
`X-Api-Key`	conditional	Required if not using `Authorization: Bearer`.
`Content-Type`	yes	`application/json`.
`Idempotency-Key`	strongly recommended	Any string ≤256 chars. The fingerprint covers the entire `vehicles[]` array; a retry with the same key + same array replays the original 207 response. Different array + same key returns `409 idempotency_key_reuse`. The fingerprint also includes the HTTP method and path, so reusing the same key across a single `POST /vehicles` and this bulk `POST /vehicles/bulk` does not collide — each path has its own cached envelope.

Body

```
{
  "vehicles": [
    {
      "motornet_code": "VW0035478",
      "plate": "FA123XY",
      "registration_year": 2022,
      "certified_km": 28500,
      "sale_price_eur": 18900.0
    },
    {
      "motornet_code": "FT0123456",
      "plate": "GG456WW",
      "registration_year": 2023,
      "certified_km": 15000,
      "sale_price_eur": 16400.0
    }
  ]
}
```

Each row carries the exact same shape and validation as the single-create POST body. Per-row validation failures **do not** abort the batch — they surface as failed entries in the response.

Field	Type	Required	Constraints
`vehicles`	array	yes	1–100 entries. Each entry is a `CreateVehicleRequest`.

Response

`207 Multi-Status`. Always 207, regardless of whether all rows succeeded, all failed, or any mix in between.

```
{
  "total": 3,
  "succeeded": 2,
  "failed": 1,
  "results": [
    {
      "row_index": 0,
      "status": "succeeded",
      "vehicle": { "vehicle_id": "vhc_a1b2c3...", "...": "full VehicleDetail" },
      "error_code": null,
      "error_message": null
    },
    {
      "row_index": 1,
      "status": "failed",
      "vehicle": null,
      "error_code": "motornet_code_not_in_catalogue",
      "error_message": "`motornet_code=BOGUS` is not available in the DealerMAX auto/VCOM vehicle catalogue. Verify that th
    },
    {
      "row_index": 2,
      "status": "succeeded",
      "vehicle": { "vehicle_id": "vhc_d4e5f6...", "...": "full VehicleDetail" },
      "error_code": null,
      "error_message": null
    }
  ]
}
```


Each `results[*].row_index` is the **zero-based index of the row in the request `vehicles[]` array**. The order is preserved: `results[N]` is the outcome of `vehicles[N]`. The partner can correlate a failure back to its own batch through this index alone.

`results[*].error_code` for failed rows uses the **same vocabulary as the single-POST surface**: `motornet_code_not_in_catalogue`, `vehicle_plate_already_registered`, `validation_error`. The partner can route every failure through a single error-handling function.

Transaction semantics

Each row is processed inside its own SQL `SAVEPOINT`. A failure on row N rolls back **only that row's** writes; the loop continues with row N+1. Successful rows accumulate in the outer transaction and are committed together at the end of the request. Failed rows leave no trace.

In practice:

- A batch of 100 rows where row 50 has an invalid `motornet_code` → 99 vehicles created, row 50 reported as failed, partner retries only that row.
- A batch where two rows share the same plate → first row succeeds (its INSERT lands first), second hits the UNIQUE constraint → 1 success + 1 `vehicle_plate_already_registered`.
- A batch where every row is malformed → 100 failures + zero writes + `db.commit()` is a no-op.

Error responses (whole-batch failures)

The 207 envelope handles all per-row failures. The following error codes apply to the call as a whole:

Status	`code`	Cause
`401`	`missing_api_key` / `invalid_api_key` / `expired_api_key`	Auth checks.
`402`	`subscription_inactive` / `grace_period_expired`	DealerMAX subscription not active.
`403`	`forbidden_dealer_not_owned`	`dealer_id` is not registered for this partner key.
`404`	`dealer_not_found`	No dealer with this ID under the calling partner.
`409`	`idempotency_key_reuse`	Idempotency key reused with a different batch body.
`422`	`validation_error`	The `vehicles[]` array fails validation as a whole (empty, > 100 rows, or one of the rows has a Pydantic-level error).

Example — Python SDK (recommended)

```
batch = [
    {"motornet_code": "VW0035478", "plate": "FA123XY", "registration_year": 2022,
     "certified_km": 28500, "sale_price_eur": 18900.0},
    {"motornet_code": "FT0123456", "plate": "GG456WW", "registration_year": 2023,
     "certified_km": 15000, "sale_price_eur": 16400.0},
    # ... up to 100 rows
]

response = client.dealers.vehicles.bulk(
    dealer_id="verdicar-001",
    vehicles=batch,
    idempotency_key=f"morning-sync-{date.today().isoformat()}",
)

print(f"{response.succeeded}/{response.total} provisioned")
for row in response.results:
    if row.status == "failed":
        print(f"  row {row.row_index} ({batch[row.row_index]['plate']}): "
              f"{row.error_code} — {row.error_message}")
```

Chunking pattern for very large imports

For initial migrations (say, 5 000 vehicles), chunk client-side into multiple calls of 100 rows each. The partnermax surface tolerates the parallelism, but the practical sweet spot is 4–6 concurrent calls — the bottleneck is the AI worker that processes each new vehicle within ~60 seconds of insertion.

```python
import asyncio
from itertools import islice

def chunks(iterable, size):
    it = iter(iterable)
    while batch := list(islice(it, size)):
        yield batch

async def migrate_all(client, dealer_id, all_vehicles):
    semaphore = asyncio.Semaphore(4)   # concurrency cap
    async def _send_chunk(idx, chunk):
        async with semaphore:
            return await client.dealers.vehicles.bulk(
                dealer_id=dealer_id,
                vehicles=chunk,
                idempotency_key=f"initial-migration-chunk-{idx:04d}",
            )

    tasks = [
        _send_chunk(idx, chunk)
        for idx, chunk in enumerate(chunks(all_vehicles, 100))
    ]
    return await asyncio.gather(*tasks)
```

With 4 concurrent workers each handling 100 rows per ~20-second call, the throughput is ~1200 vehicles per minute — a 5 000-vehicle migration completes in roughly four minutes wall-clock. The AI-content worker then catches up over the following 30 minutes (3 vehicles per minute per partner), and the cross-network indexing surfaces are warm within an hour.

`GET /v1/dealers/{dealer_id}/vehicles`

Paginated listing of vehicles in a dealer's stock.

Query parameters

Name	Type	Default	Description
`is_visible`	bool	—	Filter on the visibility flag.
`enabled_channel`	string	—	Filter on a publication channel. Allowed values: `rewind`, `nos`.
`include_deleted`	bool	`false`	Set `true` to also include soft-deleted rows.
`cursor`	string	—	Opaque base64url cursor from a previous response's `next_cursor`. The cursor is partner-scoped — passing a vehicle UUID from another partner's dealer returns `400 invalid_cursor`.
`limit`	integer	`50`	`[1, 100]`.

Default sort is `data_inserimento ASC`. Soft-deleted rows are hidden by default — pass `include_deleted=true` to surface them.

Response


```
{
  "data": [
    {
      "vehicle_id": "vhc_12345678-1234-5678-9abc-123456789abc",
      "motornet_code": "VW0035478",
      "plate": "FA123XY",
      "brand": "Volkswagen",
      "model": "T-Cross",
      "trim": "T-Cross 1.0 tsi Edition Plus 115cv dsg",
      "trim_alias": "Edition Plus",
      "fuel_type": "Benzina",
      "registration_year": 2022,
      "registration_month": 6,
      "certified_km": 28500,
      "enabled_channels": ["rewind"],
      "sale_price_eur": 18900.0,
      "cost_price_eur": 17000.0,
      "color": "Bianco perla",
      "vat_displayed": false,
      "is_visible": true,
      "vehicle_damaged": null,
      "damage_repaired": null,
      "cover_image_url": "https://xxx.supabase.co/storage/v1/object/public/auto-usate/12345678.../front-quarter_20260518...
      "image_count": 2,
      "ai_tagline": "T-Cross 2022 compatta, automatica e ben documentata",
      "ai_short": "SUV compatto con 28.500 km certificati, cambio DSG e tagliando recente.",
      "dealer_id": "verdicar-001",
      "created_at": "2026-05-18T10:00:00Z",
      "last_modified_at": "2026-05-18T10:00:30Z",
      "deleted_at": null
    }
  ],
  "has_more": false,
  "next_cursor": null
}
```


The listing response is intentionally card/sync oriented. It includes the cover photo, image count, AI tagline/short preview, `last_modified_at`, and `deleted_at` so partners can render stock cards and reconcile changes without fetching every detail page. Full image galleries, long AI content, service history, accessories, and the full Motornet technical sheet stay on `GET /vehicles/{vehicle_id}`.

Example — Python SDK (recommended)

```python
for vehicle in client.dealers.vehicles.list(dealer_id="verdicar-001", enabled_channel="rewind").data:
    print(vehicle.plate, vehicle.brand, vehicle.model, vehicle.sale_price_eur)
```

`GET /v1/dealers/{dealer_id}/vehicles/{vehicle_id}`

Single-vehicle detail with the full Motornet technical sheet.

Query parameters

Name	Type	Default	Description
`include_deleted`	bool	`false`	When `false`, a soft-deleted vehicle returns `404 vehicle_not_found`.

Response

`VehicleDetail` — see schema below.

`PATCH /v1/dealers/{dealer_id}/vehicles/{vehicle_id}`

Partial update of a vehicle. Sends only the fields that change.

Mutability rules

- **Immutable after creation**: `motornet_code`, `plate`. To correct either one, soft-delete and recreate.
- **Mutable**: every other field on the create body, including `vin`.
- **Nullable vs non-nullable**: explicit `null` clears nullable fields (for example `{"notes": null}`). `sale_price_eur` can be set to `null` only when the resulting `enabled_channels` is NOS-only; if the vehicle remains REWIND, the API returns `422 validation_error`.
- **Warranty coherence is enforced cross-request**: if the patch would leave the row with `extended_warranty_enabled=true` and `extended_warranty_months=null` (combining patch + existing state), the call returns `422 validation_error`.

Example — Python SDK (recommended)

```python
# Lower the asking price and hide the row from AI surfaces while you sort
# out the photos — reversible later.
client.dealers.vehicles.update(
    dealer_id="verdicar-001",
    vehicle_id="vhc_12345678-1234-5678-9abc-123456789abc",
    sale_price_eur=17_500.0,
    is_visible=False,
)

# Clear a nullable field by sending explicit null:
client.dealers.vehicles.update(
    dealer_id="verdicar-001",
    vehicle_id="vhc_12345678-1234-5678-9abc-123456789abc",
    notes=None,
)
```

`DELETE /v1/dealers/{dealer_id}/vehicles/{vehicle_id}`

Soft-delete a vehicle from stock. The vehicle disappears from MCP / Custom GPT / NLWeb / JSON-LD within the next index cycle, and the plate becomes reusable on the network the moment this returns.

Distinction from hiding

`PATCH` with `{"is_visible": false}` and `DELETE` are **not the same**:

- `PATCH is_visible=false` is reversible — the partner can flip it back to `true` later. Used for inventory hold without deleting the row.
- `DELETE` is terminal — the row is soft-deleted (audit history preserved, but `PATCH` afterward returns `409 dealer_deleted`-equivalent for vehicles).

Response

`204 No Content`.

Error responses

Status	`code`	Cause
`404`	`vehicle_not_found`	Unknown vehicle for this dealer.
`409`	`vehicle_already_deleted`	The vehicle is already soft-deleted.

`VehicleDetail` response

Returned by `POST`, `GET /vehicles/{id}`, and `PATCH`. Carries every partner-supplied field PLUS the derived Motornet specs.

```
{
  "vehicle_id": "vhc_12345678-1234-5678-9abc-123456789abc",
  "partner_id": "ptn_7",
  "dealer_id": "verdicar-001",
  "motornet_code": "VW0035478",
  "plate": "FA123XY",
  "vin": "WVWZZZ1KZJW123456",
  "brand": "Volkswagen",
  "model": "T-Cross",
  "trim": "T-Cross 1.0 tsi Edition Plus 115cv dsg",
  "fuel_type": "Benzina",
  "registration_year": 2022,
  "registration_month": 6,
  "certified_km": 28500,
  "color": "Bianco perla",
  "base_color": "Bianco",
  "alloy_wheel_size": 17,
  "service_history_available": true,
  "last_service_date": "2025-12-01",
  "last_service_km": 24000,
  "last_service_notes": "Tagliando certificato presso rete ufficiale.",
  "inspection_due_date": "2027-06-30",
  "last_inspection_date": "2025-06-30",
  "last_inspection_km": 22000,
  "property_tax_due_date": "2026-12-31",
  "ownership_transfer_date": "2025-06-15",
  "double_keys_available": true,
  "vehicle_damaged": null,
  "damage_repaired": null,
  "previous_owner_count": 1,
  "fuel_type_override": null,
  "power_kw_override": null,
  "wltp_consumption_combined_l_100km": 5.7,
  "wltp_consumption_urban_l_100km": null,
  "wltp_consumption_extraurban_l_100km": null,
  "co2_emissions_g_km_override": 126,
  "extended_warranty_enabled": true,
  "extended_warranty_months": 24,
  "enabled_channels": ["rewind"],
  "sale_price_eur": 18900.0,
  "cost_price_eur": 17000.0,
  "vat_displayed": false,
  "trim_alias": "Edition Plus",
  "is_visible": true,
  "description": "Tagliando appena fatto. Garanzia 24 mesi inclusa.",
  "notes": "Trasferimento di proprietà di un anno fa.",
  "last_modified_at": "2026-05-18T10:00:30Z",
  "created_at": "2026-05-18T10:00:00Z",
  "technical_details": {
    "cilindrata": 999,
    "kw": 85,
    "hp": 115,
    "cavalli_fiscali": 14,
    "alimentazione": "Benzina",
    "tipo_cambio": "Automatico DSG 7 rapporti",
    "trazione": "Anteriore",
    "lunghezza": 4108,
    "larghezza": 1760,
    "altezza": 1573,
    "passo": 2563,
    "peso": 1300,
    "bagagliaio": "385",
    "porte": 5,
    "posti": 5,
    "velocita": 187,
    "accelerazione": 10.2,
    "consumo_medio": 5.7,
    "emissioni_co2": "126",
```

```
    "euro": "Euro 6d"
  },
  "ai_content": {
    "tagline": "T-Cross 2022 perfetta per chi cerca SUV compatto ben tenuto",
    "short": "Volkswagen T-Cross 1.0 TSI Edition Plus 2022, 28.500 km, primo proprietario, garanzia 24 mesi inclusa.",
    "medium": "T-Cross 1.0 TSI DSG Edition Plus del 2022 con soli 28.500 km certificati. Tagliando appena eseguito, primo e
    "long": "<p>La Volkswagen T-Cross 1.0 TSI 115 CV Edition Plus DSG del 2022 rappresenta il punto di equilibrio ideale pe
    "highlights": [
      "Primo e unico proprietario",
      "28.500 km certificati con tagliando recente",
      "Garanzia estesa 24 mesi inclusa",
      "Cambio automatico DSG 7 marce",
      "Sensori parcheggio anteriori e posteriori"
    ],
    "faq": [
      {"question": "La garanzia copre tutti i componenti?", "answer": "La garanzia 24 mesi inclusa nel prezzo copre motore,
      {"question": "È possibile finanziare l'acquisto?", "answer": "Sì, attraverso i nostri partner finanziari proponiamo f
      {"question": "Qual è il consumo dichiarato?", "answer": "Il consumo medio dichiarato è di 5.7 L/100km secondo il cicl
    ],
    "seo_title": "Volkswagen T-Cross 1.0 TSI 2022 Edition Plus DSG 28.500 km",
    "seo_description": "T-Cross 1.0 TSI 2022 Edition Plus DSG 115 CV, 28.500 km certificati, primo proprietario, garanzia 2
    "slug": "volkswagen-t-cross-1-0-tsi-edition-plus-2022-12345678",
    "generated_at": "2026-05-18T10:01:30Z"
  },
  "image_urls": [
    "https://xxx.supabase.co/storage/v1/object/public/auto-usate/12345678.../front-quarter_20260518....jpg",
    "https://xxx.supabase.co/storage/v1/object/public/auto-usate/12345678.../interior_20260518....jpg"
  ]
}
```


`technical_details` is a flat dict where keys use Italian Motornet domain vocabulary (units in SI: cc / mm / kg / km/h / g/km / sec / litres) — same convention as the NLT detail surface so SDK consumers can write a single rendering helper.

`ai_content` is the editorial output produced asynchronously by the partnermax AI pipeline (gpt-5, ~60 seconds after vehicle creation). The block is `null` for a freshly-inserted vehicle that the worker hasn't processed yet; once populated, every field is the SAME content the consumer AI surfaces (MCP `search_vehicles` response card, ChatGPT Custom GPT, NLWeb `/ask`) display. Branch on `ai_content is None` to render a "generating…" state, or backfill with `notes` / `description` until the worker completes.

Field	Type	When populated
`ai_content.tagline`	string | null	8–12 word headline. Use on listing cards.
`ai_content.short`	string | null	≤220 chars summary. Use as meta-description fallback.
`ai_content.medium`	string | null	~400 char paragraph. Use on detail-page SEO blurbs.
`ai_content.long`	string | null	1500–3000 char marketing description (may contain HTML). Use on detail pages.
`ai_content.highlights`	array of strings | null	3–7 selling points. Render as a bullet list above the description.
`ai_content.faq`	array of `{question, answer}` | null	3–6 Q&A pairs. Render as accordion or feed to a JSON-LD `FAQPage`.
`ai_content.seo_title`	string | null	≤60 chars, `<title>`-ready.
`ai_content.seo_description`	string | null	≤160 chars, meta-description-ready.
`ai_content.slug`	string | null	URL-safe slug used in the canonical URL on the dealer site.
`ai_content.generated_at`	datetime | null	UTC timestamp of the most recent generation.

`image_urls` is populated by `POST /v1/dealers/{dealer_id}/vehicles/{vehicle_id}/images` (see the **Vehicle images** section below), in the order they were uploaded. The first entry is the cover.

Vehicle accessories catalog

Optionals, packages and equipment are not free-text fields in the vehicle create body. They are catalog-backed selections:

```
GET  /v1/dealers/{dealer_id}/vehicles/{vehicle_id}/accessories/catalog
POST /v1/dealers/{dealer_id}/vehicles/{vehicle_id}/accessories/catalog/refresh
PUT  /v1/dealers/{dealer_id}/vehicles/{vehicle_id}/accessories
```

Use `GET .../accessories/catalog` to read the current per-vehicle catalog and selection state. If the vehicle has just been created or the partner needs the latest Motornet optionals/packages, call `POST .../accessories/catalog/refresh`; the API refreshes the per-vehicle Motornet catalog and returns the same response shape.

`PUT .../accessories` replaces the partner-managed selections. Send only IDs returned by the catalog endpoint.

Catalog response


```
{
  "series": [
    {
      "id": "ser_123",
      "code": "S001",
      "description": "Climatizzatore automatico",
      "group": null,
      "category": "Comfort",
      "price_eur": null,
      "selected": true
    }
  ],
  "optionals": [
    {
      "id": "opt_456",
      "code": "OPT01",
      "description": "Vernice metallizzata",
      "group": "Esterni",
      "category": null,
      "price_eur": 650.0,
      "selected": false
    }
  ],
  "packages": [],
  "equipment": [
    {
      "id": "15",
      "code": null,
      "description": "Cerchi in lega",
      "group": null,
      "category": null,
      "price_eur": null,
      "selected": true
    }
  ],
  "alloy_wheel_size": 17
}
```

Set selections

```
{
  "optional_ids": ["opt_456"],
  "package_ids": [],
  "equipment_ids": ["15"],
  "alloy_wheel_size": 17
}
```

Selections are full replacement arrays. Omit an ID to clear it. `alloy_wheel_size` is kept only when the equipment selection includes alloy wheels; otherwise it is cleared.

Condition state is intentionally limited to the first DealerMax condition step in this SDK cycle: `vehicle_damaged` and `damage_repaired`. Repair economics, internal restoration estimates and workflow markers stay out of the partner contract.

Vehicle images (v1.2)

Three endpoints under the per-vehicle namespace let a partner attach, list, and remove photos. The photos appear on every consumer AI surface (MCP `search_vehicles`, ChatGPT Custom GPT card, NLWeb `/ask` result, the dealer site SEO + JSON-LD `Car` schema) within the next apimax cache TTL (≤ 5 min) of the upload.

```
POST   /v1/dealers/{dealer_id}/vehicles/{vehicle_id}/images
GET    /v1/dealers/{dealer_id}/vehicles/{vehicle_id}/images
DELETE /v1/dealers/{dealer_id}/vehicles/{vehicle_id}/images/{image_id}
```

Contract: order = position, position 1 = cover

The partner POSTs photos one at a time, in the desired display order. The first POST becomes `position=1` (`is_cover=true`) automatically; subsequent POSTs increment `position`. There is **no separate "set cover" endpoint** — order is the contract. To re-order, DELETE and re-POST.

`DELETE` re-ranks the remaining photos to a contiguous `1..N` sequence so the cover (`position=1`) is always present when at least one photo remains.

Limits

Limit	Value	On breach
Photos per vehicle	20	409 `vehicle_image_limit_reached`
Per-photo size	15 MB	413 `image_too_large`
MIME types	`image/jpeg`, `image/png`, `image/webp`	415 `image_format_unsupported`

WebP is converted to PNG server-side. The returned `image_url` always points at the raw upload; the cross-network AI surfaces serve an AVIF-transformed variant via Supabase's render service.

`POST /v1/dealers/{dealer_id}/vehicles/{vehicle_id}/images`

Multipart/form-data body, single field `file`. Returns the created `VehicleImage`.

```python
from partnermax import Partnermax

client = Partnermax()  # reads PARTNERMAX_API_KEY

with open("front-quarter.jpg", "rb") as fh:
    cover = client.dealers.vehicles.images.create(
        "vhc_12345678-1234-5678-9abc-123456789abc",
        dealer_id="verdicar-001",
        file=fh,
    )
print(cover.position, cover.is_cover, cover.image_url)
# 1 True https://.../auto-usate/12345678.../front-quarter_20260518....jpg
```

Node / TypeScript:

```typescript
import fs from "node:fs";
import Partnermax, { toFile } from "partnermax";

const client = new Partnermax(); // reads PARTNERMAX_API_KEY
const file = await toFile(
  fs.createReadStream("front-quarter.jpg"),
  "front-quarter.jpg",
  { type: "image/jpeg" },
);

const cover = await client.dealers.vehicles.images.create(
  "vhc_12345678-1234-5678-9abc-123456789abc",
  {
    dealer_id: "verdicar-001",
    file,
  },
);
console.log(cover.position, cover.is_cover, cover.image_url);
```

Or with `curl`:

```bash
curl -X POST \
  "https://api.dealermax.app/v1/dealers/verdicar-001/vehicles/vhc_12345678-1234-5678-9abc-123456789abc/images" \
  -H "X-Api-Key: $PARTNERMAX_API_KEY" \
  -F "file=@front-quarter.jpg;type=image/jpeg"
```

Response (`201 Created`):

```json
{
  "image_id": "img_a3f9b2c1-1234-4567-8abc-def012345678",
  "image_url": "https://xxx.supabase.co/storage/v1/object/public/auto-usate/12345678.../front-quarter_20260518....jpg",
  "position": 1,
  "is_cover": true,
  "created_at": "2026-05-18T13:45:22.123456+00:00"
}
```

`GET /v1/dealers/{dealer_id}/vehicles/{vehicle_id}/images`

Returns every photo for the vehicle, sorted by `position` ascending. No pagination — the 20-per-vehicle cap means the full list always fits in one response.

There is no single-image retrieve or update route in v1. Retrieve image metadata through this list endpoint; replace a file or change ordering by deleting the affected image and posting it again in the desired order.

```python
photos = client.dealers.vehicles.images.list(
    dealer_id="verdicar-001",
    vehicle_id="vhc_12345678-1234-5678-9abc-123456789abc",
)
for p in photos.data:
    print(p.position, p.is_cover, p.image_url)
```

`DELETE /v1/dealers/{dealer_id}/vehicles/{vehicle_id}/images/{image_id}`

Removes a photo and renumbers the remaining photos to keep `position` contiguous. Returns `204 No Content`. If the cover was deleted, the next photo (former `position=2`) is promoted to cover automatically.

```python
client.dealers.vehicles.images.delete(
    dealer_id="verdicar-001",
    vehicle_id="vhc_12345678-1234-5678-9abc-123456789abc",
    image_id="img_a3f9b2c1-1234-4567-8abc-def012345678",
)
```

Error responses

Status	Code	Cause
404	`vehicle_image_not_found`	Unknown `image_id`, or `image_id` belongs to a different vehicle
409	`vehicle_image_limit_reached`	Vehicle already has 20 photos
413	`image_too_large`	Payload > 15 MB
415	`image_format_unsupported`	MIME type not in the allow-list
502	`image_upload_failed`	Transient Supabase Storage error — retry with backoff
503	`storage_not_configured`	The environment is not wired to Supabase Storage (production has it enabled)

CHAPTER 11

Webhooks - Future Contract / Roadmap

> ***NOT AVAILABLE IN v1.*** *PartnerMAX does not deliver outbound webhook events to partner-controlled URLs in the current product. Do not build production dependencies on this chapter; use polling against the read endpoints in v1. Daily polling is the expected integration pattern for NLT offer refreshes.*

> *All examples and endpoint shapes below are future-contract design notes retained for planning only.*

The `/v1/webhooks/*` management endpoints described on this page are **planned future design notes**. They are not exposed in the public OpenAPI file, Python SDK, TypeScript SDK, or GA PartnerMAX `/v1` surface.

Overview

PartnerMAX v1 is polling-first. Partners should schedule reads such as `GET /v1/dealers`, `GET /v1/dealers/{dealer_id}/nlt/offers`, and `GET /v1/dealers/{dealer_id}/vehicles` according to their own sync cadence. For NLT offer updates, daily polling is sufficient for the supported v1 use case.

The material below is retained only as a roadmap design reference for partners who need to plan future event-driven integrations. Do not build production dependencies on it.

This document covers:

- The delivery model (transport, ordering, idempotency).
- Signature construction and verification, with reference implementations.
- Replay protection.
- The full event catalog with payload schemas.
- Retry and dead-letter behavior.
- Endpoint registration and management.
- Local development with tunneling.
- Security best practices for production handlers.

Delivery model

Transport

partnermax delivers each event as an HTTP `POST` request to the URL you have registered for that event type. The body is a single JSON object. The `Content-Type` is always `application/json; charset=utf-8`. TLS 1.2 minimum is required on your endpoint; partnermax negotiates TLS 1.3 when supported. Plain HTTP endpoints are rejected at registration time.

partnermax originates requests from a fixed list of IP addresses published at `https://developers.dealermax.app/.well-known/webhook-source-ips.json`. You may use this list to allow-list inbound traffic at your firewall, but you should not treat IP allow-listing as a substitute for signature verification — the source list may expand without notice when partnermax scales horizontally, and IP spoofing is not a defense against a forged payload that happens to originate from a trusted IP.

Ordering

partnermax does not guarantee ordering. Two events about the same dealer may arrive at your endpoint out of the order in which they occurred. Each event payload carries an `occurred_at` ISO-8601 timestamp (UTC, millisecond precision) and a monotonically increasing `sequence` integer scoped to the partner account. Consumers that depend on ordering — for example, applying `dealer.updated` events in the order they were emitted — should sort by `sequence` server-side and discard events older than the most recently applied sequence for that resource.

Delivery semantics

partnermax guarantees at-least-once delivery. Network partitions, your endpoint returning a transient error, or partnermax retrying a request that did succeed but whose response was lost may cause the same event to be delivered more than once. Every event carries an `id` field that is globally unique and stable across redeliveries; your handler must be idempotent with respect to this identifier. See Idempotent handler design below.

Response expectations

Your endpoint should respond with any 2xx status code within 30 seconds. partnermax treats:

- **2xx** as success. The event is marked delivered.
- **4xx (except 408, 429)** as a permanent failure. The event is moved to the dead-letter queue without retries. This is intended for cases where your application has decided the event is malformed or unwanted; do not respond 4xx for transient bugs.
- **408, 429, 5xx, network errors, and timeouts** as transient failures. The event is scheduled for retry per the retry policy.

If your handler does work that takes longer than 30 seconds, you should enqueue the event into your own job queue and respond 200 immediately. Holding the connection open while you process the payload will cause partnermax to time out and retry, leading to duplicate processing.

Signing and verification

Every webhook request includes two headers used for authenticity and replay protection:

Header	Description
`Partnermax-Timestamp`	Unix timestamp in seconds when the request was constructed by partnermax.
`Partnermax-Signature`	Comma-separated list of `key=value` pairs. Currently `t=<unix-ts>,v1=<hex>`.

The signature scheme is intentionally identical to Stripe's, both because it is well understood by security engineers and because most internal review processes have already vetted it.

Constructing the signed payload

The string that partnermax signs is the concatenation of three values, joined by a literal period (`.`):

```
signed_payload = <timestamp> + "." + <raw_request_body>
```

- `<timestamp>` is the value of the `Partnermax-Timestamp` header, as a decimal string.

- `<raw_request_body>` is the exact bytes of the request body, before any framework parsing. This matters: if your web framework re-serializes JSON before handing it to your handler, key order and whitespace will differ and the signature will not match. Capture the raw body and verify before parsing.

The HMAC is computed with SHA-256, keyed by your webhook signing secret, and hex-encoded:

```
v1 = hex( HMAC_SHA256( webhook_secret, signed_payload ) )
```

The resulting hex string is the value of the `v1=` field in the `Partnermax-Signature` header.

Reference implementation — Python

```python
import hmac
import hashlib
import time
from typing import Mapping


class WebhookVerificationError(Exception):
    pass


def verify_webhook(
    raw_body: bytes,
    headers: Mapping[str, str],
    secret: str,
    tolerance_seconds: int = 300,
) -> None:
    """
    Verify a partnermax webhook signature.

    Raises WebhookVerificationError on any failure. On success returns None.

    Pass the RAW request body bytes — not a parsed/re-serialized dict.
    """
    timestamp_header = headers.get("Partnermax-Timestamp")
    signature_header = headers.get("Partnermax-Signature")

    if not timestamp_header or not signature_header:
        raise WebhookVerificationError("Missing required webhook headers")

    try:
        timestamp = int(timestamp_header)
    except ValueError as exc:
        raise WebhookVerificationError("Invalid timestamp header") from exc

    now = int(time.time())
    if abs(now - timestamp) > tolerance_seconds:
        raise WebhookVerificationError(
            f"Timestamp outside tolerance window of {tolerance_seconds}s"
        )

    # Parse signature header: "t=1735000000,v1=abc123..."
    parts = dict(
        pair.split("=", 1) for pair in signature_header.split(",") if "=" in pair
    )
    received_signature = parts.get("v1")
    if not received_signature:
        raise WebhookVerificationError("No v1 signature found in header")

    signed_payload = f"{timestamp}.".encode("utf-8") + raw_body
    expected_signature = hmac.new(
        secret.encode("utf-8"),
        signed_payload,
        hashlib.sha256,
    ).hexdigest()

    # Constant-time comparison — never use == here
    if not hmac.compare_digest(expected_signature, received_signature):
        raise WebhookVerificationError("Signature mismatch")
```

Integrating with Flask:

```python
from flask import Flask, request, abort
import json

app = Flask(__name__)
WEBHOOK_SECRET = "whsec_..."  # load from your secret manager

@app.route("/webhooks/partnermax", methods=["POST"])
def handle_webhook():
    raw = request.get_data()  # raw bytes, BEFORE Flask parses JSON
    try:
        verify_webhook(raw, request.headers, WEBHOOK_SECRET)
    except WebhookVerificationError:
        abort(401)

    event = json.loads(raw)
    # Enqueue for async processing and respond immediately
    job_queue.enqueue("process_partnermax_event", event)
    return "", 200
```

Reference implementation — Node.js

```
const crypto = require("crypto");

class WebhookVerificationError extends Error {}

function verifyWebhook(rawBody, headers, secret, toleranceSeconds = 300) {
  const timestampHeader = headers["partnermax-timestamp"];
  const signatureHeader = headers["partnermax-signature"];

  if (!timestampHeader || !signatureHeader) {
    throw new WebhookVerificationError("Missing required webhook headers");
  }

  const timestamp = parseInt(timestampHeader, 10);
  if (Number.isNaN(timestamp)) {
    throw new WebhookVerificationError("Invalid timestamp header");
  }

  const now = Math.floor(Date.now() / 1000);
  if (Math.abs(now - timestamp) > toleranceSeconds) {
    throw new WebhookVerificationError(
      `Timestamp outside tolerance window of ${toleranceSeconds}s`
    );
  }

  const parts = Object.fromEntries(
    signatureHeader.split(",").map((p) => p.split("=", 2))
  );
  const receivedSignature = parts.v1;
  if (!receivedSignature) {
    throw new WebhookVerificationError("No v1 signature in header");
  }

  const signedPayload = Buffer.concat([
    Buffer.from(`${timestamp}.`, "utf-8"),
    rawBody,
  ]);

  const expectedSignature = crypto
    .createHmac("sha256", secret)
    .update(signedPayload)
    .digest("hex");

  const expectedBuf = Buffer.from(expectedSignature, "utf-8");
  const receivedBuf = Buffer.from(receivedSignature, "utf-8");

  if (
    expectedBuf.length !== receivedBuf.length ||
    !crypto.timingSafeEqual(expectedBuf, receivedBuf)
  ) {
    throw new WebhookVerificationError("Signature mismatch");
  }
}

module.exports = { verifyWebhook, WebhookVerificationError };
```

Integrating with Express. Note the explicit `express.raw()` middleware on this route — the default `express.json()` middleware will discard the raw bytes and signature verification will fail:

```
const express = require("express");
const { verifyWebhook, WebhookVerificationError } = require("./verify");

const app = express();
const WEBHOOK_SECRET = process.env.PARTNERMAX_WEBHOOK_SECRET;

app.post(
  "/webhooks/partnermax",
  express.raw({ type: "application/json" }),
  (req, res) => {
    try {
      verifyWebhook(req.body, req.headers, WEBHOOK_SECRET);
    } catch (err) {
      if (err instanceof WebhookVerificationError) {
        return res.status(401).send("invalid signature");
      }
      throw err;
    }

    const event = JSON.parse(req.body.toString("utf-8"));
    jobQueue.enqueue("process_partnermax_event", event);
    res.status(200).send();
  }
);
```

Common verification mistakes

- **Parsing then re-serializing the body.** As noted above, the signature is computed over the exact bytes of the wire payload. Re-encoding it almost always reorders keys or rewrites numeric precision and breaks the HMAC.
- **String equality on the signature.** Using `==` opens a timing side channel. Always use `hmac.compare_digest`, `crypto.timingSafeEqual`, or your language's equivalent.
- **Using the wrong secret.** Every endpoint has its own secret. Rotating an endpoint's secret invalidates the previous one immediately for that endpoint only.
- **Disabling TLS verification on the partnermax side.** This is not configurable; partnermax always validates your endpoint's TLS chain.

Replay protection

The `Partnermax-Timestamp` header is included in the signed payload, so it cannot be modified without invalidating the signature. Your verification code must enforce a tolerance window — the default and recommended value is 5 minutes (300 seconds).

Without a tolerance check, an attacker who once intercepts a valid signed request can replay it indefinitely. Five minutes is large enough to absorb normal clock skew between partnermax and your servers, small enough that captured payloads expire quickly. If your infrastructure has unusual NTP drift, you may increase the tolerance, but values above 15 minutes are strongly discouraged.

Replay protection is separate from at-least-once delivery deduplication. The tolerance window protects against malicious replays of old payloads; the `id` field on every event protects against legitimate redeliveries from partnermax's retry queue. Your handler should enforce both.

Event types catalog

Every event has the following envelope:

```
{
  "id": "evt_1001",
  "type": "dealer.created",
  "occurred_at": "2026-08-15T14:23:47.831Z",
  "sequence": 4271,
  "partner_id": "ptn_7",
  "livemode": true,
  "data": { /* event-specific payload */ }
}
```

Field reference:

Field	Type	Description
`id`	string	ULID. Globally unique, stable across redeliveries. Use this for idempotent processing.
`type`	string	One of the event types below. New event types may be added over time without a breaking version bump.
`occurred_at`	ISO-8601 timestamp	UTC, millisecond precision. The time the event was emitted by partnermax, not the time it was delivered.
`sequence`	integer	Monotonically increasing per `partner_id`. Use for ordering.
`partner_id`	string	The partner account this event belongs to.
`livemode`	boolean	`true` for production, `false` for sandbox. Compare to your environment to prevent cross-environment processing.
`data`	object	Event-specific payload. Documented per event type below.

`dealer.created`

Roadmap event emitted when a new opaque partner dealer reference is created through `POST /api/partner/dealers` on `api.dealermax.app`.

```
{
  "id": "evt_1001",
  "type": "dealer.created",
  "occurred_at": "2026-08-15T14:23:47.831Z",
  "sequence": 4271,
  "partner_id": "ptn_7",
  "livemode": true,
  "data": {
    "dealer": {
      "id": "verdicar-001",
      "external_id": "your-internal-dealer-id",
      "legal_name": "Auto Rossi S.r.l.",
      "brand_name": "Auto Rossi",
      "vat_number": "IT01234567890",
      "country": "IT",
      "city": "Milano",
      "province": "MI",
      "status": "active",
      "created_at": "2026-08-15T14:23:47.512Z",
      "mcp_indexed": false,
      "nlt_settings_configured": false
    }
  }
}
```

`mcp_indexed` will be `false` immediately after creation. It transitions to `true` within minutes once the dealer is indexed by the [MCP server at `mcp.dealermax.app`] and downstream AI surfaces. A subsequent `dealer.updated` event is not emitted for this transition; if you need to know when a dealer is queryable from AI surfaces, poll the dealer resource.

`dealer.updated`

Emitted when any persisted attribute of a dealer changes — legal name, address, contact information, business metadata. Configuration of NLT-specific settings emits the more specific `dealer.nlt_settings.changed` event instead and does NOT emit a generic `dealer.updated`.


```
{
  "id": "evt_1002",
  "type": "dealer.updated",
  "occurred_at": "2026-08-15T15:10:22.408Z",
  "sequence": 4282,
  "partner_id": "ptn_7",
  "livemode": true,
  "data": {
    "dealer": {
      "id": "verdicar-001",
      "external_id": "your-internal-dealer-id",
      "legal_name": "Auto Rossi S.r.l.",
      "brand_name": "Auto Rossi Premium",
      "vat_number": "IT01234567890",
      "country": "IT",
      "city": "Milano",
      "province": "MI",
      "status": "active",
      "updated_at": "2026-08-15T15:10:22.183Z"
    },
    "changed_fields": ["brand_name"],
    "previous_values": {
      "brand_name": "Auto Rossi"
    }
  }
}
```


`changed_fields` lists every attribute whose value differs from the prior state. `previous_values` echoes the prior values of those fields only. Fields not listed in `changed_fields` are not included in `previous_values` even if you sent a payload that contained them.

`dealer.deactivated`

Emitted when a dealer transitions from `status: "active"` to `status: "inactive"`, either by explicit partner action (`DELETE /v1/dealers/{id}`) or by [cascade deactivation] when the partner subscription is cancelled and the grace period expires.

```
{
  "id": "evt_1003",
  "type": "dealer.deactivated",
  "occurred_at": "2026-08-15T16:45:11.221Z",
  "sequence": 4291,
  "partner_id": "ptn_7",
  "livemode": true,
  "data": {
    "dealer": {
      "id": "verdicar-001",
      "external_id": "your-internal-dealer-id",
      "status": "inactive",
      "deactivated_at": "2026-08-15T16:45:11.087Z"
    },
    "reason": "partner_subscription_cancelled",
    "reactivation_eligible": true
  }
}
```

`reason` is one of:

Value	Meaning
`partner_request`	Explicit `DELETE /v1/dealers/{id}`.
`partner_subscription_cancelled`	Cascade deactivation after grace period.
`policy_violation`	Administrative deactivation by DealerMAX. Includes a separate `policy_violation_reference` for support correspondence.

When `reactivation_eligible` is `true`, the partner dealer reference may be restored through the partner-registry activation endpoint once the underlying cause is resolved. When `false`, the dealer is in a terminal state and a fresh dealer reference must be created if needed.

`dealer.nlt_settings.changed`

Emitted when NLT economic configuration for a dealer changes — mark-up percentage, down-payment tiers, default offer overrides. See `PATCH /v1/dealers/{id}/nlt-settings`.

```
{
  "id": "evt_1004",
  "type": "dealer.nlt_settings.changed",
  "occurred_at": "2026-08-15T17:02:18.554Z",
  "sequence": 4304,
  "partner_id": "ptn_7",
  "livemode": true,
  "data": {
    "dealer_id": "verdicar-001",
    "nlt_settings": {
      "agency_markup_percent": 4.5,
      "down_payment_tiers": {
        "low": 0,
        "medium": 3000,
        "high": 6000
      },
      "currency": "EUR",
      "effective_at": "2026-08-15T17:02:18.512Z"
    },
    "previous_settings": {
      "agency_markup_percent": 3.0,
      "down_payment_tiers": {
        "low": 0,
        "medium": 2500,
        "high": 5000
      },
      "currency": "EUR"
    }
  }
}
```


Settings changes propagate to AI-surfaceable artifacts (JSON-LD on the dealer's microsite, MCP server quotation responses, NLWeb summaries) within the index-refresh window. The event fires when the database write commits, not when downstream surfaces have caught up. If you need to know when a price change is visible to consumer-facing AI surfaces, poll the dealer's `mcp_indexed_at` field on the dealer resource.

`partner.subscription.updated`

Emitted when the DealerMAX subscription backing your partner account changes state. See [subscription gating].


```
{
  "id": "evt_1005",
  "type": "partner.subscription.updated",
  "occurred_at": "2026-08-15T18:33:42.117Z",
  "sequence": 4322,
  "partner_id": "ptn_7",
  "livemode": true,
  "data": {
    "subscription": {
      "stripe_subscription_id": "sub_1NXxx...",
      "status": "active",
      "plan": "enterprise",
      "current_period_start": "2026-08-01T00:00:00.000Z",
      "current_period_end": "2026-09-01T00:00:00.000Z",
      "cancel_at_period_end": false,
      "dealer_seat_limit": 500,
      "dealer_seat_usage": 187
    },
    "previous_status": "past_due",
    "grace_period_active": false,
    "grace_period_expires_at": null
  }
}
```

Possible `subscription.status` values: `active`, `past_due`, `unpaid`, `canceled`, `incomplete`. When `status` is `past_due` or `unpaid`, `grace_period_active` will be `true` and `grace_period_expires_at` will be populated; API access continues during this window. Terminal states (`canceled`, `incomplete_expired`) revoke active partner keys and cascade-deactivate dealer references registered for the partner immediately.

`partner.subscription.cancelled`

Emitted once when a subscription transitions to `status: "canceled"` (terminal). This is a convenience event distinct from `partner.subscription.updated`; you may subscribe to one or both.

```
{
  "id": "evt_1006",
  "type": "partner.subscription.cancelled",
  "occurred_at": "2026-08-22T00:00:00.000Z",
  "sequence": 4498,
  "partner_id": "ptn_7",
  "livemode": true,
  "data": {
    "subscription": {
      "stripe_subscription_id": "sub_1NXxx...",
      "status": "canceled",
      "cancelled_at": "2026-08-22T00:00:00.000Z",
      "cancellation_reason": "customer_request"
    },
    "cascade_deactivated_at": "2026-08-22T00:00:05.000Z",
    "affected_dealer_count": 187
  }
}
```

`cascade_deactivated_at` is the time at which the partner's registered dealer references were hidden from public/AI surfaces. Reactivation after billing recovery is explicit per dealer.

`partner.key.issued`

Emitted when a new API key is created for your partner account during onboarding or a support-managed replacement.

```
{
  "id": "evt_1007",
  "type": "partner.key.issued",
  "occurred_at": "2026-08-15T19:14:08.732Z",
  "sequence": 4341,
  "partner_id": "ptn_7",
  "livemode": true,
  "data": {
    "key": {
      "id": "key_7",
      "prefix": "pmk_live_aBcD",
      "label": "production-backend-2026-08",
      "scope": "all_dealers",
      "created_by": "system:partner-bootstrap",
      "expires_at": null
    }
  }
}
```

The raw secret is never included in this event. Only the prefix (the first 12 visible characters of the key) is exposed, sufficient for cross-referencing with API audit logs. If a key was just generated, use the secure credential-delivery flow from DealerMAX support; do not rely on webhook delivery to recover the plaintext.

`partner.key.revoked`

Emitted when an API key is revoked during support-managed replacement, account deactivation, or a PartnerMAX security response to a suspected compromise.


```
{
  "id": "evt_1008",
  "type": "partner.key.revoked",
  "occurred_at": "2026-08-15T20:00:14.992Z",
  "sequence": 4358,
  "partner_id": "ptn_7",
  "livemode": true,
  "data": {
    "key": {
      "id": "key_7",
      "prefix": "pmk_live_aBcD",
      "label": "production-backend-2026-08",
      "revoked_at": "2026-08-15T20:00:14.811Z",
      "revoked_by": "system:partner-bootstrap",
      "revocation_reason": "operator_action"
    }
  }
}
```


`revocation_reason` values: `operator_action` (support/operator replacement or deactivation), `rotation_policy` (scheduled support-managed replacement), `security_response` (DealerMAX-initiated due to suspected leak), `subscription_terminated` (cascade with subscription cancellation).

A revocation takes effect immediately on the API edge; any in-flight requests using the revoked key complete, subsequent requests return 401. If you receive this event for a key you did not intend to revoke, contact support@dealermax.app immediately.

Delivery retry policy

A failed delivery — anything other than a 2xx response within 30 seconds, excluding 4xx-permanent codes — is retried with exponential backoff:

Attempt	Delay from previous attempt	Cumulative time from original event
1	(initial)	0
2	1 minute	1 minute
3	5 minutes	6 minutes
4	15 minutes	21 minutes
5	1 hour	1 hour 21 minutes
6	6 hours	7 hours 21 minutes

After the fifth retry (sixth total delivery attempt) fails, the event is moved to the dead-letter queue. partnermax does not silently drop the event — it is retained for 30 days, and you may request manual redelivery via `support@dealermax.app` once you have fixed the underlying issue with your endpoint.

If your endpoint sustains a high failure rate (>50% over a 10-minute window), partnermax will temporarily reduce the delivery rate to that endpoint to avoid amplifying an outage. Recovery is automatic once the failure rate falls below the threshold.

Manual replay

To request replay of one or more failed deliveries, email `support@dealermax.app` with the `event_id` (or list of `event_id`s) and the target endpoint. Replays use the original event payload (preserving the original `id` for idempotency) and a fresh timestamp and signature.

You may also request replay of successful deliveries — useful when developing a new handler that needs to reproduce a previously seen event. Replays always go to the currently registered URL, not to wherever the original delivery was sent.

Endpoint registration

Creating an endpoint

```
POST /v1/webhooks/endpoints HTTP/1.1
Host: api.dealermax.app
Authorization: Bearer pmk_live_...
Content-Type: application/json
Idempotency-Key: 0e3c6e85-9f12-4a48-b86f-1b8c2d6a9e54

{
  "url": "https://api.your-product.example/webhooks/partnermax",
  "event_types": [
    "dealer.created",
    "dealer.updated",
    "dealer.deactivated",
    "dealer.nlt_settings.changed"
  ],
  "label": "production-handler"
}
```

```
{
  "id": "we_19",
  "url": "https://api.your-product.example/webhooks/partnermax",
  "event_types": [
    "dealer.created",
    "dealer.updated",
    "dealer.deactivated",
    "dealer.nlt_settings.changed"
  ],
  "label": "production-handler",
  "status": "active",
  "secret": "whsec_kRm8nP3qL2vJ7xY4tB9zA1cF6dH5jN0e",
  "created_at": "2026-08-15T20:30:00.000Z"
}
```

The `secret` is returned in the response body only on creation. It is never retrievable later — store it in your secrets manager immediately. If lost, rotate via `POST /v1/webhooks/endpoints/{id}/rotate-secret`, which invalidates the old secret on success.

Constraints

- Maximum 3 endpoints per partner account. This is a deliberate limit to keep delivery accounting predictable; if you need to fan out to more destinations, do so on your side after receiving once.
- The endpoint URL must use HTTPS, must resolve via public DNS, and must respond to a probe request issued at registration time. Endpoints behind authentication that rejects the probe will be created in `status: "pending_verification"` and must pass a manual check before activation.
- Subscribing to event types you cannot yet handle is supported — they will accumulate in the dead-letter queue if your endpoint returns 4xx, or quietly succeed if you respond 200 and discard.

Listing, updating, and deleting

- `GET /v1/webhooks/endpoints` — list all endpoints for your partner account.
- `GET /v1/webhooks/endpoints/{id}` — retrieve a single endpoint.
- `PATCH /v1/webhooks/endpoints/{id}` — modify URL, label, or subscribed event types.
- `DELETE /v1/webhooks/endpoints/{id}` — disable an endpoint permanently. In-flight deliveries to a deleted endpoint are dropped; queued retries are cancelled.

Local development with webhook tunneling

In local development you cannot register `http://localhost:3000/...` because the URL must be publicly reachable for partnermax to deliver to it. Two recommended workflows:

ngrok

```
ngrok http 3000
# Forwarding  https://7f3a-203-0-113-42.ngrok-free.app -> http://localhost:3000
```

Register the ngrok URL as your endpoint, plus your local route path:

```
POST /v1/webhooks/endpoints

{
  "url": "https://7f3a-203-0-113-42.ngrok-free.app/webhooks/partnermax",
  "event_types": ["dealer.created", "dealer.updated"],
  "label": "local-dev-alice"
}
```

The ngrok hostname rotates on every restart on free plans — re-register or use `PATCH /v1/webhooks/endpoints/{id}` to update the URL. Consider an ngrok paid plan for a stable subdomain.

Cloudflare Tunnel

```
cloudflared tunnel --url http://localhost:3000
# https://random-words-1234.trycloudflare.com
```

Identical workflow to ngrok. Stable named tunnels are available on the Cloudflare Zero Trust free plan.

Triggering test events

When webhook preview is enabled for a partner account, DealerMAX may expose the sandbox-only endpoint `POST /v1/webhooks/send-test-event` to deliver a synthetic event of a chosen type to a chosen endpoint. The payload uses placeholder identifiers (clearly marked with `_test_` in the IDs) and `livemode: false`. Your handler logic should treat the test payload identically to a real one, but write to a test database — never to production storage.

You may also drive real events from the sandbox environment by authenticating with a `pmk_sand_*` key against the standard host (`api.dealermax.app`) — same routing model as every other endpoint, see Sandbox.

Security best practices

Always verify signature before processing

Treat any inbound webhook as untrusted until the HMAC verifies. An attacker who discovers your endpoint URL (for example via a leaked log) but does not know your secret can send arbitrary payloads — if your handler acts on payload contents before checking the signature, you have a vulnerability. Verification should be the first work your handler does.

Respond 2xx within 30 seconds

Acknowledge receipt and defer real work. A handler that calls your own database, third-party APIs, or AI services synchronously is at risk of timing out under load and entering an unproductive retry loop with partnermax. The standard pattern:

```
@app.route("/webhooks/partnermax", methods=["POST"])
def handle_webhook():
    raw = request.get_data()
    try:
        verify_webhook(raw, request.headers, WEBHOOK_SECRET)
    except WebhookVerificationError:
        abort(401)

    event = json.loads(raw)
    # Persist + enqueue. Total budget here: <100ms.
    db.execute(
        "INSERT INTO webhook_inbox (event_id, type, body) VALUES (%s, %s, %s) "
        "ON CONFLICT (event_id) DO NOTHING",
        (event["id"], event["type"], json.dumps(event)),
    )
    job_queue.enqueue("process_partnermax_event", event["id"])
    return "", 200
```

The `ON CONFLICT DO NOTHING` clause on `event_id` is your idempotency mechanism — see below.

Idempotent handler design

Because delivery is at-least-once, your handler may see the same event two or more times. Design every handler to be safe under repeated execution.

Three common patterns:

1. Inbox table with unique constraint. Persist `event_id` to a table with a unique index on it; ignore conflicts. This is the cleanest pattern when your handler does database work in the same connection.

2. Conditional update by versioned attribute. For `dealer.updated`, compare `occurred_at` to your local `last_updated_at` for the dealer; ignore older or equal timestamps.

3. Outbox + downstream idempotency. If you propagate the event to downstream systems, use the partnermax `event_id` as the upstream idempotency key for those calls. Downstream systems that themselves implement idempotency will then collapse duplicates.

Avoid relying on processing-time deduplication windows (TTL caches keyed on `event_id`) — they break on handler restarts, scale-outs, and the long tail of dead-letter replays.

Rotate secrets on suspected compromise

If you suspect your webhook secret has leaked (committed to a public repository, exposed in an error log, etc.), rotate immediately:

```
POST /v1/webhooks/endpoints/{id}/rotate-secret
```

The response returns a new `secret`. The previous secret is invalid the instant the rotation request completes. There is no overlap window — you must coordinate the deploy of the new secret to your handler with the rotation call.

For zero-downtime rotation, create a second endpoint subscribed to the same events with the new secret, verify it works end-to-end, then delete the old endpoint.

Source IP allow-listing

Optional defense-in-depth. The list of egress IPs is published at `https://developers.dealermax.app/.well-known/webhook-source-ips.json` and updated whenever the list changes. Subscribe to the `partnermax-infra` Atom feed at `https://developers.dealermax.app/feed/infra.atom` for advance notice of changes, typically 7 days before they take effect.

IP allow-listing should never substitute for signature verification, but it can reduce the attack surface for endpoints that are otherwise reachable from the public internet.

Replay window tuning

The default 5-minute tolerance window balances clock-skew tolerance against replay risk. Do not extend it without reason. If your infrastructure has demonstrably unusual NTP drift (you can measure it by logging `Partnermax-Timestamp - now()` for each request), tune as needed, but treat any extension beyond 15 minutes as a finding to investigate at the NTP layer rather than work around.

Monitor delivery health

The endpoint-stats API (`GET /v1/webhooks/endpoints/{id}/stats`) exposes delivery success rate, p50/p95 response latency, and recent failure samples per endpoint for integration with your own observability stack. Configure alerts on:

- Success rate dropping below 99% over a rolling 1-hour window.
- p95 response time exceeding 5 seconds — your handler is approaching the 30s timeout under load.
- Any deliveries entering the dead-letter queue.

Related documentation

- Authentication — partner API key model; webhook endpoint management endpoints are authenticated with the same API key.
- Errors — error format you should return to partnermax (4xx) versus errors you should never return (5xx for retryable bugs).
- Glossary — HMAC-SHA256
- Glossary — Idempotency-Key

CHAPTER 12

Error Handling

> *RFC 7807 Problem Details for HTTP APIs.*

Every non-2xx response from the partnermax API is a JSON document with the media type `application/problem+json`, conforming to RFC 7807. This is true for every partner-authenticated endpoint, including API key management.

This document covers:

- The exact shape of error responses, including our two RFC 7807 extension fields.
- The complete catalog of error codes mapped to HTTP statuses and recovery actions.
- Concrete JSON examples for the most common failure modes.
- Retry semantics and exponential backoff guidance.
- How to use the `trace_id` field when filing a support ticket.

Error response format

A typical error response looks like this:

```
HTTP/1.1 422 Unprocessable Entity
Content-Type: application/problem+json
Cache-Control: no-store

{
  "type": "https://developers.dealermax.app/errors/validation_error",
  "title": "Validation failed",
  "status": 422,
  "detail": "One or more fields failed schema validation. See 'errors' for per-field detail.",
  "instance": "/v1/dealers",
  "code": "validation_error",
  "trace_id": "00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01",
  "errors": [
    {
      "pointer": "/vat_number",
      "code": "invalid_vat_number",
      "message": "Value 'IT00000000000' is not a registered Italian VAT number in the VIES registry."
    },
    {
      "pointer": "/address/postal_code",
      "code": "invalid_postal_code",
      "message": "Value '999999' is not a valid Italian postal code."
    }
  ]
}
```

Field reference

Field	Required	Type	Description
`type`	Yes	URI	A URI identifying the error class. Resolvable at `https://developers.dealermax.app/errors/{code}` to a human-readable description. Stable across API versions.
`title`	Yes	string	Short, human-readable summary. English only. Stable across API versions for a given `type`.
`status`	Yes	integer	HTTP status code, duplicated in the body for clients that consume the body without inspecting headers.
`detail`	Optional	string	Per-occurrence explanation. May change between requests for the same `type` (e.g., includes the specific value that failed).
`instance`	Optional	string	URI path of the request that produced the error.
`code`	Yes (extension)	string	Machine-readable error code. Snake_case, stable across API versions. Use this for switch/branch logic in your client.
`trace_id`	Yes (extension)	string	W3C `traceparent` value (version-trace_id-span_id-flags). Include this in any support ticket. See trace_id propagation.
`errors`	Optional (extension)	array	For 422 validation errors, an array of per-field problems. Each entry has `pointer` (JSON Pointer per RFC 6901), `code`, and `message`.

What about localization?

All error `title` and `message` strings are in **English**. Localization to Italian or other languages is not provided at the API layer. Your application is expected to map our `code` values to whatever locale you serve.

Error code catalog

The table below covers every error code the partnermax API can return. The `type` column shows the suffix appended to `https://developers.dealermax.app/errors/` to produce the full URI.

4xx — Client errors

HTTP	`code`	`type` (suffix)	When it occurs	How to recover
400	`invalid_request`	`invalid_request`	The request body is not valid JSON, headers are malformed, or the URL path is missing a required parameter.	Inspect the `detail` field. Most often a JSON parse error or a missing `Content-Type`. Fix the request and retry.
401	`missing_api_key`	`missing_api_key`	Neither `X-Api-Key` nor `Authorization` header was sent.	Add the header. See Authentication.
401	`invalid_api_key`	`invalid_api_key`	The key value does not match the active key for this partner/environment (revoked, replaced, deleted, or typo).	Verify environment (sandbox vs production) and that your secret manager holds the current key. If the key is lost or exposed, request a support-managed replacement.
402	`subscription_inactive`	`subscription_inactive`	The DealerMAX subscription has entered a terminal state.	Restore payment in the billing portal. See Authentication → Subscription lifecycle.
402	`grace_period_expired`	`grace_period_expired`	Subscription was `past_due` / `unpaid` for ≥14 days.	Same as above.
403	`forbidden_dealer_not_owned`	`forbidden_dealer_not_owned`	The dealer ID in the path is not registered for the authenticated partner.	Use a dealer ID returned by `client.dealers.create(...)` or `GET /v1/dealers`. Cross-partner access is never granted.
403	`capability_required`	`capability_required`	The API key does not carry the capability required for this operation (e.g., a key-lifecycle operation without `can_issue_keys`).	Use the active partner key issued by DealerMAX support, or request support if the credential needs replacement.
403	`cannot_revoke_self`	`cannot_revoke_self`	`DELETE /v1/keys/{key_id}` targeted the same key that authenticated the request.	Request a support-managed replacement. PartnerMAX v1 keeps one active key per partner/environment and does not support self-revocation.
404	`dealer_not_found`	`dealer_not_found`	The dealer ID in the path does not exist under the calling partner (cross-partner queries also return 404, by design).	Verify the ID with `GET /v1/dealers`.
404	`offer_not_found`	`offer_not_found`	The offer ID does not exist for the given dealer, or the offer was withdrawn by the	Re-list with `GET /v1/dealers/{id}/nlt/offers`.

HTTP	`code`	`type` (suffix)	When it occurs	How to recover
			upstream NLT network.	
404	`api_key_not_found`	`api_key_not_found`	The key ID in `DELETE /v1/keys/{id}` does not exist for the calling partner (already revoked, never existed, or owned by a different partner — same response for all three to prevent enumeration).	Verify the key ID via `GET /v1/keys`.
409	`idempotency_key_reuse`	`idempotency_key_reuse`	An `Idempotency-Key` was reused within 24 hours with a different request body.	Either retry with the original body (which returns the cached response) or generate a fresh idempotency key.
409	`key_rotation_conflict`	`key_rotation_conflict`	`POST /v1/keys/issue` could not atomically rotate the current key because the key that authenticated the request is no longer active.	Fetch `GET /v1/keys` with the current active key if you have one, or contact DealerMAX support for a controlled replacement.
409	`dealer_deleted`	`dealer_deleted`	A `PATCH` was issued against a soft-deleted dealer. Soft-delete is a terminal state from the partner's side; re-creation requires DealerMAX support.	If accidental, contact support; otherwise create a new dealer record.
409	`dealer_already_deleted`	`dealer_already_deleted`	`DELETE /v1/dealers/{id}` was issued against a dealer that is already soft-deleted.	No action — the call is idempotent at the row level even though the API surfaces the 409 for observability.
422	`validation_error`	`validation_error`	One or more body fields failed schema validation. See the `errors` array for per-field detail.	Fix each field listed in `errors[]`. Each entry includes a `field`, a `code`, and a human-readable `message`.

5xx — Server errors

HTTP	`code`	`type` (suffix)	When it occurs	How to recover
500	`internal_error`	`internal_error`	Unhandled exception on our side. Always include the `trace_id` in a support ticket.	Retry with exponential backoff. If the error persists, file a support ticket with the `trace_id`.
502	`upstream_dependency_unavailable`	`upstream_dependency_unavailable`	A required upstream (Stripe, VIES, Supabase, MCP indexer) returned an error we could not paper over.	Retry with exponential backoff. Most upstream failures self-heal within minutes.
503	`service_unavailable`	`service_unavailable`	The API is shedding load (e.g., during a planned deploy) or in active maintenance.	Honor `Retry-After`. We never deploy without a graceful drain.
504	`upstream_timeout`	`upstream_timeout`	An upstream call (typically the pricing service or the indexer) exceeded its budget.	Retry with exponential backoff.

Concrete examples

`401 invalid_api_key`

```
HTTP/1.1 401 Unauthorized
Content-Type: application/problem+json
WWW-Authenticate: ApiKey realm="partnermax"

{
  "type": "https://developers.dealermax.app/errors/invalid_api_key",
  "title": "Invalid API key",
  "status": 401,
  "detail": "The API key in the X-Api-Key header is not recognized. The key may have been revoked, deleted, or is malformed
  "instance": "/v1/dealers",
  "code": "invalid_api_key",
  "trace_id": "00-9c8e7f6a5b4c3d2e1f0a9b8c7d6e5f4a-1234567890abcdef-01"
}
```

`402 subscription_inactive`

```
HTTP/1.1 402 Payment Required
Content-Type: application/problem+json
WWW-Authenticate: PartnerSubscription realm="partnermax", error="subscription_inactive"

{
  "type": "https://developers.dealermax.app/errors/subscription_inactive",
  "title": "Subscription inactive",
  "status": 402,
  "detail": "Your partner DealerMAX subscription is in status 'canceled'. Restore payment in the billing portal to re-enabl
  "instance": "/v1/dealers",
  "code": "subscription_inactive",
  "trace_id": "00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01",
  "subscription_status": "canceled",
  "detail": "Restore payment in the billing portal to resume API access."
}
```

`422 validation_error` (with per-field errors)

```
HTTP/1.1 422 Unprocessable Entity
Content-Type: application/problem+json

{
  "type": "https://developers.dealermax.app/errors/validation_error",
  "title": "Validation failed",
  "status": 422,
  "detail": "One or more fields failed schema validation. See 'errors' for per-field detail.",
  "instance": "/v1/dealers/verdicar-001/nlt-settings",
  "code": "validation_error",
  "trace_id": "00-1a2b3c4d5e6f70819a2b3c4d5e6f7081-9a8b7c6d5e4f3210-01",
  "errors": [
    {
      "pointer": "/agency_markup_percent",
      "code": "value_out_of_range",
      "message": "Value 25.0 exceeds the maximum allowed agency mark-up of 15.0 percent."
    },
    {
      "pointer": "/down_payment_tiers_eur",
      "code": "array_length_mismatch",
      "message": "Expected exactly 3 down-payment tiers, received 2."
    },
    {
      "pointer": "/down_payment_tiers_eur/0",
      "code": "value_below_minimum",
      "message": "Down-payment tier values must be ≥ 0. Received -500."
    }
  ]
}
```

`429 rate_limited`

```
HTTP/1.1 429 Too Many Requests
Content-Type: application/problem+json
Retry-After: 27
RateLimit-Limit: 300
RateLimit-Remaining: 0
RateLimit-Reset: 27

{
  "type": "https://developers.dealermax.app/errors/rate_limited",
  "title": "Rate limit exceeded",
  "status": 429,
  "detail": "You exceeded the per-key quota of 300 requests per minute for read endpoints. Retry after 27 seconds.",
  "instance": "/v1/dealers/verdicar-001/nlt/offers",
  "code": "rate_limited",
  "trace_id": "00-aa11bb22cc33dd44ee55ff66aa11bb22-1122334455667788-01",
  "limit": 300,
  "window_seconds": 60,
  "retry_after_seconds": 27
}
```

Retry guidance

The table below tells you whether to retry and what backoff strategy to apply.

HTTP	`code`	Retryable?	Strategy
400	`invalid_request`	No	Fix the request. Retrying without changes will fail again.
401	`missing_api_key` / `invalid_api_key`	No	Authenticate, then retry.
402	`subscription_inactive` / `grace_period_expired`	No	Restore payment, then retry.
403	All `forbidden_*` / `capability_required` / `cannot_revoke_self`	No	Fix the underlying authorization condition.
404	All `*_not_found`	No	Verify the ID.
409	`idempotency_key_reuse`	No	Generate a fresh idempotency key, or retry with the original body.
409	`key_rotation_conflict`	No	Refresh key metadata with an active key or contact DealerMAX support.
409	`dealer_domain_already_taken`	No	Choose a different domain.
422	`validation_error`	No	Fix each field in `errors[]`.
429	`rate_limited`	**Yes**	Honor `Retry-After`. Apply jitter. See Rate Limits → Exponential backoff.
500	`internal_error`	**Yes, with caution**	Exponential backoff (see below). Cap at 3 attempts. If still failing, file a support ticket with `trace_id`.
502	`upstream_dependency_unavailable`	**Yes**	Exponential backoff. Cap at 5 attempts.
503	`service_unavailable`	**Yes**	Honor `Retry-After`. If absent, exponential backoff.
504	`upstream_timeout`	**Yes**	Exponential backoff. Cap at 3 attempts.

Exponential backoff

For any retryable error without a `Retry-After` header, use full-jitter exponential backoff:

```
base = 500ms
cap  = 30s
delay_n = random_between(0, min(cap, base * 2^n))
```

Where `n` is the zero-indexed retry attempt number (n=0 for the first retry).

The official SDKs implement this strategy by default and will not retry non-idempotent operations (anything other than `GET`, `HEAD`, or a request carrying an `Idempotency-Key` header).

Mutating requests and idempotency

The platform retains a 24-hour idempotency cache keyed on `(partner_id, Idempotency-Key)`. Repeating a `POST` or `PATCH` with the same idempotency key returns the **original response body and status code**, even if the underlying state has since changed. This makes mutating retries safe.

If you retry with the same key but a different body, you receive `409 idempotency_key_reuse` — the platform refuses to silently process a different request under the same key, to prevent accidental state divergence.

`trace_id` propagation

The `trace_id` field in every problem response is the value of the W3C traceparent the platform observed for the failing request. The format is:

```
00-{trace-id}-{parent-id}-{trace-flags}
```

For example: `00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01`.

Sending your own traceparent

If your client sends a `traceparent` header on the request, the platform honors it and returns the same trace ID in the error response. This lets you correlate your application logs with our backend traces:

```
curl -X POST https://api.dealermax.app/api/partner/dealers \
  -H "X-Api-Key: $PARTNERMAX_API_KEY" \
  -H "Content-Type: application/json" \
  -H "traceparent: 00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01" \
  -H "Idempotency-Key: provision-acme-2026-05-17" \
  -d '{"legal_name": "ACME Motors S.R.L.", ...}'
```

If you do not send a `traceparent`, the platform generates one. Either way, the value comes back in the `trace_id` field of any error response and in the `traceresponse` header of every response (success or failure).

Filing a support ticket

When opening a support ticket, always include:

1. The `trace_id` from the problem response.
2. The `instance` URI (the failing path).
3. The approximate timestamp (UTC).
4. The `code` and HTTP status.
5. Your partner ID (returned in the response of any successful API call under `partner_id`, and shown at onboarding).

With those five fields, our on-call engineer can pull the full trace, correlated logs from the API layer, and downstream calls (Stripe webhook history, Supabase queries, indexer state) within minutes.

Do **not** include the API key value or any partner secret in a support ticket. The `trace_id` alone is sufficient.

Secret redaction in client logs

The official SDKs redact credentials in normal exception strings and object representations. Treat raw HTTP request headers as sensitive anyway: `exc.request.headers`, transport debug dumps, and reverse-proxy captures can still contain the exact `X-Api-Key` or `Authorization` value the client sent. Never log raw request headers unless your logger masks those fields before persistence.

Next steps

- Authentication — The full auth model, including the subscription-gated lifecycle and 402 cascade.
- Rate Limits — Quotas, headers, and backoff specifics for 429 responses.
- API Reference — Per-endpoint request/response schemas (OpenAPI 3.1).
- Getting Started — End-to-end walkthrough.

CHAPTER 13

Rate Limits

The partnermax API publishes per-key rate-limit quotas to protect platform stability and ensure fair access. Limits are scoped to the API key, not to source IP. This document covers the quota table, the response headers we emit, how to handle `429 Too Many Requests`, and how to request higher limits.

> ***v1 enforcement state.*** *The middleware enforces the published per-minute budgets through Redis and emits `RateLimit-*` plus `X-RateLimit-*` response headers on every protected `/v1/` response. When a bucket is exhausted the API returns `429 Too Many Requests` with `Retry-After`. If Redis is temporarily unreachable, partnermax fails open and logs the degradation rather than blocking the API on an auxiliary dependency.*

Limits at a glance

The default limits below apply to every newly-issued API key, in both sandbox and production environments. Sandbox and production keys have **separate, independent counters**.

Bucket	Limit	Window	Endpoints
Default read	**300 req/min**	Fixed 60-second window, per API key	All `GET` endpoints
Mutating	**60 req/min**	Fixed 60-second window, per API key	All `POST`, `PATCH`, `DELETE` endpoints

Window reset behavior

Counters reset on the server's minute boundary. The `RateLimit-Reset` header is the number of seconds until the current fixed window resets. There is no automatic short-window burst allowance in v1; plan bulk jobs against the sustained per-minute ceiling or coordinate a temporary operational increase through support.

Separate buckets

- Each API key has its own counter, but PartnerMAX v1 allows only one active key per partner account per environment. Historical, revoked, or expired keys do not add usable quota.
- Sandbox and production are fully isolated. Sandbox throttling never affects production.
- Source IP is not factored into the default quotas.

Response headers

Every response — successful or throttled — carries the RFC 9239 draft standard rate-limit headers. `X-RateLimit-*` alias headers are also emitted with the same values.

RFC 9239 draft headers (preferred)

Header	Meaning
`RateLimit-Limit`	The quota ceiling for the bucket that handled this request.
`RateLimit-Remaining`	Number of requests remaining in the current window. May go to 0.
`RateLimit-Reset`	Seconds until the current window resets and `Remaining` is replenished. Always a positive integer.

Example on a successful response:

```
HTTP/1.1 200 OK
Content-Type: application/json
RateLimit-Limit: 300
RateLimit-Remaining: 247
RateLimit-Reset: 42
```

Alias headers (also emitted)

The API also emits the `X-RateLimit-*` family popularized by GitHub and others:

Header	Equivalent RFC 9239 header
`X-RateLimit-Limit`	`RateLimit-Limit`
`X-RateLimit-Remaining`	`RateLimit-Remaining`
`X-RateLimit-Reset`	`RateLimit-Reset` (also expressed in seconds, not unix-epoch — note this differs from the GitHub convention)

We recommend new integrations rely on the standard `RateLimit-*` headers.

`429 Too Many Requests` response

When you exceed a quota, the next request returns:

```
HTTP/1.1 429 Too Many Requests
Content-Type: application/problem+json
Retry-After: 27
RateLimit-Limit: 300
RateLimit-Remaining: 0
RateLimit-Reset: 27

{
  "type": "https://developers.dealermax.app/errors/rate_limited",
  "title": "Rate limit exceeded",
  "status": 429,
  "detail": "You exceeded the per-key quota of 300 requests per minute for read endpoints. Retry after 27 seconds.",
  "instance": "/v1/dealers/verdicar-001/nlt/offers",
  "code": "rate_limited",
  "trace_id": "00-aa11bb22cc33dd44ee55ff66aa11bb22-1122334455667788-01",
  "limit": 300,
  "window_seconds": 60,
  "retry_after_seconds": 27
}
```

The `Retry-After` header is always present on a `429` response and is always expressed in **seconds** (not as an HTTP-date). It is authoritative — sleeping for the indicated duration guarantees the next request will not be throttled by the same bucket.

The body conforms to RFC 7807; see Error Handling for the full problem-details specification.

Exponential backoff recommendation

For any `429` response, honor `Retry-After`. For other retryable errors without an explicit retry hint, apply full-jitter exponential backoff:

```
base = 500ms
cap  = 30s
delay_n = random_between(0, min(cap, base * 2^n))
```

Where `n` is the zero-indexed attempt number. Cap retries at **5 attempts** for read endpoints and **3 attempts** for mutating endpoints. After the cap, surface the error to the caller.

curl

```
#!/usr/bin/env bash
set -euo pipefail

URL="https://api.dealermax.app/v1/dealers"
MAX_RETRIES=5

for n in $(seq 0 $((MAX_RETRIES - 1))); do
  response=$(curl -sS -w "\n%{http_code}" -H "X-Api-Key: $PARTNERMAX_API_KEY" "$URL")
  body=$(echo "$response" | head -n -1)
  status=$(echo "$response" | tail -n 1)

  if [ "$status" = "200" ]; then
    echo "$body"
    exit 0
  fi

  if [ "$status" = "429" ] || [ "$status" = "503" ] || [ "$status" = "502" ] || [ "$status" = "504" ]; then
    retry_after=$(echo "$body" | jq -r '.retry_after_seconds // empty')
    if [ -n "$retry_after" ]; then
      sleep "$retry_after"
    else
      # Full-jitter exponential backoff
      max_delay=$(awk -v n="$n" 'BEGIN { d = 0.5 * (2 ^ n); print (d > 30 ? 30 : d) }')
      delay=$(awk -v max="$max_delay" 'BEGIN { srand(); print max * rand() }')
      sleep "$delay"
    fi
    continue
  fi

  # Non-retryable error
  echo "Request failed with status $status: $body" >&2
  exit 1
done

echo "Exhausted $MAX_RETRIES retries." >&2
exit 1
```

Python

```python
import os
import random
import time
import httpx

API_KEY = os.environ["PARTNERMAX_API_KEY"]
BASE_URL = "https://api.dealermax.app"
MAX_RETRIES = 5
BASE_DELAY = 0.5
CAP_DELAY = 30.0


def request_with_backoff(method: str, path: str, **kwargs) -> httpx.Response:
    headers = kwargs.pop("headers", {})
    headers["X-Api-Key"] = API_KEY

    with httpx.Client(base_url=BASE_URL, timeout=30.0) as client:
        for attempt in range(MAX_RETRIES):
            response = client.request(method, path, headers=headers, **kwargs)

            if response.status_code < 400:
                return response

            if response.status_code in (429, 502, 503, 504):
                retry_after = response.headers.get("Retry-After")
                if retry_after is not None:
                    delay = float(retry_after)
                else:
                    delay = random.uniform(0, min(CAP_DELAY, BASE_DELAY * (2 ** attempt)))
                time.sleep(delay)
                continue

            # Non-retryable: raise immediately with the problem-details body
            response.raise_for_status()

        raise RuntimeError(f"Exhausted {MAX_RETRIES} retries for {method} {path}")


# Usage
response = request_with_backoff("GET", "/v1/dealers")
print(response.json())
```

The official Python SDK applies this strategy automatically:

```python
from partnermax import Partnermax

client = Partnermax(
    api_key=os.environ["PARTNERMAX_API_KEY"],
    max_retries=5,          # default
    timeout=30.0,           # default seconds
)
dealers = client.dealers.list()
```

TypeScript

```typescript
const API_KEY = process.env.PARTNERMAX_API_KEY!;
const BASE_URL = "https://api.dealermax.app";
const MAX_RETRIES = 5;
const BASE_DELAY_MS = 500;
const CAP_DELAY_MS = 30_000;

async function requestWithBackoff(
  method: string,
  path: string,
  init: RequestInit = {},
): Promise<Response> {
  const headers = new Headers(init.headers);
  headers.set("X-Api-Key", API_KEY);

  for (let attempt = 0; attempt < MAX_RETRIES; attempt++) {
    const response = await fetch(`${BASE_URL}${path}`, { ...init, method, headers });

    if (response.status < 400) {
      return response;
    }

    if ([429, 502, 503, 504].includes(response.status)) {
      const retryAfter = response.headers.get("Retry-After");
      const delayMs = retryAfter
        ? Number(retryAfter) * 1000
        : Math.random() * Math.min(CAP_DELAY_MS, BASE_DELAY_MS * 2 ** attempt);
      await new Promise((resolve) => setTimeout(resolve, delayMs));
      continue;
    }

    // Non-retryable: surface the problem-details body
    const problem = await response.json();
    throw Object.assign(new Error(problem.title), { problem, status: response.status });
  }

  throw new Error(`Exhausted ${MAX_RETRIES} retries for ${method} ${path}`);
}

// Usage
const response = await requestWithBackoff("GET", "/v1/dealers");
const dealers = await response.json();
```

The official Python SDK applies this strategy automatically:

```python
from partnermax import Partnermax

client = Partnermax(
    api_key=os.environ["PARTNERMAX_API_KEY"],
    max_retries=5,          # default
    timeout=30.0,           # default seconds
)

dealers = client.dealers.list()
```

The official TypeScript / JavaScript SDK is published as `partnermax` on npm and handles transient retry helpers consistently with the Python SDK. JS consumers that call REST directly should implement the back-off loop shown above.

Enterprise tier and custom limits

The default quotas above suit most partner integrations. If your traffic profile requires higher sustained throughput — large dealer rosters, frequent full-catalog re-syncs, or high-frequency MCP-driven query patterns — contact your account manager to negotiate an **Enterprise tier** with custom limits. Support will confirm the active ceiling and rollout timing before the higher limit is used in production traffic.

Enterprise tier is included in the platform commercial agreement; there is no separate API surface for self-service limit elevation.

Temporary operational limit increases

For one-off events — initial dealer onboarding bulk-load, catalog re-sync after a content migration, planned load test — you can request a **temporary operational limit increase** through support:

1. Open a support ticket via your account manager, or by emailing `support@dealermax.app`.
2. Provide: the API key ID, the target start time (UTC), the requested duration (max 4 hours), and the expected peak rate.
3. Once approved, the elevated limit is applied automatically at the start time and reverts at the end time.

Standard approval turnaround is **1 business day**. Emergency requests for production outage mitigation are treated as Sev 1 support requests and prioritized immediately; no separate public approval-time guarantee is published for emergency limit changes.

Temporary limit increases are billed at standard rates — they are operational accommodations, not commercial upgrades.

Next steps

- Error Handling — Full RFC 7807 problem-details specification and the complete error code catalog.
- Authentication — API key model, including how to issue and rotate keys.
- API Reference — Per-endpoint reference.
- Getting Started — End-to-end walkthrough.

CHAPTER 14

Versioning and Deprecation

This document defines how the Partnermax API evolves over time, the guarantees we make to integrators about stability, and the mechanics through which we communicate breaking changes. The goal is simple: integrations built today should keep working tomorrow, and when a breaking change is unavoidable, partners should have enough lead time and signal density to migrate without firefighting.

Versioning Model

The Partnermax API uses **URL-based major versioning**. The major version is encoded as the first path segment after the host:

```
https://api.dealermax.app/v1/dealers
https://api.dealermax.app/v2/dealers   # future
```

There is no `Accept` header version negotiation, no query parameter version, and no date-pinned version. The only versioning signal is the path prefix.

Why URL-based

URL-based versioning was chosen for three reasons that matter at partner scale:

1. **Cache key correctness.** CDN and reverse-proxy caches key on URL by default. Header-based versioning forces every layer in the stack to be version-aware or risk serving the wrong payload to the wrong client.
2. **Observability.** Server logs, APM traces, and request metrics expose the path natively. Version-by-header requires custom dimensions in every monitoring tool the partner uses.
3. **Debuggability.** A partner engineer reading a stack trace, a log line, or a cURL example can identify the API version at a glance.

Semantic Versioning for SDKs

While the wire protocol uses major-only versioning in the URL, the **official SDKs follow Semantic Versioning 2.0.0** (Semver). SDK versions match the API major version they target:

SDK version range	Targets API version
`1.x.y`	`/v1/`
`2.x.y`	`/v2/` (future)

The Semver components carry the following meaning for Partnermax SDKs:

- **MAJOR** (`1.x.y` → `2.x.y`): the SDK now targets a new API major version. The wire surface has breaking changes. Code changes are required to upgrade.
- **MINOR** (`1.2.y` → `1.3.y`): the SDK adds support for new endpoints, new optional fields, new error subcodes, or new methods on existing resources. Existing code keeps working.
- **PATCH** (`1.2.3` → `1.2.4`): bug fixes, dependency bumps, internal refactors, documentation. No behavioral changes for the integrator.

SDKs are auto-regenerated from the OpenAPI specification on every change (see SDK Auto-Bump Policy below). A partner pinning `^1.2.0` in `package.json` or `requirements.txt` will receive new features and bug fixes automatically and will never receive a breaking change until they explicitly bump the major version.

Stability Contract

Within a single major version, the following changes are **always allowed** and may ship at any time without prior notice:

- **New endpoints.** A new resource path or a new HTTP method on an existing path.
- **New optional response fields.** Existing fields keep their type and meaning; new fields are additive.
- **New optional request fields.** Existing fields keep their behavior; new fields default to a safe value when omitted.
- **New error subcodes.** Within an existing HTTP status code and error category, new `type` values may be introduced. Integrators should match on the documented `type` values they handle and fall back to the HTTP status family for unknown subcodes.
- **New enum values in response fields.** Integrators should treat unknown enum values as forward-compatible, not as an error.
- **New webhook event types.** Subscribers receive only the event types they have explicitly subscribed to; introducing new event types does not affect existing subscriptions.
- **Internal performance improvements, error message wording, latency changes within the SLO band.**

The following changes are **never allowed** within a single major version:

- **Field removal.** A field present in a `/v1/` response will never disappear from `/v1/`.
- **Field rename.** A field's JSON key will not change.
- **Field type change.** A field documented as `string` will not become an `integer`, an `array`, or an `object`.
- **Endpoint removal.** A path and method that returns a 2xx today will continue to be routable in `/v1/`.
- **HTTP status code change for an existing outcome.** If `GET /v1/dealers` returns `200 OK` today, it will continue to return `200` for the same outcome.
- **Header removal.** Headers documented as part of a response will continue to be sent.
- **Stricter validation on existing fields.** A request that validates today will continue to validate. We may relax validation; we will not tighten it.
- **Semantic change to existing fields.** A field documented as "amount in cents" will not become "amount in euros."

Breaking changes happen only at major version boundaries: `/v1/` → `/v2/`. They follow the Breaking Change Policy below.

Breaking Change Policy

When a breaking change is necessary, Partnermax commits to a **minimum 180-day deprecation window** before the deprecated surface is removed. The window is measured from the day the `Deprecation` header is first emitted in production responses to the day the endpoint returns `404 Not Found` or `410 Gone`.

The 180-day window is a floor, not a ceiling. For changes affecting core resources (dealers, NLT configurations, subscriptions), the window is typically 270 days. Enterprise-tier partners can request additional extensions on a case-by-case basis, negotiated through their named technical contact.

During the deprecation window:

1. The deprecated endpoint continues to function exactly as documented. No behavior changes.
2. Every response includes the RFC 8594 `Deprecation` and `Sunset` headers (see RFC 8594 Headers below).
3. The migration guide for the successor version is published before the deprecation window opens.
4. A dedicated entry appears in the changelog on the announcement date.
5. All partner contacts on file receive an announcement email.
6. Pro and Enterprise tier partners receive a Slack notification in their dedicated channel.

After the sunset date:

- The endpoint returns `410 Gone` with a Problem Details body pointing to the successor.
- The endpoint remains in the OpenAPI specification for 30 additional days, marked `deprecated: true` and with a description noting the sunset date, to aid SDK consumers debugging stale clients.

- After the 30-day grace period the endpoint is removed from the OpenAPI specification entirely.

RFC 8594 Headers

Partnermax implements RFC 8594 — The Sunset HTTP Header Field and the related Internet-Draft for the Deprecation header.

Deprecated endpoints include three headers on every response:

Header	Format	Meaning
`Deprecation`	`@<unix-timestamp>`	The instant the deprecation was announced. Per RFC, the `@` prefix marks the value as a Unix timestamp in seconds.
`Sunset`	RFC 7231 HTTP-date	The planned removal date. After this date the endpoint may return `410 Gone`.
`Link`	`<url>; rel="successor-version"`	The URL of the successor endpoint or migration guide.

Example: deprecated endpoint, 180-day window

```
HTTP/1.1 200 OK
Content-Type: application/json
Deprecation: @1740009600
Sunset: Wed, 19 Aug 2026 23:59:59 GMT
Link: <https://developers.dealermax.app/docs/migrations/v1-to-v2#dealers-list>; rel="successor-version"
X-Request-Id: 4f8c1d2a-5e6b-4a91-8c32-9d7e4b1a5f6c
Partnermax-Api-Version: 1

{
  "object": "list",
  "data": [...],
  "has_more": false
}
```

In this example, `@1740009600` resolves to **Wednesday, 19 February 2026 00:00:00 UTC**. The sunset date is exactly 181 days later. The `Link` header points at the migration guide section covering the `dealers.list` change.

Example: deprecated endpoint, final 30 days before sunset

When fewer than 30 days remain before the sunset date, the response additionally includes a `Warning` header in the format defined by RFC 9111:

```
HTTP/1.1 200 OK
Content-Type: application/json
Deprecation: @1740009600
Sunset: Wed, 19 Aug 2026 23:59:59 GMT
Link: <https://developers.dealermax.app/docs/migrations/v1-to-v2#dealers-list>; rel="successor-version"
Warning: 299 - "This endpoint will be removed on 2026-08-19. Migrate to /v2/dealers."
X-Request-Id: 4f8c1d2a-5e6b-4a91-8c32-9d7e4b1a5f6c
Partnermax-Api-Version: 1

{
  "object": "list",
  "data": [...],
  "has_more": false
}
```

The `Warning` header gives a human-readable hint that surfaces in many HTTP client loggers without requiring structured-log integration.

Example: response after sunset

```
HTTP/1.1 410 Gone
Content-Type: application/problem+json
X-Request-Id: 4f8c1d2a-5e6b-4a91-8c32-9d7e4b1a5f6c
Partnermax-Api-Version: 1

{
  "type": "https://developers.dealermax.app/errors/endpoint_sunset",
  "title": "Endpoint removed",
  "status": 410,
  "detail": "GET /v1/dealers was removed on 2026-08-19. Use GET /v2/dealers. See the migration guide for the request and re
  "instance": "/v1/dealers",
  "successor": "https://api.dealermax.app/v2/dealers",
  "migration_guide": "https://developers.dealermax.app/docs/migrations/v1-to-v2#dealers-list"
}
```

Partners should treat `410 Gone` on a previously-working endpoint as a deployment-stopping signal in CI: it means the SDK is too old for the production API surface.

Deprecation Announcement Channels

When an endpoint is deprecated, the announcement reaches partners through four independent channels. The redundancy is intentional: a partner who misses the changelog will see the response header, and a partner whose alerting suppresses response headers will get the email.

1. **Response headers.** Every response to the deprecated endpoint includes `Deprecation`, `Sunset`, and `Link` from the announcement date forward. This is the most reliable channel because it reaches every running integration regardless of human attention.
2. **Changelog.** A dated entry at developers.dealermax.app/docs/changelog describes the change, the successor, the sunset date, and links to the migration guide. The changelog is also available as an RSS feed at `developers.dealermax.app/docs/changelog.rss`.
3. **Email to partner contacts.** Every partner has at least one technical contact on file. Deprecation announcements go to all technical contacts associated with the partner account. The email contains the same information as the changelog entry plus a partner-specific impact summary computed from the partner's recent request logs (which deprecated endpoints they currently call).
4. **Slack notification for Pro and Enterprise tiers.** Pro and Enterprise tier partners have a dedicated Slack channel with the Partnermax team. Deprecation announcements are posted there with the same content as the email plus a thread for follow-up questions.

Free Trial and Starter tier partners receive announcements only through response headers, the changelog, and email. The Slack channel for these tiers is the community Slack workspace, where general announcements are also posted but without per-partner impact summaries.

SDK Auto-Bump Policy

Partnermax uses Stainless to generate official SDKs from the OpenAPI specification. The generation pipeline runs on every merge to the OpenAPI spec repository.

The Semver bump rule is enforced automatically by the generation pipeline:

OpenAPI change	SDK version bump
New endpoint, new optional field, new error subcode, new enum value	MINOR (`1.2.3` → `1.3.0`)
Documentation-only change, internal SDK refactor, dependency bump	PATCH (`1.2.3` → `1.2.4`)
API major version increment (`/v1/` → `/v2/`)	MAJOR (`1.x.y` → `2.0.0`)

Major bumps never happen as a side effect of a routine OpenAPI change. They require an explicit decision and a new major version path in the spec, and they always coincide with the publication of a migration guide.

Supported SDK languages

Generated SDKs in v1:

Language	Package manager	Package name	Status
Python	PyPI	`partnermax`	Generally available
TypeScript / JavaScript	npm	`partnermax`	Generally available — generated from the same OpenAPI 3.1 spec via Stainless, published from CI with signed build provenance

Additional language SDKs (Go, Java, Ruby, PHP) are not on the published roadmap; partners working in those languages can request the PartnerMAX OpenAPI 3.1 contract during approved onboarding and generate a client with their preferred generator (`openapi-generator`, `oapi-codegen`, `swagger-codegen`, etc.). The spec is the authoritative contract — any code-generation tooling that consumes OpenAPI 3.1 will produce a working client.

The Python SDK ships with type hints regenerated from the OpenAPI spec on every release, so a new optional response field appears in the SDK types the same day it appears in the API.

Pinning recommendations

For application code, pin the Python SDK with a tilde constraint:

```
# pyproject.toml
[project]
dependencies = [
    "partnermax ~= 1.5"
]
```

```
# pyproject.toml
[project]
dependencies = ["partnermax>=1.5,<2"]
```

```
// go.mod
require github.com/dealermax/partnermax-go v1.2.0
```

This pattern picks up MINOR and PATCH bumps automatically (forward-compatible by the stability contract) and requires an explicit version bump for MAJOR.

For library code that embeds the Partnermax SDK as a transitive dependency, prefer a broader constraint such as `^1.0.0` to maximize compatibility with downstream applications.

Migration Guides

Every `/vN/` → `/v(N+1)/` transition is accompanied by a dedicated migration guide published at:

```
developers.dealermax.app/docs/migrations/v{N}-to-v{N+1}
```

For example, when `/v2/` ships, the migration guide for partners on `/v1/` will be at `developers.dealermax.app/docs/migrations/v1-to-v2`.

A migration guide always contains:

1. **Summary** — a 3-5 sentence elevator pitch of why the major version was incremented, in language a non-engineering stakeholder can read.
2. **Timeline** — the announcement date, the sunset date, and any milestones in between (for example, a sandbox-only early-access date).
3. **Change inventory** — a structured list of every breaking change, grouped by resource. Each entry has the old shape, the new shape, the migration step, and a link to a code-diff example in each SDK language.
4. **Behavior-only changes** — changes that do not break the wire protocol but change semantics (for example, idempotency key TTL extended from 24 hours to 7 days). Integrators should review these even if their code compiles against both versions.
5. **Cookbook** — common migration patterns. For each pattern, side-by-side `/v1/` and `/v2/` code samples in TypeScript, Python, and cURL.
6. **FAQ** — questions surfaced during the early-access period.
7. **Support** — how to escalate migration issues, with the relevant Slack channel and email address for each tier.

The migration guide is published **at least 60 days before** the announcement date, so partners can read it before the deprecation window starts.

Beta and Experimental Endpoints

New API surface is occasionally released in beta before being promoted to a stable `/v1/` endpoint. Beta endpoints live at:

```
https://api.dealermax.app/v1/beta/<resource>
```

The `/v1/beta/` path prefix is the only signal that an endpoint is unstable. Beta endpoints have **no stability guarantees**: they may change shape, return new error codes, change rate limits, or be removed without notice.

Opting in

To call a beta endpoint, the request must include the opt-in header:

```http
GET /v1/beta/risk-scoring/dealers/verdicar-001 HTTP/1.1
Host: api.dealermax.app
Authorization: Bearer pmk_live_...
Partnermax-Experimental: true
```

Without the `Partnermax-Experimental: true` header, beta endpoints return `403 Forbidden` with a Problem Details body indicating the opt-in requirement. This guards against accidental dependence on unstable surfaces in production code paths.

What “beta” means in practice

Beta endpoints are exercised by a subset of partners under a feedback loop with the Partnermax engineering team. They go through three states before stabilization:

1. **Alpha** — internal only. Not documented externally, no `/v1/beta/` exposure.
2. **Private beta** — feature-flagged per partner. The Partnermax-Experimental header is necessary but not sufficient; the partner must also be enrolled. Partners receive a Slack message inviting them to the private beta with a brief on the feedback expected.
3. **Public beta** — open to any partner that sends the opt-in header. The endpoint is documented in the public reference, marked with a `Beta` badge, and changelog entries for it are tagged `[beta]`.

When a beta endpoint stabilizes, it is moved out of `/v1/beta/` and onto `/v1/`. The `/v1/beta/` path remains routable for the same 30-day grace period as a deprecated endpoint, then returns `410 Gone`. The migration from beta to stable is documented in a short migration note in the changelog, but does not get a full migration guide because beta consumers have already signed up for instability.

What beta is not

Beta is not a vehicle for shipping breaking changes to stable endpoints. A change that would break an existing `/v1/` endpoint always goes through the `/v2/` major-version process described above, regardless of how small the change is. The beta channel is reserved for net-new surface area.

Summary Checklist for Integrators

Concern	Action
Stay on the current major version	Pin SDK with `^1.x` constraint, let MINOR/PATCH bumps in automatically
React to deprecations	Monitor for `Deprecation` and `Sunset` response headers; subscribe to the changelog RSS feed
Plan major-version migrations	Read the migration guide as soon as it is published, schedule a 60-day implementation window
Use beta endpoints safely	Gate them behind a feature flag in your code; never put a beta endpoint on the critical path of a customer-facing flow
Handle sunset day	Catch `410 Gone`, log it loudly, fall back to a degraded mode if possible, alert your on-call

Related Documents

- 03 — Authentication — API key formats and rotation policy
- 05 — Errors — Problem Details schema and error subcode taxonomy
- 09 — Sandbox — how to test deprecation behavior before it ships to production
- 10 — SLA — uptime and latency commitments, including for deprecated endpoints during the sunset window
- Changelog — chronological record of all API changes

CHAPTER 15

Sandbox Environment

> *Test the full integration without touching production data.*

The partnermax sandbox is a copy of the production API surface that lets partner engineers exercise every code path — provisioning dealers, configuring NLT economics, fetching dealer-aware catalog data, simulating Stripe subscription lifecycle transitions — without risk of leaking test data into the live DealerMAX network or charging real payment instruments.

This document describes what the sandbox is, how it differs from production, how to get access, what data is pre-populated, and a worked end-to-end example you can run against your own sandbox key in under fifteen minutes.

What the sandbox is

The sandbox is reachable on the **same host** as production:

```
https://api.dealermax.app
```

Routing between sandbox and production is determined by the key prefix: any request authenticated with a `pmk_sand_*` key resolves against the sandbox data plane (a separate Supabase database and Stripe test-mode account, both in the EU `eu-central-1` region), and any request authenticated with a `pmk_live_*` key resolves against production. The OpenAPI specification, the SDK code paths, the response shapes, the error subcodes, the rate-limit policy, and the authentication model are all identical to production.

What differs is the data and the side effects:

- Sandbox dealers **do not** appear in the production MCP server at `mcp.dealermax.app`.
- Sandbox dealers **do not** appear in the published ChatGPT Custom GPT.
- Sandbox dealers **do not** appear in the NLWeb `/ask` results served from production dealer sites.
- Sandbox Stripe operations run in Stripe test mode (no real money moves).

This isolation is enforced at the infrastructure level (separate Supabase project, separate Stripe account) and at the application level by the auth resolver: a request authenticated with a sandbox key cannot read or mutate production rows, and vice versa.

Differences from production

Where a row below says “same as production,” the partner code should not need any conditional logic.

Aspect	Production	Sandbox
Host	`api.dealermax.app`	`api.dealermax.app` (same host)
Environment band	`pmk_live_*` key	`pmk_sand_*` key
API surface (paths, methods, request shapes, response shapes)	`/v1/*`	Same as production
Authentication model	API key in `X-Api-Key` or `Authorization: Bearer`	Same
Underlying database	Supabase Frankfurt, production project	Supabase Frankfurt, separate sandbox project
Stripe integration mode	Live mode	Test mode
Stripe webhooks	Live-mode signing key	Test-mode signing key
Rate limits	Per the tier on the partner contract	Same numerical limits
Uptime SLA	99.9% (see SLA)	None — best-effort, may be down during deploys
Data retention	Indefinite (subject to commercial agreement)	30 days, after which sandbox data may be wiped
Cross-network indexing in MCP / Custom GPT / NLWeb	Yes — production dealers are indexed	No — sandbox dealers are never exposed
Outbound webhook delivery to partner endpoints	Not available in v1	Not available in v1
Support response time	Per tier (Free / Starter / Pro / Enterprise)	Best-effort, slower than production tier
API version (`/v1/`)	All published versions	Same

Two practical consequences worth highlighting:

1. **Rate limits are the same in absolute terms.** A partner with a 1000 req/min ceiling in production has the same 1000 req/min ceiling in sandbox. Load tests run against sandbox produce numbers comparable to what the production budget allows. Note however that sandbox is not load-test capacity — see Limitations.
2. **Key bands are strictly isolated.** A `pmk_sand_*` key submitted with any production-scope request returns `401 invalid_api_key` (the row is absent from the production tables it would need to resolve against). A `pmk_live_*` key against sandbox-scope requests behaves identically. There is no way to accidentally hit the wrong environment with the wrong key — the prefix is part of the routing decision.

Getting sandbox access

Sandbox access is granted on request. There are two paths.

Path 1 — as an existing partner

If your organization already has an active partnermax partner account, request a sandbox key by emailing `support@dealermax.app` from the registered partner technical contact. Provide:

- The partner account name.
- The engineering contact email that should receive the credential.
- The intended use (local integration, automated tests, partner QA, etc.).

Sandbox keys are issued within **1 business day** (typically same business day) and delivered to the named engineering contact through a secure-credential delivery flow. The raw key is shown exactly once at delivery time and must be stored in the partner's secret manager immediately.

Path 2 — before the production contract is signed

Prospective partners evaluating partnermax can request a sandbox-only account without committing to the full commercial agreement. Email `support@dealermax.app` with:

- Company name and primary engineering contact.
- A 2-3 sentence description of the integration the partner is exploring.
- Expected sandbox usage (number of test dealers, peak request rate during testing).

The DealerMAX commercial team reviews evaluation requests and issues a sandbox key within 1 business day. Evaluation sandbox keys carry the same default lifetime as standard sandbox keys.

Sandbox key lifetime

Sandbox keys do not auto-expire in v1. They remain valid until replaced by DealerMAX support or until the sandbox account is decommissioned. Partners are expected to rotate sandbox keys on the same 90-day cadence recommended for production (see Authentication).

Number of sandbox keys per partner

Each partner can hold **one active sandbox key** and **one active production key**. Use your own secret manager to distribute the sandbox key to engineers, CI jobs, and QA environments. If the sandbox key is lost, exposed, or due for routine replacement, contact DealerMAX support from the registered technical contact.

Sandbox dataset

The sandbox database is pre-populated with a small, deterministic dataset of fictitious Italian dealers and a Stripe test-mode customer that represents the partner organization. The dataset is intended to be enough to exercise the integration end-to-end, not to mirror the full production catalog.

Demo dealers

A handful of demo dealers are pre-populated, distributed across Italian regions to give the partner realistic shapes to work with. Each demo dealer has:

- A realistic but synthetic dealer name (for example, `Concessionaria Demo Lombardia S.r.l.`).
- A synthetic VAT number (P.IVA) reserved for documentation use only. These VAT numbers begin with the prefix `IT9999` and are not registered with the Italian revenue agency.
- A complete address in the relevant Italian province.
- A pre-configured set of NLT economic settings (mark-up percentage, three down-payment tiers each as `{percent_of_list, fixed_eur}`) that exercises a typical configuration.
- Access to a curated slice of the NLT catalog data — real network offers, anonymized and frozen in time so test assertions are stable.

The exact set of demo dealers is listed in the partner welcome email at sandbox provisioning time.

Demo Stripe customer

Each sandbox account is pre-populated with **one Stripe test-mode customer** representing the partner organization itself (not the partner's dealers — the partner pays DealerMAX, the dealers do not). This customer has:

- An active test-mode subscription on the partner's chosen tier.
- A test-mode card on file (Stripe's `4242 4242 4242 4242` test card; any future expiry; any 3-digit CVC).
- Sample invoices for the previous 3 billing cycles, marked paid.

This is enough to exercise the subscription-gating logic end-to-end (subscription active → API key works → simulate cancellation → API key returns 402 → restore → API key works again).

NLT catalog snapshot

The NLT catalog data in sandbox is a **frozen snapshot** of production catalog data, taken at sandbox initialization and refreshed periodically. This means:

- Specific car models, monthly canons, mileage tiers, and inclusion bundles in sandbox are stable across requests. A test that asserts a specific canon for a specific VW T-Cross configuration will not break the next day because the production catalog changed.
- The snapshot is internally consistent (every model referenced in a configuration is present in the catalog).
- The snapshot is **not** the live production catalog. It will lag production. Partners building features that depend on the latest catalog data should validate those features against production with a real partner key.

Worked example — end-to-end

This walkthrough provisions a sandbox dealer, configures its NLT economics, reads back the dealer-aware NLT listing, and tears down. It takes ~10 minutes to run from a clean terminal.

Prerequisites:

- A sandbox API key (see Getting sandbox access).
- `curl` and `jq` installed.
- Two environment variables exported:

```
export PMX_KEY="pmk_sand_..."
export PMX_HOST="https://api.dealermax.app"
```

Step 1 — provision a sandbox dealer

```
curl -X POST "$PMX_HOST/api/partner/dealers" \
  -H "X-Api-Key: $PMX_KEY" \
  -H "Content-Type: application/json" \
  -H "Idempotency-Key: walkthrough-step-1-$(date +%s)" \
  -d '{
    "external_dealer_id": "walkthrough-demo-001",
    "activate": true,
    "metadata": {
      "internal_dealer_code": "WALK-001"
    }
  }' | jq .
```

Expected response (abridged):

```
{
  "dealer_id": "walkthrough-demo-001",
  "partner_id": "ptn_7",
  "status": "active",
  "public_surfaces_enabled": false,
  "created_at": "2026-05-18T10:00:00Z",
  "updated_at": "2026-05-18T10:00:00Z",
  "created": true
}
```

`public_surfaces_enabled: false` here is expected: sandbox dealers are never indexed across the production AI surfaces.

Record the dealer ID for the next steps:

```
export DEALER_ID="walkthrough-demo-001"
```

Step 2 — configure NLT economics for the dealer

```
curl -X PATCH "$PMX_HOST/v1/dealers/$DEALER_ID/nlt-settings" \
  -H "X-Api-Key: $PMX_KEY" \
  -H "Content-Type: application/json" \
  -H "Idempotency-Key: walkthrough-step-2-$(date +%s)" \
  -d '{
    "agency_markup_percent": 3.5,
    "down_payment_tiers": {
      "low":    { "percent_of_list": 10.0, "fixed_eur": 0 },
      "medium": { "percent_of_list": 20.0, "fixed_eur": 0 },
      "high":   { "percent_of_list": 30.0, "fixed_eur": 0 }
    },
    "image_mode": "scenario_seasonal"
  }' | jq .
```

Expected response (abridged):

```
{
  "dealer_id": "walkthrough-demo-001",
  "agency_markup_percent": 3.5,
  "down_payment_tiers": {
    "low":    { "percent_of_list": 10.0, "fixed_eur": 0 },
    "medium": { "percent_of_list": 20.0, "fixed_eur": 0 },
    "high":   { "percent_of_list": 30.0, "fixed_eur": 0 }
  },
  "image_mode": "scenario_seasonal",
  "image_scenario_locked": null,
  "currency": "EUR",
  "effective_from": "2026-05-18T10:00:30Z"
}
```

The `dealer_id` field on the NLT-settings resource echoes the partner-owned external dealer id used in the path.

Step 3 — read back the dealer-aware NLT listing

```
curl "$PMX_HOST/v1/dealers/$DEALER_ID/nlt/offers?limit=3" \
  -H "X-Api-Key: $PMX_KEY" | jq '.data[0]'
```

The catalog returns the network offers already repriced with the dealer's 3.5% mark-up and the down-payment tiers from step 2. The listing shape carries `offer_id`, `slug`, `canonical_url`, `brand`, `model`, `segment`, up to 18 priced quotation cells (duration x km/year) and the three down-payment scenario amounts returned separately. See NLT Catalog for the full response schema and the pricing formula.

Step 4 — fetch a single offer in detail

Pick an `offer_id` from the listing response and fetch its detail view:

```
export OFFER_ID="..."  # take from step 3
curl "$PMX_HOST/v1/dealers/$DEALER_ID/nlt/offers/$OFFER_ID" \
  -H "X-Api-Key: $PMX_KEY" | jq .
```

The detail response is dealer-aware: `down_payment_scenarios_eur` reflects the tiers configured in step 2, the rendered image follows `image_mode`, and the gallery / technical_details / faqs / network_offers / available_addons fields are populated from the same resolvers that back the public dealer site.

Step 5 — clean up (optional)

The dealer provisioned in step 1 will be wiped automatically when the sandbox 30-day retention window rolls over, but you can wind it down immediately. Two options, both audit-logged:

- **Deactivate** (reversible) — `PATCH /v1/dealers/{id}` with `{"status": "inactive"}`. The dealer drops off the AI surfaces but can be re-activated later via the same endpoint.
- **Soft-delete** (terminal from the partner's side) — `DELETE /v1/dealers/{id}`. Sets `dealer_disabled=true` on the underlying user record so future `PATCH` calls return `409 dealer_deleted`. Audit history is preserved; restoration requires DealerMAX support.

```
# Reversible deactivation:
curl -X PATCH "$PMX_HOST/v1/dealers/$DEALER_ID" \
  -H "X-Api-Key: $PMX_KEY" \
  -H "Content-Type: application/json" \
  -d '{"status": "inactive"}'

# Or a terminal soft-delete:
curl -X DELETE "$PMX_HOST/v1/dealers/$DEALER_ID" \
  -H "X-Api-Key: $PMX_KEY"
```

Test-mode billing

The sandbox runs against a separate billing account in test mode. The partner's recurring subscription is simulated, no money moves, and the test card flow follows the standard documented at `docs.stripe.com/testing`. The most common card used by demo sandbox customers is `4242 4242 4242 4242`; the rest of the test surface (declined charges, 3DS challenges, renewal failures) is available through Stripe's published catalogue. Any future expiry date and any 3-digit CVC are accepted.

This is the only direct, partner-facing reference to Stripe in the documentation. Everywhere else, "subscription" refers to your DealerMAX commercial agreement; how the platform collects payment is implementation detail.

Promoting from sandbox to production

Promotion is intentionally low-friction: no code change is required beyond swapping the API key.

What changes

Setting	Sandbox	Production
API key	`pmk_sand_*`	`pmk_live_*`
Outbound webhook settings	Not applicable in v1	Not applicable in v1

The base URL **does not change** — both environments are served from `https://api.dealermax.app`. The environment band is encoded in the key prefix, so swapping `PARTNERMAX_API_KEY` is the only required action.

If the partner reads these values from environment variables, promotion is a deployment of new environment values with no application code change.

What does not change

- Request and response shapes.
- HTTP status codes.
- Error subcodes (RFC 7807 Problem Details, see Error Handling).
- Rate-limit headers (the absolute values may differ if the production tier exceeds the sandbox allocation, but the header format is identical).
- Idempotency semantics.

Production cutover checklist

Before flipping production traffic on, work through this checklist:

1. **Verify the production API key works.** Make a single read call (`GET /v1/dealers?limit=1`) and confirm `200 OK`. On a fresh partner account this returns an empty list — that is expected, since production data starts empty.
2. **Verify polling jobs and monitoring in production mode.** Scheduled syncs should handle cursor pagination, `Retry-After`, and transient 5xx retries before real dealer data is provisioned.
3. **Verify the partner's Stripe subscription is active.** Make any API call; ensure no `402 Payment Required` response. If you see 402, the subscription is not yet attached to the partner account.
4. **Verify monitoring is wired to production.** Logs, metrics, and alerting in the partner's observability stack should filter on the production environment, not sandbox.
5. **Verify the on-call rotation is staffed for the cutover window.** First 48 hours after cutover are where most integration issues surface.
6. **Verify rollback plan.** If production behavior diverges from sandbox in an unexpected way, the rollback is "swap `PARTNERMAX_API_KEY` back to the sandbox value" — but you need to know in advance which deployment knob does that.
7. **Provision the first real dealer.** This is the no-going-back step. The dealer becomes discoverable on production AI surfaces (MCP, Custom GPT, NLWeb) within minutes of provisioning.
8. **Verify discoverability.** After the indexing window, query the production MCP at `mcp.dealermax.app` for the new dealer and confirm it appears.

If any of steps 1-6 fail, do not proceed to step 7. Open a support ticket and resolve before cutting over.

Limitations

The sandbox has the following limitations relative to production. They are documented so integration test plans can account for them.

Outbound webhook delivery is not available in v1

The platform does not deliver outbound webhook events to partner-controlled URLs in v1. See Webhooks for the roadmap design notes. A v1 integration does not need a webhook handler: use scheduled polling for dealer references, stock, and NLT offer refreshes. If your team wants to prototype future event-driven code, keep that prototype outside the production v1 dependency path.

Reduced support response time

Sandbox issues fall outside the production SLA. Partner-tier support response times (documented in SLA) apply to production. Sandbox-only issues are handled on a best-effort basis, typically within 2-3 business days regardless of partner tier. If a sandbox issue blocks a production cutover, flag it as a cutover blocker and the response time accelerates to match the production-tier SLA.

No uptime SLA

The sandbox does not carry an uptime guarantee. It may be unavailable during deployments, infrastructure changes, or capacity rebalancing operations. In practice the sandbox is up >99% of the time, but partners building integration tests should retry on transient failures and not treat sandbox unavailability as a test failure.

Data wipe at 30 days

Sandbox data older than 30 days may be wiped without notice. This applies to dealers, NLT configurations, and any other state created via the API. The pre-populated demo dataset is reseeded on every wipe, so it is always present.

If a partner's CI runs nightly, the partner should treat each CI run as a clean-slate test: provision the dealers it needs, run the assertions, then either deactivate them or accept the wipe. CI code that asserts on dealers created days ago will become flaky.

No cross-network indexing

This is a feature, not a defect, but it is worth restating: sandbox dealers do not appear in the MCP server, the ChatGPT Custom GPT, or the NLWeb `/ask` endpoint. If part of the integration test plan is "verify the dealer appears in the AI surfaces," that step must run against production with a real dealer.

Rate limits are budget, not capacity

A partner with a 1000 req/min limit can send up to 1000 req/min against sandbox and the platform will honour it under normal load. However, sandbox is sized for integration testing, not for production-scale load testing. Sustained heavy load on sandbox may be rate-limited beyond the documented limit to protect other partners using the same environment. For genuine load testing against production-equivalent capacity, contact partnermax support to arrange a dedicated load-test window.

Related documents

- 02 — Getting Started — first sandbox call, in ~5 minutes
- 03 — Authentication — API key format, prefix bands, and rotation
- 04 — API Reference — full endpoint reference
- 06 — Error Handling — Problem Details schema, including `402 subscription_inactive`
- 05 — Webhooks — v1 polling guidance and future design notes
- 08 — Versioning and Deprecation — how to test deprecated endpoints in sandbox
- 10 — SLA — production SLO, which sandbox is explicitly excluded from
- 12 — Onboarding Journey — where the sandbox fits in the full onboarding flow

CHAPTER 16

SLA and Support

This document defines the operational commitments Partnermax makes to its partners: how available the API is, how fast it responds, what counts (and does not count) toward those numbers, where to look when something is wrong, what support response time to expect for each commercial tier, and how maintenance is announced.

The numbers in this document govern the production API surface — every request authenticated with a `pmk_live_*` key against `https://api.dealermax.app`. Sandbox traffic (requests authenticated with `pmk_sand_*` keys against the same host) is explicitly excluded — see Sandbox for sandbox-specific characteristics.

Uptime SLO

The Partnermax production API targets **99.9% monthly availability** for the `/v1/` surface.

Definitions

- **Availability** is measured at the production load balancer, on requests that reach Partnermax. A request is **successful** if the load balancer receives a response from the application with a status code in the 2xx, 3xx, or 4xx range (4xx counts as success because it reflects correct API behavior — for example, returning `404 Not Found` for a missing resource is the API working as intended).
- A request is **unsuccessful** if the application returns 5xx, fails to respond within the load balancer's gateway timeout, or the load balancer itself is unreachable.
- **Monthly availability** = (successful requests in the calendar month) / (total requests reaching the load balancer in the calendar month), expressed as a percentage.
- The calendar month is the UTC calendar month.

What 99.9% means in absolute terms

Time window	Allowed downtime at 99.9%
Day	1 minute 26 seconds
Week	10 minutes 4 seconds
Month (30 days)	43 minutes 12 seconds
Year	8 hours 45 minutes

Partnermax tracks actual uptime per calendar month and publishes the historical record on the status page.

Measurement methodology

The reported uptime is computed from load balancer access logs, not from external blackbox probes. This is the same data used to compute SLO compliance internally and is the same data used to determine service credit eligibility (see Service Credits Policy).

External monitoring tools (the partner's own uptime monitor, third-party services like Pingdom or Datadog Synthetics) may report different numbers because they include the partner's own network path. Partnermax cannot warrant uptime over a network path it does not control. If a partner's monitor sees a >0.1% delta from Partnermax's reported uptime, the next step is to compare against the Status Page for any incidents in the same window, then escalate to support if the discrepancy persists.

Latency SLO

Latency targets are p95 server-side, measured at the application:

Endpoint class	p95 target	Notes
Read endpoints (`GET`)	< 500 ms	Excludes upstream billing-provider latency on subscription-state checks
Mutating endpoints (`POST`, `PUT`, `PATCH`, `DELETE`)	< 1000 ms	Excludes billing-provider propagation lag

What is and is not counted

The server-side latency timer starts when the request is received by the application layer and stops when the application begins writing the response body. It excludes:

- Time spent in the partner's network reaching the Partnermax load balancer.
- Time spent in the load balancer queue under back-pressure (this is captured separately and surfaced as load-balancer p95, with its own dashboards).
- Time spent in outbound calls to Stripe that the partner triggers indirectly. For example, when a partner creates a new dealer and the platform must verify the partner's DealerMAX subscription, the round-trip to Stripe is excluded from the API latency SLO because Stripe's latency is outside Partnermax's control.

Per-region considerations

The Partnermax API is deployed in EU (Frankfurt). Partners with infrastructure in the EU see end-to-end latency close to the p95 targets above plus their own network RTT. Partners with infrastructure in North America see an additional ~80-110 ms RTT to Frankfurt. This network component is outside Partnermax's measurement scope. Multi-region deployment is on the roadmap; see developers.dealermax.app/docs/roadmap for current status.

What Does Not Count Toward Uptime

The following are explicitly excluded from the uptime SLO calculation:

1. **Sandbox environment.** Sandbox traffic (any request authenticated with a `pmk_sand_*` key) is best-effort and carries no SLO, even though it is served from the same host as production.
2. **Scheduled maintenance.** Maintenance windows announced at least 7 days in advance via the status page and email are excluded. See Maintenance Windows Policy.
3. **Force majeure.** Outages caused by events outside Partnermax's reasonable control: natural disasters, large-scale internet outages, government action, cyberattacks at the upstream cloud provider level, war. The status page will document the nature of the event and the recovery timeline.
4. **Partner-side network issues.** If the partner's network cannot reach the Partnermax load balancer, the Partnermax service is not impaired. The status page is the source of truth for whether an incident is on Partnermax's side.
5. **Upstream provider outages cascading through Partnermax.** Specifically: - billing-provider outages causing subscription-gating logic to fail open or fail closed in unexpected ways. Stripe publishes their own status at status.stripe.com; a Partnermax incident caused by a confirmed billing-provider outage is reported on Partnermax's status page but does not count against the SLO. - Supabase / PostgreSQL outages at the managed-database level are documented and reported transparently, but for SLO accounting purposes the Partnermax team treats these as Partnermax-side incidents (since the data plane is part of the Partnermax service surface). This is more conservative than the strict letter of the exclusion clause and is intentional.
6. **Partner abuse triggering rate limits.** A partner exceeding their rate limit and receiving `429 Too Many Requests` is the API behaving correctly, not an outage. Rate-limit responses are 4xx and count as successful for uptime purposes.

Status Page

The status page is the source of truth for current and historical service health:

```
https://developers.dealermax.app/status
```

The partner-facing status page is exposed through `https://developers.dealermax.app/status` and backed by independent health checks. Real-time readiness for the developer portal is exposed at `https://developers.dealermax.app/health/ready`. Infrastructure-wide status is also published on the separate DealerMAX status surface at `https://status.dealermax.app`.

The Partnermax developer platform and the wider DealerMAX network platform are tracked as separate service surfaces with separate incident histories.

Components tracked separately

The status page breaks the service into independent components, each with its own status indicator:

Component	What it covers
API (`/v1/`)	The production HTTP API surface
Database	The primary Supabase/PostgreSQL data plane
Stripe integration	Subscription state checks, webhook ingestion
NLT catalog data plane	Read access to the dealer-aware NLT catalog
Documentation site	The developer documentation portal

A degradation in one component does not necessarily mean the others are affected. For example, the documentation site being down does not affect API availability — partner integrations continue to work, only access to the developer portal is impacted.

Incident reports

Every incident, regardless of severity, gets a status page entry. Sev 1 and Sev 2 incidents receive a post-mortem published within 7 business days of resolution. Post-mortems include:

- Timeline of detection, mitigation, and resolution.
- Root cause analysis at a level of detail appropriate to partner audiences (not internal-only).
- Action items the Partnermax team has committed to, with owners and target dates.
- Whether the incident counts against the uptime SLO (per the exclusions above).

Subscriptions

Partners can subscribe to status page updates by:

- Email — to any number of distribution addresses per partner.
- RSS — `developers.dealermax.app/status/rss`.
- Webhook — push notifications to a partner-controlled URL whenever a component status changes.
- Slack — auto-post to a channel (configured via the status page's own Slack integration).
- SMS / phone — supported for Enterprise tier on the technical contact's number.

Status-page webhook notifications are distinct from PartnerMAX API event webhooks. PartnerMAX application event webhooks are not available in v1; use polling for dealer, vehicle, and NLT synchronization.

Historical uptime

The status page exposes monthly uptime percentages per component for the trailing 90 days, and a yearly summary. This is the same data used to determine service credit eligibility, so partners can independently verify SLO compliance without a support ticket.

Incident Severity Matrix

Severity	Definition	Examples
Sev 1	Complete outage of the `/v1/` API surface, or data loss event affecting any partner	API returning 5xx on all requests; database unreachable; Stripe webhook secret compromise
Sev 2	Partial outage affecting >10% of partners, or critical feature unavailable to a single major partner	One database read replica down causing intermittent failures; Stripe integration broken affecting all subscription checks; a single Enterprise partner unable to provision dealers
Sev 3	Degraded performance not meeting SLO, or non-critical feature unavailable	p95 latency > 2x target; documentation site slow but functional; one non-critical background sync delayed >10 minutes
Sev 4	Cosmetic issue, documentation defect, or minor UX bug	Typo in API response message; documentation example with wrong syntax; status page color slightly off

Severity is assigned by the Partnermax on-call engineer at incident detection and may be re-classified during the incident as scope becomes clearer. A Sev 3 that turns out to affect more partners gets promoted to Sev 2; a Sev 2 whose impact narrows gets demoted to Sev 3. The final post-mortem reflects the highest severity reached during the incident.

Severity assignment rubric

The decision tree the on-call engineer uses to assign severity at detection time:

1. **Is any production data lost, corrupted, or visible to the wrong tenant?** If yes, Sev 1 regardless of other factors. Data loss and tenant-isolation breaches are the most consequential failure modes and bypass the impact-based classification.
2. **What fraction of production API traffic is returning 5xx?** If >50%, Sev 1. If 10-50%, Sev 2. If <10% but a specific partner is fully affected, Sev 2. If <10% with no identifiable concentrated impact, Sev 3.
3. **Is the failure customer-perceived?** A failure caught by internal monitoring but never reaching a partner request (for example, a failing background job that has a retry buffer) is typically Sev 3 until the buffer is exhausted, at which point it becomes Sev 2.
4. **Is recovery automatic?** A self-healing incident (load balancer drained an unhealthy node and traffic recovered within 30 seconds) is logged as a Sev 3 incident even if the peak impact was higher, because the partner-visible duration is short.

The rubric is applied at detection and then re-applied at each major milestone of the incident (mitigation, partial restore, full restore). Re-classifications are logged in the incident timeline.

Detection sources

Incidents are detected through three independent paths:

- **Internal monitoring.** Synthetic probes from inside the Partnermax infrastructure exercise the canonical API endpoints every 30 seconds. SLO burn-rate alerts on the multi-window multi-burn-rate scheme fire to PagerDuty when error rate or latency deviate.
- **External monitoring.** Independent uptime monitoring from outside the Partnermax network (geographically distributed probes) cross-checks the internal monitoring. Disagreements between internal and external probes trigger investigation regardless of severity.
- **Partner reports.** A support ticket flagged as “incident” from any partner is treated as a detection event. The on-call engineer triages within the tier-appropriate response window and either reclassifies as a non-incident or opens a status page entry.

A confirmed incident is one that any of the three detection paths has identified and that the on-call engineer has acknowledged. The status page is updated within 5 minutes of confirmation for Sev 1 and Sev 2, and within 15 minutes for Sev 3.

Support Tiers

Support is provided in four commercial tiers. Higher tiers carry faster response time commitments and richer communication channels.

Tier summary

Tier	Primary channel	Slack	Dedicated contact	Uptime SLA
Free Trial	Community Slack workspace	Community channel only	No	None
Starter	Email (`support@dealermax.app`)	Community Slack	No	None
Pro	Slack (shared channel)	Dedicated channel	No	None
Enterprise	Dedicated Slack + named contact	Dedicated channel	Yes	99.9% monthly

The Free Trial tier is intended for evaluation only. Starter is the default paid tier and is appropriate for partners running steady-state low-volume integrations. Pro is appropriate for partners building production-critical integrations that need faster turnaround. Enterprise is for partners with named technical contact requirements, contractual SLA enforceability, and 1-hour critical incident response.

Response time matrix

The following matrix gives the maximum time Partnermax commits to **first response** for a support ticket, by tier and severity. First response means a human-acknowledged ticket with at least an initial assessment, not a resolved ticket.

	Sev 1	Sev 2	Sev 3	Sev 4
Free Trial	Best-effort, community-supported	Best-effort	Best-effort	Best-effort
Starter	4 business hours	1 business day	2 business days	5 business days
Pro	2 business hours	8 business hours	1 business day	3 business days
Enterprise	1 hour, 24/7	4 hours, 24/7	8 business hours	2 business days

Notes:

- “Business hours” means 09:00–18:00 Central European Time, Monday through Friday, excluding Italian public holidays.
- “24/7” means around the clock, including weekends and holidays.
- Enterprise tier Sev 1 response is staffed by the on-call rotation, with the named technical contact looped in within 4 hours of incident detection.
- Free Trial tier has no contractual response time. The community Slack workspace is monitored during business hours by Partnermax engineers on a best-effort basis.

Resolution targets

Response time is committed; resolution time is not. The complexity of an incident is highly variable and committing to a resolution time would either drive perverse incentives (escalation theater) or be honored only by giving up scope (partial fixes shipped to clock-meet). Instead, Partnermax commits to:

- Sev 1 and Sev 2: continuous active work from detection to mitigation, with status updates at least every 60 minutes for Sev 1 and every 4 hours for Sev 2.
- Sev 3 and Sev 4: timely fix on a normal release cadence (typically next sprint for Sev 3, when convenient for Sev 4).

Communication Channels per Tier

Free Trial

- **Community Slack workspace.** Public channel `#partnermax-community`, monitored by engineers during business hours.
- **Documentation portal.** Self-serve answers via developers.dealermax.app/docs.
- **No email support.** Tickets sent to `support@dealermax.app` from Free Trial accounts are auto-replied with a pointer to the community Slack.

Starter

- **Email.** `support@dealermax.app` is the primary channel. Replies come from a named engineer, not a generic alias.
- **Community Slack.** Available as a secondary channel but not staffed with SLA.
- **No dedicated channel.** Tickets are tracked in the support system with ticket IDs but not in a Slack channel shared with the partner.

Pro

- **Dedicated Slack channel.** Each Pro partner gets a shared Slack channel with Partnermax engineers (`#partner-<partner-slug>`). Used for support tickets, change announcements, deprecation notices, and general questions.
- **Email.** Tickets can also be opened by email; they appear in the same dedicated channel.
- **No named contact.** The Slack channel is staffed by the rotation, not a single individual.

Enterprise

- **Dedicated Slack channel.** Same as Pro, plus the named technical contact is a member.
- **Named technical contact.** A specific Partnermax engineer assigned to the account. Acts as the partner's first point of escalation for major issues and the partner's voice into Partnermax engineering for feature requests.
- **Quarterly roadmap calls.** A 60-minute video call once per quarter with the named technical contact and a product representative. Used to surface upcoming Partnermax roadmap items and to capture the partner's feature requests.
- **Monthly business review.** A 30-minute call with the named technical contact covering usage statistics, incident summary if any, and operational health.
- **Direct phone line for Sev 1.** A phone number that bypasses the support queue for confirmed Sev 1 incidents.

Maintenance Windows Policy

Scheduled maintenance is announced at least **7 days in advance** through:

- A scheduled-maintenance entry on the status page.
- An email to all partner technical contacts.
- A post in the dedicated Slack channel for Pro and Enterprise tiers.

Typical maintenance window

Routine scheduled maintenance runs on **Sunday 02:00–04:00 CET**. This window is chosen to minimize impact on the Italian and European partner base, which represents the bulk of API traffic during weekday business hours. North American partners should be aware that this is Saturday evening in their time zones; status page subscribers receive a notification before each window opens.

Not every Sunday has a maintenance window — most do not. When there is no maintenance scheduled, no Sunday-window announcement is sent.

Impact during a maintenance window

The typical maintenance window does not take the API offline. Most maintenance is performed by deploying new application versions behind the load balancer with a rolling restart that keeps the API surface continuously available. Partners should observe normal API behavior during the window.

When a maintenance operation does require downtime (rare, typically <2 times per year), the announcement clearly states the expected impact: “API will be unavailable for up to 10 minutes during the 02:00 CET window on Sunday DD MMM.”

Emergency maintenance

Emergency maintenance may occur with less than 7 days notice. This is reserved for situations that demand immediate action:

- Critical security patches (CVE responses, dependency vulnerabilities with active exploitation).
- Capacity emergencies (unexpected traffic spike requiring infrastructure rebalancing).
- Data integrity issues requiring an immediate database operation.

Emergency maintenance is communicated as it happens via the status page, email, and Slack channels. The partner team will provide the best available estimate of impact and duration at the time of communication, with updates as the situation evolves.

Emergency maintenance is **not** excluded from the uptime SLO. If emergency maintenance causes downtime, that downtime counts against the monthly availability number.

Change-window approval for high-impact maintenance

Maintenance operations that the engineering team classifies as high-impact (database schema migrations on hot tables, load balancer reconfiguration, certificate rotation) require an additional internal approval beyond normal scheduling. The approval gate enforces:

- A documented rollback plan with a tested rollback procedure.
- A pre-maintenance health snapshot to compare against post-maintenance.
- A communication plan covering status page, email, and Slack with prepared draft text.
- An on-call engineer assigned as change owner for the duration of the window.

High-impact maintenance windows are announced to Enterprise tier partners 14 days in advance instead of the standard 7, with an explicit description of the change being made and the expected user-visible impact.

Partner-requested maintenance freezes

Enterprise tier partners can request maintenance freezes around partner-side events (the partner’s own product launches, partner-side migration windows, end-of-quarter financial close periods). Freeze requests are honored on a best-effort basis and may be declined if they conflict with security-critical work. The request process is to message the named technical contact at least 21 days before the requested freeze with the start and end dates and the business reason.

Escalation Matrix for Enterprise Tier

Enterprise-tier partners have an explicit escalation path documented in their contract. Each escalation level adds Partnermax personnel; it does not replace the previous level.

Level 1: Named technical contact

The named technical contact is the first point of escalation for any issue the partner team wants to raise above the routine Slack channel. The named contact is reachable via:

- The dedicated Slack channel (DM if needed).
- A direct email address provided at contract signing.

- A direct phone number for Sev 1 incidents.

Target acknowledgement: within 1 hour during business hours, within 4 hours outside business hours.

Level 2: Engineering on-call

For Sev 1 and confirmed Sev 2 incidents, the named technical contact loops in the on-call engineering rotation. The on-call engineer is paged immediately and joins the incident bridge.

Target acknowledgement: within 15 minutes of being paged, 24/7.

Level 3: CTO

For Sev 1 incidents not mitigated within 4 hours, or for incidents with significant commercial impact (data loss, major reputation event, multi-partner cascading failure), the CTO is informed and takes ownership of the response coordination.

Target acknowledgement: within 1 hour of being engaged.

Level 4: CEO

Reserved for partner-relationship issues that have escalated past CTO involvement (typically contractual disputes, not technical issues). Engaged at the named technical contact's discretion in consultation with the partner's executive sponsor.

Service Credits Policy

Service credit eligibility, percentage of monthly fees credited, claim filing process, and maximum credit per month are commercial terms that vary by partner tier and by the specific terms of the partner's executed agreement. No public percentage table is published in v1; the signed agreement governs.

Indicative principles (subject to contractual confirmation):

- Service credits apply only to the production API surface, not to sandbox.
- Service credits are claimed by the partner within 30 days of the close of the affected calendar month, by emailing the named contact (Enterprise) or `support@dealermax.app` (Pro/Starter).
- Service credits are issued as a percentage credit against the next monthly invoice, not as a cash refund.
- The maximum credit in any single month is capped by the executed agreement.
- Excluded events (per What Does Not Count Toward Uptime) do not trigger service credits.

The final terms appear in the commercial agreement.

Related Documents

- [01 — Introduction] — what Partnermax is and which problem it solves
- 05 — Errors — error subcode taxonomy, including 5xx incident-class responses
- 08 — Versioning and Deprecation — stability guarantees within `/v1/`
- 09 — Sandbox — what the sandbox is and why it is excluded from this SLA
- 12 — Onboarding Journey — where SLA awareness fits in the partner onboarding flow
- Status page — live and historical service health
- Stripe status — upstream dependency status

CHAPTER 17

Security and Compliance

Executive summary

partnermax is operated by Azure S.r.l., an Italian limited-liability company headquartered in the European Union. All customer data is stored in the EU (Frankfurt region, AWS `eu-central-1` underneath Supabase). Data in transit is protected with TLS 1.3; data at rest is encrypted with AES-256 via Supabase-managed keys. Every API request is authenticated with a partner-scoped key that is hashed at rest with SHA-256, and every mutating operation is recorded in an immutable audit log. The platform relies on a small set of EU- and US-based sub-processors, each holding SOC 2 Type 2 or equivalent attestation; the complete list is published below. A GDPR-compliant Data Processing Agreement is available on request, and DealerMAX serves as data controller for the network operations layer while you, the partner, are data controller for the dealer records you provision.

This document describes the controls in detail and the shared responsibility model between DealerMAX, partners, and downstream dealers. For questions not answered here, contact `support@dealermax.app` (PGP key available; see Responsible disclosure).

Data residency

Where your data lives

All persistent storage — Postgres databases, file objects in storage buckets, audit log archives, and backup snapshots — resides in Frankfurt, Germany, in the AWS `eu-central-1` region. Supabase's managed Postgres clusters are pinned to this region with no automatic geographic failover outside of the EU. Disaster-recovery replicas, when configured, are placed in a secondary EU availability zone within the same region.

Compute (the FastAPI services that back the API) runs on Railway infrastructure, which we host out of an EU-based region for partnermax traffic. CDN and edge layers (Cloudflare) serve cached static assets from globally distributed points-of-presence; the origin and all dynamic responses are EU-resident.

What does not leave the EU

Personal data and customer business data (dealer records, API audit logs, vehicle catalog metadata associated with a dealer) are processed only in the EU. Synthetic logs, anonymized error fingerprints, and platform telemetry that contains no personal data may be processed by US-based observability sub-processors (Sentry); see the Sub-processor list.

LLM subprocessor boundary

DealerMAX does not send customer personal data, API keys, raw request headers, audit-log bodies, partner billing records, dealer contact emails or phone numbers, internal notes, plates, or VINs to LLM subprocessors. LLM enrichment payloads are restricted to vehicle/catalogue attributes and editorial metadata needed for grounded summaries or stock copy. Where OpenAI is used, processing is governed by a zero-retention DPA; EU-only LLM processing is treated as a contract-specific control and is not implied by the default PartnerMAX v1 contract.

CLOUD Act considerations

The US CLOUD Act allows US authorities to compel US-headquartered providers to produce customer data regardless of where it is stored. Customers operating under European data-protection scrutiny have repeatedly raised this as a concern.

Azure S.r.l. is an Italian company subject to Italian and EU jurisdiction. Customer business data is held in infrastructure within the EU operated under contracts governed by EU law. Where a sub-processor is US-headquartered (Supabase, Stripe, Postmark, Cloudflare, Sentry, OpenAI), we hold a Data Processing Agreement with that sub-processor that incorporates Standard Contractual Clauses (SCCs) and supplementary measures (encryption at rest with provider-managed keys, transit encryption, and operational access logging) consistent with EDPB Recommendations 01/2020.

We respond to law enforcement requests in accordance with applicable law and our published guidelines. We do not voluntarily disclose customer data and we challenge legal process that we believe to be over-broad or improper. Where legally permissible, we will notify affected customers before disclosure.

Encryption

In transit

partnermax requires TLS 1.3 on all customer-facing endpoints. Cipher suites are limited to those with forward secrecy (ECDHE key exchange) and AEAD constructions (AES-GCM and CHACHA20-POLY1305). Plaintext HTTP is unconditionally redirected to HTTPS with a `Strict-Transport-Security` header carrying a 12-month `max-age` and `includeSubDomains`. The `developers.dealermax.app` domain is preloaded into the HSTS preload list.

Certificate issuance is automated via Let's Encrypt and Cloudflare, with rotation roughly every 90 days. Certificate Authority Authorization (CAA) records on the apex domain restrict issuance to those two CAs.

At rest

Database storage uses AES-256 disk encryption with keys managed by Supabase. Storage buckets use AES-256 server-side encryption (S3-compatible). Database backup snapshots inherit encryption from the source volume. Long-term audit log archives are written to encrypted object storage with object-level encryption enabled.

API keys and OAuth tokens are stored hashed or encrypted according to their use case; plaintext secrets are never written to logs. Partner API keys use SHA-256 without a salt so every internal service can resolve the same key row consistently. The plaintext value of a credential is returned to you exactly once at creation time and never again.

Key management

Encryption keys for application-layer secrets (database credentials, third-party API tokens, signing keys for internal services) are managed via Supabase Vault. Access to the Vault is restricted to the platform engineering on-call rotation and is logged.

Customer-supplied encryption keys (BYOK) are not currently supported. For most enterprise integrations, the combination of provider-managed encryption and our shared-responsibility model is sufficient; if your security review requires BYOK, raise it with your account manager so we can scope the work.

Authentication and authorization

API authentication

Partner API keys use the shape `pmk_{env}_{base64url}`. Current `live`, `sand`, and `test` keys are 41 characters: a 9-character environment prefix plus a 32-character random body produced from 24 random bytes (`secrets.token_urlsafe(24)`, ≈190 bits of entropy). The plaintext key is shown once at creation and never persisted in plaintext anywhere in our infrastructure. On creation, the platform computes `sha256(key_plaintext)` (no salt) and stores the hex digest alongside the first 12 characters of the full plaintext in `dealer_public_api_keys` as `key_prefix`. On request validation, the same hash is recomputed from the incoming `X-Api-Key` or `Authorization: Bearer ...` header and compared in constant time against the stored hash.

The salt-free scheme is a deliberate platform-level decision: it allows the same key row to be resolved consistently by the internal services that authenticate partner traffic without each service maintaining its own per-deployment salt. Brute-force resistance comes from the random body's about 190 bits of entropy, not from per-key salting; an attacker who exfiltrates the keys table still cannot reverse a hash within any meaningful timeframe.

This means a database dump of the API keys table cannot be used to authenticate. An attacker who steals the database obtains hex hashes and 12-character prefixes (used for log correlation and rotation), not authentication material.

Plain-text key material is never logged. Our access log redacts the `X-Api-Key` and `Authorization` headers at the edge before any log line is written. The `GET /v1/keys` endpoint surfaces keys by prefix only after creation; the full secret is never retrievable after issuance.

Capability model

API keys carry capabilities. In v1 the only named capability is `can_issue_keys`, which guards controlled key-lifecycle operations. Dealer, NLT-settings, and NLT-catalog endpoints are scoped at row level by partner / dealer ownership rather than by a separate capability bit. Replacement keys keep the same capability set, so a key can never widen its own scope.

The initial key delivered at onboarding carries `can_issue_keys`, but PartnerMAX v1 still allows only **one active API key per partner account per environment**. Routine rotation, emergency replacement, and deactivation are handled with DealerMAX support.

Authorization model

Every API operation is scoped to a partner account. The platform extracts the `partner_id` from the authenticated key and enforces a server-side check that the target resource (dealer, NLT settings, audit log) belongs to that partner. There is no shared-tenant API surface — a partner cannot read, update, or even probe for the existence of another partner's resources. A request that would return another partner's data instead returns `404 Not Found`.

The same access check runs at the database layer as a defense-in-depth measure: the queries that back read endpoints filter by `partner_id` even when the row identifier alone would technically be sufficient, so that an application-layer logic bug cannot leak cross-tenant data.

Audit logging

Every mutating API call is logged with the following fields, immediately, before the operation returns to the caller:

Field	Description
`timestamp_utc`	ISO-8601 timestamp with millisecond precision.
`partner_id`	The authenticated partner account.
`dealer_id`	If the operation targeted a specific dealer, its identifier.
`endpoint`	Method and path, normalized (path parameters substituted with `{id}`).
`request_id`	The `X-Request-Id` returned to the caller.
`trace_id`	The W3C `traceparent` span identifier propagated through the request.
`source_ip`	Originating IP at our edge (after Cloudflare-set headers are validated).
`user_agent`	The `User-Agent` from the request.
`response_status`	The HTTP status returned.
`key_prefix`	The 12-character prefix of the API key used (never the full key).

Audit log records are append-only. Partners may request a CSV or JSONL export of their account's audit log for any date range by emailing `support@dealermax.app`; exports are provided within five business days at no charge. The log is retained for 12 months minimum, longer for records that pertain to ongoing incidents or legal hold.

Read-only operations are not written to the application audit log to keep the log readable for security analysis. Aggregate read metrics (per-endpoint request counts per partner per hour) are recorded separately for billing and capacity planning.

Sub-processor list

The following third parties process customer data on our behalf. We have a Data Processing Agreement in place with each one. We commit to providing 30 days' notice via email and the partnermax status page before adding a new sub-processor.

Name	Purpose	Data categories processed	Location	Certifications / compliance	DPA
Supabase Inc.	Managed Postgres database, object storage, authentication primitives	Dealer records, partner records, hashed API keys, audit log records, vehicle catalog metadata	USA (corporate); EU (`eu-central-1`, Frankfurt, for our deployment)	SOC 2 Type 2	supabase.com/dpa
Stripe Inc.	Subscription billing for partner accounts, payment processing	Partner billing contact, payment method tokens (Stripe-managed, not stored by us), subscription state	USA	PCI-DSS Level 1, SOC 2 Type 2	stripe.com/legal/dpa
Postmark / Wildbit	Transactional email delivery (provisioning notifications, billing receipts, operational alerts)	Recipient email, message contents (rendered from templates), delivery status	USA, EU	SOC 2 Type 2	postmarkapp.com/dpa
Railway Corp	Compute infrastructure for API services	All in-transit request and response data (ephemeral; not persisted by Railway)	EU region for partnermax workloads	SOC 2	Available on request
Cloudflare Inc.	CDN, DDoS protection, DNS, edge TLS termination	Request metadata at edge (source IP, headers, paths), cached static assets	Global (PoPs); EU-resident edge for European traffic	ISO 27001, SOC 2	cloudflare.com/cloudflare-customer-dpa/
Sentry / Functional Software Inc.	Error tracking and exception aggregation	Stack traces, request paths (URL only), request IDs, error fingerprints — no request bodies, no personal data	USA, EU (we use the EU data residency option)	SOC 2	sentry.io/legal/dpa/
OpenAI Global LLC	LLM inference for the NLWeb `/ask` summarize mode and select internal features	Natural-language queries about catalog data; never raw dealer or partner records	USA	DPA with zero-retention API enabled (data not used for training, not retained beyond request processing)	Available on request

The same list, in machine-readable form, is published at `https://developers.dealermax.app/.well-known/subprocessors.json` and is the authoritative source. The version above is a human-readable snapshot.

If you require notification of sub-processor changes via RSS rather than email, subscribe to `https://developers.dealermax.app/feed/subprocessors.atom`.

GDPR posture

Controller-processor relationships

The relationships are layered:

```
graph TD
    A[Data subject<br/>e.g. end consumer browsing a dealer site] -->|provides personal data| B[Dealer]
    B -->|controller of consumer data| C[Partner]
    C -->|controller of dealer business data| D[DealerMAX]
    D -->|processor for partner's dealer business data| D
    D -->|controller of network operations data<br/>audit logs, billing, telemetry| D
```

- **Azure S.r.l.** is the controller for network operations data: the audit log of API calls, billing records of partner subscriptions, security telemetry, and the inferred catalog metadata that DealerMAX surfaces across the network's AI surfaces. We are processor for the dealer business data that you, as partner, provision through the API.
- **You, the partner**, are the controller of dealer business data you provision. You determine which dealers to onboard, what brand and contact information they expose, what NLT economic configuration they operate under, and on what basis their data is processed.
- **Each downstream dealer** is the controller of any consumer personal data it collects (lead forms, account registrations, transactional records). DealerMAX never directly collects consumer personal data through the partnermax API surface.

DPA template

A standard Data Processing Agreement is available for execution by all partners. The template incorporates the European Commission's 2021 Standard Contractual Clauses (Module Two: controller-to-processor) for any data transfer to a sub-processor outside the EU/EEA, and lists the sub-processors above with the required transfer safeguards.

Request the DPA by emailing `support@dealermax.app` from your registered business contact. Turnaround is typically two business days for execution of the standard form, longer for negotiated amendments. The DPA becomes effective on countersignature and is incorporated by reference into your subscription agreement.

Default retention policies

Data category	Retention	Notes
Dealer records (active)	For the life of the partner account	Deleted on partner subscription cancellation per cascade-deactivation policy.
Dealer records (deactivated)	90 days from deactivation	Hard-deleted thereafter unless legal hold applies.
Audit log	12 months minimum	Longer if required for security investigation or legal hold. Partner may request export at any time.
Billing records	7 years	Required by Italian tax law (DPR 600/1973).
Backup snapshots	30 days rolling	Encrypted; restore-only access.

Data subject rights

Under GDPR, individuals whose personal data is processed have specific enumerated rights. partnermax supports the exercise of those rights as follows.

Right of access

A data subject (or their authorized representative) may request a copy of personal data we hold about them. Requests are accepted at `support@dealermax.app` with reasonable identity verification (typically a confirmation reply from the email address on record). Response is provided within 30 days as required by GDPR Art. 12(3), extendable by two further months for complex cases with notice within the first month.

Right of rectification

Inaccurate or incomplete personal data may be corrected on request. For partner-account-level records (your billing contact, technical contact, etc.), email `support@dealermax.app`. For records you have provisioned about your dealers, use the API. For records held by DealerMAX in the network operations layer (audit log entries referring to a data subject), contact support.

Right of erasure

A data subject may request erasure of personal data held about them. Where we are the controller (network operations data), we honor erasure requests subject to retention obligations and the legal-hold exception. Where you are the controller (dealer records you provision), we will forward the request to your registered technical contact and you will handle the erasure under your own processes; we will erase the upstream records on your instruction.

Audit log entries are not erased on data-subject request; they constitute records of processing and are retained per GDPR Art. 30. Audit log entries can be redacted of personal data (replacing the name with a stable pseudonym) on request when the underlying processing has ceased.

Right of portability

Where data is processed on the basis of consent or contract, the data subject may request export in a structured, commonly-used, machine-readable format. partnermax exports dealer business data as JSON-LD or CSV on request via the API or support channel.

Response process

Step	Owner	SLA
Request received at `support@dealermax.app`	DealerMAX support	Same business day acknowledgment
Identity verification	DealerMAX support	1–5 business days
Forwarding to partner (where partner is controller)	DealerMAX support	1 business day from verification
Production of records or completion of action	DealerMAX engineering	14 business days from verification
Final response to data subject	DealerMAX support	30 calendar days from receipt, per GDPR Art. 12(3)

Audit logs

The audit log captures every mutating API call. The schema and retention policy are described under Audit logging above. This section covers operational properties relevant to your own security review.

Tamper resistance

Audit log records are written to an append-only table with no `UPDATE` or `DELETE` privileges granted to the application service account. The table is replicated to a separate archive bucket on a 24-hour cadence; the archive uses object-locking ("compliance mode" equivalent) so that records cannot be deleted before the 12-month retention window expires, even with full administrative access.

The audit log generation path itself is also instrumented: failures to write an audit record cause the originating request to fail closed (the operation is rolled back). This ensures that a mutating operation never succeeds without a corresponding log entry.

Export format

Audit log exports are provided as newline-delimited JSON (JSONL) compressed with gzip. The schema matches the Audit logging table above. Exports may be delivered via secure download link with one-time-use signed URL (default), via SFTP to a partner-provided destination (on request), or via an Amazon S3 bucket the partner controls (on request, with the partner sharing a presigned upload URL).

Partners that require continuous streaming export to a SIEM may request a custom integration through their account manager. This is presently a manual operations workflow; a self-serve streaming API is on the roadmap.

Internal access

Engineer access to the production audit log is restricted to the on-call rotation and is itself logged at the infrastructure layer (Supabase access log + database session log). Engineer queries against partner data are made under named individual accounts with no shared credentials, and are recorded for review.

Vulnerability management

Penetration testing cadence

We engage an independent third-party security firm for a full-stack penetration test on an annual cadence. The most recent test report is available under NDA on request via `support@dealermax.app`. Engagement scope includes API surface, authentication and capability enforcement, billing webhook ingestion, and infrastructure misconfiguration review. Findings are tracked to remediation with severity-weighted SLAs:

Severity	Remediation SLA
Critical	7 calendar days
High	30 calendar days
Medium	90 calendar days
Low / Informational	Best effort, prioritized in normal backlog

A redacted summary of each year's pentest results (findings count by severity, remediation status) is published in the Trust & Security center (URL documented forward; static page in preparation).

Continuous internal scanning

Source code is scanned by GitHub Advanced Security for known vulnerable dependencies (Dependabot), insecure code patterns (CodeQL SAST), and exposed secrets in commits. Build pipelines block merges on Critical or High findings until remediation or risk acceptance is recorded.

Dynamic scanning (DAST) is run against the staging environment on every release-candidate deployment, and against production weekly during a low-traffic window.

Security advisories

Vulnerability disclosures that affect partners are published at `https://developers.dealermax.app/.well-known/security-advisories.atom` (Atom feed). The same disclosures are also posted in the changelog and emailed to the registered technical contact for every active partner account.

The standard `security.txt` file is published at `https://developers.dealermax.app/.well-known/security.txt` per RFC 9116 with current contact, PGP key, and policy references.

Responsible disclosure

We welcome security research. To report a suspected vulnerability:

- Email `support@dealermax.app`. For reports containing exploit details or sensitive observations, request the current secure disclosure channel before sending payloads or secrets. Security contact metadata is published at `https://developers.dealermax.app/.well-known/security.txt`.
- Provide enough detail to reproduce. Steps to reproduce, request/response samples (with secrets redacted), and an assessment of impact are most helpful.
- Allow up to 5 business days for acknowledgment and up to 90 days for coordinated public disclosure. Where remediation requires longer (for instance, a coordinated rollout across many partners), we will explain the timeline and revisit it with you.

We do not currently operate a paid bug bounty program. We recognize researchers in our Hall of Recognition (URL documented forward; static page in preparation) with their consent and, for material findings, with a token of appreciation arranged on a case-by-case basis.

Out of scope:

- Denial-of-service testing against production. Conduct any rate-limit or load-testing research against the sandbox environment by authenticating with a `pmk_sand_*` key (same host as production, separate data plane — see Sandbox).
- Social engineering of staff, customers, or downstream dealers.
- Physical attacks against offices or infrastructure.
- Findings exclusively against third-party sub-processors; report those to the sub-processor directly.

Compliance roadmap

SOC 2

SOC 2 Type 1 readiness target: **Q3 2026**. The attestation date will be announced after the assessment scope is confirmed. Type 1 attestation describes the design of controls at a point in time. Type 2 attestation, which describes operating effectiveness over a period (typically 6–12 months), follows.

A SOC 2 readiness assessment was conducted in the most recent product cycle and the control gaps identified are tracked against the target above. Customers requiring SOC 2 today are referred to the SOC 2 attestations of our sub-processors (Supabase, Stripe, Postmark, Railway, Cloudflare, Sentry) for evidence of the underlying infrastructure layer’s controls.

ISO 27001

ISO 27001 readiness assessment is on the roadmap. Engagement of an accredited certification body is contingent on completing the SOC 2 Type 2 program first; the operational and documentation overlap between the two regimes is significant and we plan to pursue them in sequence rather than in parallel.

HIPAA

Not applicable. partnermax processes business-to-business data in the automotive sector and does not process protected health information (PHI). If your use case touches PHI through a different product surface, contact `support@dealermax.app` before integrating.

CCPA / California state privacy law

Azure S.r.l. is an EU-based controller and does not directly market to or transact with California residents through partnermax. To the extent your partner application serves California residents, you are the controller of that consumer-facing relationship and are responsible for your own CCPA / CPRA compliance posture. We act as your processor under the shared responsibility model described below and will honor verifiable consumer rights requests forwarded to us under your direction.

Threat model

The following STRIDE summary captures the principal threats considered in the security design of partnermax. It is a high-level orientation, not an exhaustive enumeration.

Spoofing — impersonation of a legitimate identity

- **Threat:** Attacker obtains or guesses a partner API key and impersonates the partner.
- **Mitigation:** High-entropy random keys; constant-time hash comparison server-side; one active key per partner/environment; support-managed replacement with immediate old-key deactivation.

Tampering — unauthorized modification of data in transit or at rest

- **Threat:** Network attacker modifies API requests in flight.
- **Mitigation:** TLS 1.3 with forward-secret cipher suites; HSTS with preload; certificate pinning at the CAA layer.
- **Threat:** Insider or persistent attacker modifies stored audit records.
- **Mitigation:** Append-only audit table with no UPDATE/DELETE privileges to the service account; daily replicated archive with object-lock; engineer database access logged.

Repudiation — actor denies having performed an action

- **Threat:** A partner disputes a configuration change or a dealer creation.
- **Mitigation:** Comprehensive audit log including timestamp, partner_id, key_prefix, source_ip, user_agent, and trace_id; logs retained 12 months minimum and longer for disputed records.

Information disclosure — unauthorized access to data

- **Threat:** Cross-tenant data access via parameter manipulation.
- **Mitigation:** Server-side `partner_id` enforcement on every query; defense-in-depth filtering at the database layer; 404 (not 403) for unauthorized resource references to avoid existence disclosure.
- **Threat:** API key recovery from database dump.

- **Mitigation:** Keys hashed at rest; plaintext keys never persisted; redaction of `Authorization` header at the log edge.
- **Threat:** Sub-processor data exfiltration.
- **Mitigation:** Encryption at rest with provider-managed keys; minimal data sharing (e.g., Sentry receives error fingerprints and stack traces but no request bodies); contractual controls under DPA + SCCs.

Denial of service — degradation of availability

- **Threat:** Volumetric or application-layer attacks against the API.
- **Mitigation:** Cloudflare edge for L3/L4 DDoS absorption; per-partner rate limiting at the application layer; circuit-breaker patterns for downstream sub-processor outages so that the platform degrades gracefully rather than cascading.

Elevation of privilege — gaining unauthorized capabilities

- **Threat:** Partner attempts to invoke administrative or cross-partner operations.
- **Mitigation:** Least-privilege role design at the API; admin-only endpoints exposed on a separate hostname not reachable with partner keys; segregation of duties for internal engineering actions on customer data, with two-person review for high-impact operations.

Shared responsibility model

Security is operated across three layers. This table makes explicit which party owns which control.

Control area	DealerMAX	Partner	Downstream dealer
Physical security of data centers	✓ (via Supabase, AWS)	—	—
Network and edge DDoS protection	✓	—	—
Operating system and database patching	✓	—	—
API platform security (auth, ACL, rate limit, audit log)	✓	—	—
Encryption of customer data at rest and in transit	✓	—	—
Sub-processor management and DPAs	✓	—	—
Vulnerability scanning of partnermax code	✓	—	—
Incident detection and response at platform layer	✓	—	—
Secure storage of API keys	—	✓	—
Authorization checks before invoking partnermax (e.g., does this user have the right to provision a new dealer)	—	✓	—
Patching of partner application stack	—	✓	—
Incident detection and response at partner application layer	—	✓	—
Lawful basis for processing dealer data	—	✓	—
Consumer-facing privacy notice on dealer microsites	—	—	✓
Lawful basis for processing consumer data (leads, account, transaction)	—	—	✓
Consumer data subject rights handling (where dealer is direct controller)	—	—	✓
Security of dealer's own systems and staff access	—	—	✓

Where the partner is a software vendor reselling partnermax infrastructure to its dealer customers (the canonical pattern), the partner sits in the middle: responsible for safe operation of its own application, for proper provisioning of dealers in partnermax, and for educating its dealer customers on the controls they must operate. We provide dealer-facing guidance (URL documented forward; static page in preparation) that the partner may share or adapt.

Incident notification

In the event of a data breach affecting customer personal data, partnermax notifies affected customers within 72 hours of becoming aware, consistent with GDPR Art. 33. Notification includes:

- Description of the nature of the breach including, where possible, categories and approximate number of data subjects and records concerned.
- Name and contact details of our data protection contact.
- Description of likely consequences of the breach.
- Description of measures taken or proposed to address the breach and to mitigate possible adverse effects.

Notification reaches you through three channels in parallel:

Channel	Audience	Purpose
Email to the partner technical contact	Engineering / operations lead	Actionable technical detail and any required customer action
Status page at `https://developers.dealermax.app/status`	All partners and observers	Authoritative timeline of platform-impacting events
Email to the partner legal/compliance contact	Compliance and legal	Formal Art. 33 notification with the breach record reference

If the breach affects fewer than the threshold for individual data-subject notification under Art. 34 but does affect a single partner's data, we will notify that partner directly and not through the status page.

We maintain an internal incident response playbook covering detection, containment, evidence preservation, customer communication, regulator notification (Garante per la protezione dei dati personali, the Italian DPA), post-incident review, and corrective action. Material lessons-learned from incidents are incorporated into the platform and disclosed to partners in the next quarterly security update.

For lower-severity operational incidents (degraded performance, regional sub-processor outages with no customer data impact), notification is via the status page only.

Contacts

Purpose	Address
Vulnerability reports	`support@dealermax.app` (PGP available, see Responsible disclosure)
Data Processing Agreement, legal questions	`support@dealermax.app`
Privacy and data subject rights	`support@dealermax.app`
General support	`support@dealermax.app`
Press	`support@dealermax.app`
Status and incidents	`https://developers.dealermax.app/status`

Related documentation

- Authentication — API key model, rotation, and best practices.
- Webhooks — v1 polling guidance and future design notes.
- Errors — RFC 7807 problem details format and security-relevant status codes.
- Brand Guidelines — permitted use of the DealerMAX brand by partners.
- Glossary — terminology definitions.

CHAPTER 18

Onboarding Journey

> *From verified company signup or enterprise enablement to the first production API call.*

PartnerMAX has two onboarding paths. They share the same public API host, `https://api.dealermax.app`, and the same SDK contract, but they do not represent the same commercial flow.

Path	Use when	Commercial posture
Standard Trial Path	A partner wants to evaluate SDK/API integration quickly.	Self-service, 15-day live trial, limited enterprise access, no custom SLA, no contractual commitments until conversion.
Enterprise Onboarding Path	A partner needs procurement, security review, custom quotas, DPA/SLA, or migration planning before launch.	Commercial agreement, support-managed credentials, sandbox and production enablement, custom support model where contracted.

The generated SDK exposes dealer creation, paid Motornet lookup, stock and NLT operations through the public host. Partners should use SDK methods, not raw internal service hostnames.

Standard Trial Path

1. The partner opens `https://developers.dealermax.app/signup`.
2. The partner enters work email and Italian Partita IVA.
3. The backend verifies the anti-abuse challenge and resolves company data from OpenAPI.it IT-advanced.
4. The backend sends a one-time magic link to the email address.
5. The partner opens the magic link, reviews the verified company summary, and accepts the required legal checkboxes.
6. DealerMAX creates the partner account, starts a 15-day live trial, grants 3,000 credits, and returns one live Partner key exactly once.
7. The partner stores the key in its own secret manager and starts integrating with SDK/API calls.
8. Stripe subscription must become active before the trial expires to keep production access.

The trial key is a real live key, but it has `expires_at` set to the trial end. This means every service that uses the shared key resolver rejects the key after the trial if the subscription has not converted. When Stripe becomes active, the webhook clears the trial expiry on the active key.

First API Calls

The recommended first calls are:

```
GET /v1/keys
client.dealers.create(...)
GET /v1/dealers/{external_dealer_id}/nlt/offers
POST /v1/dealers/{external_dealer_id}/vehicles
```

Dealer creation for SDK integrations uses `client.dealers.create(...)`. The generated client sends the request to the partner-registry endpoint:

```
POST /api/partner/dealers
```

Partner dealer IDs are opaque external IDs owned by the partner. DealerMAX does not collect the partner dealer's commercial anagraphics during this flow.

Use the default `activate=true` when the partner wants to configure stock, NLT settings, or catalog reads immediately. A reference created with `activate=false` remains visible in the partner registry, but PartnerMAX `/v1` SDK calls intentionally omit it until the partner activates it.

Delete/revoke operations are soft from the partner API perspective: the mapping and audit history remain available for platform audit and idempotency. A separate support/privacy workflow is required for hard erasure or anonymization.

Credits

The trial grant is written to the normal credit ledger:

```
credit_transactions.transaction_type = ADD
credit_transactions.note = PartnerMAX trial grant | expires_at=...
```

Credits are consumed by the DealerMAX billing ledger for paid services such as targa/VIN lookup. PartnerMAX must reuse that ledger rather than implementing a separate wallet.

Stripe Conversion

Stripe is optional at trial start and mandatory for continuation. The trial completion response may include a Checkout URL when Stripe is configured. After the subscription becomes `active` or Stripe-side `trialing`, the webhook:

- links the Stripe subscription to the partner user;
- updates `dmax_subscription_status`;
- extends `dmax_access_until`;
- clears trial `expires_at` from the active Partner key.

Terminal subscription states revoke active Partner keys and suspend partner dealer references.

Enterprise Onboarding Path

Enterprise onboarding is the support-managed path for partners that require a commercial agreement before production rollout. It covers procurement, DPA, security review, custom quotas, SLA/support terms, sandbox enablement, production enablement, and migration planning.

The Enterprise Onboarding Path may use sandbox keys before production enablement and live keys after contractual approval. It does not change the technical rule that partners call `https://api.dealermax.app` through the SDK rather than internal service hostnames.

CHAPTER 19

Brand Guidelines

> *How to reference DealerMAX in your product, marketing, and documentation.*

These guidelines describe how partners may use the DealerMAX name, logo, and associated terminology when integrating with partnermax. The intent is twofold: protect the integrity of a brand many of our partners' own customers will recognize, and give you a clear, low-friction path to communicate your integration.

If anything in this document is ambiguous in your specific situation, the safest move is to email `support@dealermax.app` with a draft. Review turnaround is typically 2 business days. We would much rather answer a question than ask you to take something down.

Brand terminology

DealerMAX has three trademark terms that should be used in their canonical form. The first two are Italian; we leave them in Italian because they are part of the registered trademark portfolio and because the network operates as an Italian-origin brand even when communication is in English. Provide an English gloss in parentheses on first use; subsequent references in the same document may drop the gloss.

Term	Canonical form	First-use gloss	Notes
DealerMAX	DealerMAX	—	Always capitalized as shown. Not "DealerMax", not "Dealermax", not "Dealer MAX".
AIO	AIO (AI Optimization)	the discipline of structuring content so AI surfaces can cite it accurately	Acronym is canonical; the expansion in parentheses is for first-use clarity.
Dealer Identity Layer	Dealer Identity Layer	the infrastructure category DealerMAX defines	Title-case, all three words. The English category descriptor is the canonical form.
Presenza Citabile	Presenza Citabile (Citable Presence)	the outcome an AIO-equipped dealer achieves: being citable in AI answers	Italian. Leave in Italian even in English copy. Gloss on first use, drop on subsequent.

Other product names and surfaces:

- **partnermax** — lowercase. The developer-facing layer. When referring to the service in running text, "partnermax" reads correctly without capitalization. At the start of a sentence, "Partnermax" is acceptable for grammatical reasons.
- **DealerMAX network** — the collective of dealers, AI surfaces, partners, and infrastructure. Used when referring to the cross-tenant capabilities (e.g., "the DealerMAX network's MCP server").
- **MCP server** — when referring specifically to the Model Context Protocol server at `mcp.dealermax.app`.
- **NLWeb /ask** — when referring to the Microsoft NLWeb-compliant natural-language endpoint at `apimax.azcore.it/public/ask`.

Do not abbreviate "DealerMAX" to "DM", "DMx", or any other shorthand in customer-facing copy. Internal engineering shorthand is fine.

Brand assets

Asset bundle

The complete brand kit is available at:

```
https://developers.dealermax.app/brand-kit.zip
```

Note: this URL is documented forward; the asset bundle preparation is being finalized. While the canonical archive is in preparation, partners may request brand assets directly via `support@dealermax.app` and we will provide vector and raster files appropriate to your use.

The bundle contains:

- Primary logomark in SVG (vector, scales without quality loss).
- Primary logomark in PNG at 72 DPI for web and 300 DPI for print, in multiple sizes (32, 64, 128, 256, 512, 1024 px wide).
- Light-mode variant (logomark in dark color on light background).
- Dark-mode variant (logomark in light color on dark background).
- “Powered by DealerMAX” badge in SVG and PNG, light and dark.
- Brand color palette as a Sketch / Figma library file.
- Typography guidelines for setting “DealerMAX” wordmark inline.

Clear-space requirement

The DealerMAX logomark must be displayed with surrounding clear space equal to the height of the “D” character in the mark. Nothing — competing brand marks, text, navigation elements, decorative borders — should encroach on this space.

```
┌──────────────────┐
│     (clear)      │
│                  │
│   [LOGOMARK]     │
│                  │
│     (clear)      │
└──────────────────┘
```

Minimum size

The logomark must not be rendered smaller than:

- **48 px wide** for digital screen use (web, mobile, in-product UI).
- **12 mm wide** for print.

Below these sizes, the wordmark becomes illegible and the mark loses recognizability. When you cannot meet minimum size, use the “Powered by DealerMAX” badge (which is designed to be legible at smaller dimensions, down to 24 px height) or drop the visual mark and use the text “DealerMAX” instead.

Backgrounds

- The light-mode logomark is approved for use on white, near-white (#F5F5F5 or lighter), and pastel backgrounds with sufficient contrast.
- The dark-mode logomark is approved for use on black, near-black (#1A1A1A or darker), and saturated dark color backgrounds with sufficient contrast.
- Do not overlay the logomark on photographic backgrounds, gradient backgrounds with insufficient uniform contrast, or busy patterns. If your design context requires a photographic backdrop, place the logomark in a solid-color container (rounded

rectangle, “lockup card”) to preserve the clear-space requirement.

Prohibited modifications

Do not:

- Recolor the logomark. The brand yellow `#FDB825` is part of the registered mark.
- Stretch, skew, rotate, or otherwise distort the logomark.
- Apply visual effects (drop shadow, glow, embossing, outlining) to the logomark.
- Crop or extract elements from the logomark (e.g., using only the “D” letterform).
- Combine the logomark with another visual mark inside a single container so as to create an apparent co-mark or sub-mark.
- Use animated or moving versions of the logomark, except where DealerMAX has supplied an approved motion variant.
- Re-render the wordmark “DealerMAX” in a typeface other than the brand-supplied set (Michroma + the brand-defined fallback stack). For text references where the wordmark is set inline with body copy, use your body typeface — do not attempt to mimic the brand typography.

Permitted use of “Powered by DealerMAX” badge

The “Powered by DealerMAX” badge is the recommended way for partners to signal the integration to their own customers. Its use is optional — there is no contractual requirement to display it — but where you choose to display it, follow these rules.

Where to display

- **Footer of the partner application or website.** This is the canonical placement. The badge sits alongside other “Powered by” or “Built with” attribution that your product carries.
- **Integration setup screen.** When a dealer onboards through your application and the experience touches DealerMAX infrastructure (e.g., the dealer’s microsite, AI-citation features, NLT catalog wiring), it is appropriate to display the badge on the configuration page that explains the integration.
- **Partner marketplace listing.** If your application is listed in a third-party marketplace (DMS vendor app store, automotive software directory) and the listing supports a “Technologies used” or “Integrations” section, the badge belongs there.

Where not to display

- **Customer-facing AI surfaces.** If you are surfacing AI-generated text, descriptions, or recommendations to end consumers, do not place the badge on the consumer-facing output. The consumer relationship is with the dealer, not with DealerMAX.
- **Pricing or commercial pages of your product.** Co-branding next to your pricing implies a joint commercial relationship that does not exist. Keep the badge in attribution contexts.
- **Email signatures, business cards, social media profile images.** These imply employment or formal affiliation with DealerMAX. Don’t.

Sizing

The badge must be rendered at a minimum height of **24 pixels** for digital use. There is no maximum size, but in practice the badge should be visually subordinate to your own product brand. A useful rule: the badge height should be no greater than 1.5× the height of your “Copyright” footer text.

Color variants

Two color variants are provided: light-mode (intended for use on white or light backgrounds) and dark-mode (intended for dark backgrounds). Do not use the light variant on dark and vice versa — contrast will fail and the badge will look broken.

Do not recolor the badge to match your product palette. The brand yellow accent is part of the visual mark.

Proportions

The badge bundle ships the badge in fixed aspect ratio. Do not stretch the badge. Scale uniformly only.

Permitted vs non-permitted language patterns

The following table is the authoritative guide to how to reference DealerMAX in writing.

Context	Permitted	Non-permitted	Why
Attribution in product or footer	“Powered by DealerMAX”	“DealerMAX inside”	“Inside” implies an embedded component relationship that misstates the architecture.
Describing your integration in marketing	“Integrates with the DealerMAX network”	“DealerMAX-compatible” without prior written permission	“Compatible” implies a certification we do not currently award.
Describing AI-citation features in marketing	“AI-citation infrastructure by DealerMAX” or “powered by DealerMAX’s AI-citation layer”	“DealerMAX AI” as a standalone product name in your copy	“DealerMAX AI” is not a standalone product; it is a capability of the network.
Describing partnermax itself	“the DealerMAX partner API” or “the DealerMAX integration API”	Presenting partnermax as your own developed software, e.g., “our integration platform” without attribution	This is a non-starter. partnermax is DealerMAX intellectual property.
Naming your competing or adjacent product	Any name that does not evoke the DealerMAX brand	Similar-sounding names like “DealerPro”, “MaxDealer”, “DealerMAX Pro”, “DealerMAXX”, “DealerMAX++”	Confusing trademarks expose both companies to dispute and dilute customer trust.
Press release on your integration	“Today, [Partner] announced integration with the DealerMAX network…” (after press review, see below)	A press release positioning your product as a replacement for or sub-brand of DealerMAX	Misrepresents the partnership and creates customer confusion.
Technical documentation on your side	Reproducing snippets of partnermax documentation under fair use with attribution; linking to `developers.dealermax.app`	Wholesale copying of partnermax documentation under your own brand	Documentation is licensed CC BY 4.0 — see License. Attribution is the minimum requirement.
Customer support to your end users	Describing the integration honestly: “Our integration uses DealerMAX’s network infrastructure for AI citation”	Implying that customer-reported issues will be diagnosed or resolved by DealerMAX	Your customers are your customers. Escalation paths to DealerMAX run through you, not directly from your end user.

Specific copy patterns to avoid

These have come up in partner copy in the past and have required correction:

- **“DealerMAX-certified”** — there is no certification program. Until and unless one exists, do not claim certification.
- **“DealerMAX-preferred partner”** — the preferred-partner program is not yet formalized. Until it is, claims of preferred status are not approvable.
- **“Built on DealerMAX”** — “built on” implies your application would not exist without DealerMAX, which overstates the integration. “Integrates with DealerMAX” is the correct phrasing for most cases.
- **“DealerMAX Engine”**, **“DealerMAX Cloud”**, **“DealerMAX OS”** — these are not products in our portfolio. Do not coin them.

Press releases and announcements

Any press release, public blog post, podcast episode, video, or recorded webinar that:

- Uses the DealerMAX name in the title, headline, or first paragraph; OR
- Uses DealerMAX brand assets prominently; OR
- Makes any factual claim about DealerMAX (size, capabilities, roadmap, market position, customer base, technology)

must be reviewed and approved by DealerMAX before publication.

Review process

1. Email draft to `support@dealermax.app`. Include: - The full text of the release / post / talking points. - Any visual assets in their intended placement (mockup, layout). - The planned distribution channels and approximate audience size. - Your target publication date.
2. Service-level commitment: **5 business days** from receipt to first response. Reviews fall into one of three outcomes: - **Approved as-is.** Publish at your scheduled date. Please send the live link once published; we will amplify on the DealerMAX channels at our discretion. - **Approved with edits.** A revised draft is returned with suggested changes (typically corrections to fact or brand-language alignment). Re-circulate for confirmation if you make material changes to anything outside the suggested edits. - **Held for discussion.** A small subset of submissions require a brief call to align (typically when the release frames the relationship in a way that is materially inaccurate). We will propose times within the 5-business-day SLA.
3. Approval is good for the specific draft and the specific publication. If you reuse copy elsewhere or syndicate to a new venue, no fresh review is required as long as the content is materially the same. Substantial rewrites need a fresh review.

Co-branded announcements

For Enterprise-tier partners launching a publicly visible integration, DealerMAX may co-brand the announcement: joint press release, joint blog post, coordinated social distribution, or a launch event. This is a deeper level of engagement than the standard review and requires more lead time (typically 3–4 weeks). Surface the opportunity to your account manager early.

Co-branded announcements are not a contractual guarantee of Enterprise tier; they are arranged based on strategic fit, audience overlap, and timing. Most launches at any tier proceed under the standard review process described above and that path is the right one to plan around.

Domain naming conventions for partner-facing pages

You are free to choose any subdomain pattern you like for the pages your application exposes to your dealer customers — these are your pages, your brand, your domain. Some partners use `dealers.your-product.com`, others use `your-product.com/dealers/{slug}/`, others use per-dealer subdomains like `{dealer-slug}.your-product.com`. None of those choices implicates DealerMAX brand or trademark.

What you **must not** do:

- Register a second-level domain that contains “dealermax” or a close variant. Examples that are **not** permitted: `dealermax-integration.com`, `dealermax-partners.io`, `mydealermax.com`, `dealermaxx.app`, `dealer-max.dev`. These create confusion in the market and search results, and we will pursue enforcement.
- Register a subdomain on a generic domain that mimics our developer-facing presence. Examples that are **not** permitted: `developers.dealermax.example.com`, `dealermax.api.your-product.com`.
- Use the DealerMAX wordmark in URL paths that imply ownership of the brand: `your-product.com/dealermax/...` is permitted only if the path is genuinely about the DealerMAX integration (e.g., a help center article). It is not permitted as a routing prefix for arbitrary content.

What you **should** do:

- If you want to publish a help center article specifically about your DealerMAX integration, name it descriptively: `your-product.com/help/integrations/dealermax`. This is approved and helpful for discoverability.
- If you build a product page describing the DealerMAX integration, similarly: `your-product.com/integrations/dealermax`. Approved.

When in doubt, the principle is whether the domain or path implies that **you** are DealerMAX or are exclusively authorized by DealerMAX. If yes, do not. If it merely describes that you integrate with DealerMAX, fine.

Voice and tone alignment

DealerMAX's brand voice is shaped by a specific commercial position: AI infrastructure that runs behind the scenes so that the human salesperson stays at the center of the dealer's customer relationship. The canonical phrasing — translated from the brand's Italian source — is:

> *Your customer, before reaching you, has already talked to their ChatGPT. They are not looking for another bot — they are looking for a person to trust. DealerMAX does not replace the salesperson. The AI works behind the scenes; the salesperson is the human who closes.*

This is the consumer-facing position. In partner-facing developer documentation (like this one), the voice is straightforwardly technical — the consumer-facing positioning does not need to surface.

When you reference DealerMAX in customer-facing copy

If you reference DealerMAX in copy your end users will see — typically when describing why your product is better because of the integration — the recommended posture is:

- **Frame DealerMAX as infrastructure, not as a chatbot or assistant.** Phrases like "AI works behind the scenes" or "AI-citation infrastructure" align with the brand. Phrases like "DealerMAX talks to your customers" or "DealerMAX answers your customers' questions" do not — they reframe the brand into something it explicitly is not.
- **Center the human relationship.** The brand's commitment is that AI brings the customer to the dealer's door pre-qualified; the salesperson is the human who closes. Copy that frames DealerMAX as a replacement for the salesperson is off-brand.
- **Avoid words and phrases the brand has explicitly disavowed:** "conversational AI", "AI assistant for car shoppers", "voice agent", "chatbot widget", "automated customer nurturing". These are not part of how DealerMAX describes itself and reusing them positions the brand against itself.

We do not require partner copy to repeat DealerMAX's positioning verbatim. We do ask that copy referencing the brand not contradict it.

When you are NOT referencing DealerMAX

You are not required to align your own brand voice with DealerMAX's. Your product is yours, with its own voice and positioning. The alignment expectation applies only when your copy directly names the DealerMAX brand or describes the integration.

Logo violations and enforcement

Most brand-guideline issues we have encountered are good-faith mistakes — a logo recolored to fit a partner's site theme, a casual social post that mis-spells "DealerMax", a marketing draft that overstates the partnership. The remediation in those cases is a friendly email pointing to the specific guideline and asking for a correction within a reasonable window (typically 14 days for digital corrections, longer for already-printed materials).

Where misuse is willful, ongoing, or commercially material — for instance, registration of a confusing trademark, deliberate misrepresentation of the partnership in fundraising materials, or commercial use of the logomark with substantive modifications — DealerMAX reserves the right to pursue trademark enforcement under Italian and EU intellectual property law, including injunctive relief and damages where applicable. The “DealerMAX” word mark and associated visual elements are registered trademarks of Azure S.r.l. in the European Union and selected additional jurisdictions.

We very much prefer the friendly-email route. The above is here so that you understand the boundary, not because we expect to reach it.

License and intellectual property

Asset	License
partnermax service code (proprietary)	All rights reserved. Use is governed by your subscription agreement.
Software Development Kits (Python at PyPI, TypeScript at npm)	Apache License 2.0. Reuse, fork, and embed permitted under standard Apache terms.
Documentation at `developers.dealermax.app` (this site)	Creative Commons Attribution 4.0 International (CC BY 4.0). You may reuse, adapt, and distribute with attribution to “DealerMAX” and a link back to the source.
DealerMAX wordmark, logomark, “Powered by DealerMAX” badge, brand color palette	Registered trademarks of Azure S.r.l. Use is permitted under these brand guidelines; not permitted otherwise.

Attribution form for documentation

When you reuse documentation content under CC BY 4.0:

> *Adapted from “[Title of source page]” by Azure S.r.l., licensed under CC BY 4.0. Original at [URL].*

Acceptable in a footnote, in a colophon, or inline. The form is non-prescriptive as long as attribution is reasonably visible and the source URL is provided.

Contact

For brand questions, asset requests, or guideline clarifications: **`support@dealermax.app`**. For press review submissions: **`support@dealermax.app`**.

Related documentation

- Webhooks — technical integration surface.
- Security and Compliance — security posture you can reference in your customer-facing trust pages.
- Glossary — terminology used in this document and across the docs.

CHAPTER 20

Appendix - Production Evidence Pack

This appendix lists the production artifacts that support procurement, security review, and case-study claims. Public artifacts are independently reachable. Restricted metrics or dealer-specific examples are disclosed only during enterprise review, with an agreed measurement window and NDA where required.

Verification date: 2026-06-29.

Public evidence artifacts

- MCP endpoint health response Artifact: `https://mcp.dealermax.app/mcp`. Verification result: HTTP 200, `text/event-stream`, Streamable HTTP transport.
- MCP server-card Artifact: `https://mcp.dealermax.app/.well-known/mcp/server-card.json`. Verification result: HTTP 200, `application/json`.
- MCP registry reference Artifact: `https://registry.modelcontextprotocol.io/v0.1/servers/app.dealermax%2Fpublic-search/versions/latest`. Verification result: HTTP 200, `application/json`.
- ChatGPT Custom GPT publication route Artifact: `https://gpt.dealermax.app`. Verification result: HTTP 302 redirect to the ChatGPT GPT URL. This is publication/routing evidence, not an OpenAI endorsement.
- NLWeb `/ask` grounded response sample Artifact: `GET https://apimax.azcore.it/public/ask?query=franchigia&mode=summarize&limit=1` with header `x-dealer-domain: gamma-auto.it`. Verification result: HTTP 200; response contains `mode=summarize`, `summary_is_refusal=false`, and citation URL `https://www.gamma-auto.it/glossario/franchigia`.
- `llms.txt` sample Artifact: `https://developers.dealermax.app/llms.txt`. Verification result: HTTP 200, `text/plain`, 2,495 bytes.
- OpenAPI Custom GPT spec Artifact: `https://apimax.azcore.it/.well-known/openapi-custom-gpt.json`. Verification result: HTTP 200, `application/json`, 18,129 bytes.
- C2PA trust anchor Artifact: `https://apimax.azcore.it/.well-known/c2pa-root-ca.pem`. Verification result: HTTP 200, `application/x-pem-file`, 675 bytes.
- Partner-facing status page Artifact: `https://developers.dealermax.app/status`. Verification result: HTTP 200, `text/html`.
- Developer portal readiness probe Artifact: `https://developers.dealermax.app/health/ready`. Verification result: HTTP 200, body `{"status":"ready"}`.
- SDK package - Python Artifact: `https://pypi.org/project/partnermax/`. Verification result: public registry latest version verified as `0.13.0`.
- SDK package - TypeScript/JavaScript Artifact: `https://www.npmjs.com/package/partnermax`. Verification result: public registry latest version verified as `0.5.0`.
- PartnerMAX OpenAPI contract hash Artifact: `docs/appendix/openapi.json`. Verification result: SHA-256 `47d03a3fbeaaf62ae3f6b19b564beb7dccf12034dc2f20f1d4d71b72d4f7149a`.

Restricted or dealer-specific evidence

- Active dealer base Public brief discloses `27 production dealers`. Dealer list and dealer identities are not included in this public brief.

- Monthly MCP queries Not disclosed in the public brief. Available under NDA during enterprise review with the requested 30-day measurement window.
- Monthly Custom GPT invocations Not disclosed in the public brief. Available under NDA during enterprise review with the requested 30-day measurement window.
- `sitemap-llm.xml` sample No public central `sitemap-llm.xml` endpoint is asserted by this brief. Per-dealer sitemap evidence is available on request with dealer consent.
- JSON-LD rendered dealer example Available on request with dealer consent; no dealer-specific URL is embedded in this public brief.
- C2PA verification sample Available on request with a signed media asset and verification output; the public trust anchor is listed above.
- Status incident policy screenshot Available on request as a time-stamped export from the partner-facing status surface.

Evidence handling rules

- Do not describe the ChatGPT Custom GPT as pre-validated or endorsed by OpenAI unless written approval evidence is attached to the enterprise evidence pack.
- Do not describe the MCP server with universal AI-client compatibility claims. The supported claim is compatibility with MCP-capable clients that support Streamable HTTP transport.
- Do not publish usage metrics without a measurement window, source query, and owner approval.

CHAPTER 21

Appendix — Glossary

> *Terminology used throughout the partnermax documentation.*

Terms are listed alphabetically. Where a term is a brand or product reference, the canonical capitalization is preserved. Cross-references to other documentation pages are provided where relevant.

A

AIO (AI Optimization)

The discipline of structuring content, metadata, and infrastructure so that AI surfaces (chatbots, agents, search-with-AI summaries) can find, parse, and cite that content accurately. AIO is the canonical name for the practice DealerMAX implements on behalf of its dealer network. In customer-facing copy, leave the term in Italian-origin acronym form with the English expansion in parentheses on first use. See Brand Guidelines — Brand terminology.

API key

The credential a partner application uses to authenticate against the partnermax API. Keys carry a `pmk_live_` prefix in production and a `pmk_sand_` prefix in sandbox; the 12-character prefix is the value retained for display and log correlation. The plaintext key is shown once at creation; only the SHA-256 hash of the plaintext is stored at rest and resolved by the shared platform key resolver. See Authentication and Security and Compliance — Authentication and authorization.

C

Cascade deactivation

The automatic deactivation of all dealer references registered for a partner when its subscription reaches a terminal state such as `canceled` or `incomplete_expired`. DealerMAX hides those dealers from public/AI surfaces while keeping terminal dealer deletion as a separate lifecycle decision. Partners observe the resulting state through normal polling reads.

D

Dealer

An automotive retail business onboarded into the DealerMAX network via partnermax. Each dealer is a child record of the partner account that provisioned it (`utenti.parent_id` points to the partner's identifier). A dealer has business attributes (legal name, brand name, VAT number, location), an NLT economic configuration, and a derived rendering on the network's AI surfaces. See Dealers API.

Dealer-aware pricing

NLT canon price calculated using a specific dealer's NLT agency mark-up and NLT down-payment tiers. The same vehicle in the network catalog will produce different quoted prices for different dealers because their economic configuration differs. partnermax exposes dealer-aware pricing through the catalog endpoints when the caller scopes the request to a dealer identifier. See NLT API.

Dealer Identity Layer

The canonical category descriptor DealerMAX uses to position the product: the infrastructure layer that gives an automotive dealer an identity addressable, citable, and discoverable by AI surfaces. Title-case, all three words. See Brand Guidelines — Brand terminology.

DealerMAX

The parent SaaS platform and brand operated by Azure S.r.l., an Italian limited-liability company. partnermax is the developer-facing reseller layer of DealerMAX. The name is always rendered "DealerMAX" — not "DealerMax", "Dealermax", or "Dealer MAX". See Brand Guidelines.

DPA (Data Processing Agreement)

The contract between DealerMAX (as processor) and the partner (as controller) governing the processing of personal data under GDPR. Incorporates the European Commission's 2021 Standard Contractual Clauses where data may transit to a non-EU sub-processor. Available on request via `support@dealermax.app`. See Security and Compliance — GDPR posture.

G

Grace period

The window during which API access can continue for a partner account whose Stripe subscription has transitioned to a non-terminal collection state such as `past_due` or `unpaid`. Terminal states (`canceled`, `incomplete_expired`) do not wait for this window before key revocation and cascade deactivation.

H

HMAC-SHA256

Future-design cryptographic algorithm for outbound webhook payloads. Outbound webhooks are not available in PartnerMAX v1; current integrations use polling. See Webhooks.

I

Idempotency-Key

A request header used on POST operations to ensure the request is processed at most once. The header value is a partner-generated string (UUID recommended) that partnermax records together with the response. If the same key is presented again within the idempotency retention window, the original response is replayed rather than the operation being re-executed. The format and behavior match Stripe's idempotency-key convention. See [API conventions].

J

JSON-LD

JavaScript Object Notation for Linked Data. The structured-data format embedded in HTML pages served from dealer-facing microsites that AI surfaces consume to index dealer offerings, vehicles, locations, and AIO-relevant metadata. JSON-LD blocks rendered by the network conform to Schema.org vocabulary with extensions where appropriate.

M

MCP (Model Context Protocol)

An open standard published by Anthropic that defines how AI models discover, authenticate against, and invoke external tools and data sources. DealerMAX operates a public MCP server at `mcp.dealermax.app` that exposes the network's dealer and catalog data to MCP-aware AI clients. Dealers provisioned through partnermax are automatically indexed by this server. See the MCP specification for the standard.

N

NLT (Noleggio a Lungo Termine)

Italian for "long-term car rental". The flagship product DealerMAX dealers resell, structurally distinct from financing-led car sales. NLT contracts run typically 24–60 months, include maintenance and insurance, and are quoted with a monthly canon. partnermax exposes NLT catalog data and per-dealer NLT economic configuration via the API. See NLT API.

NLT agency mark-up

The percentage a dealer adds to the network's base canon price for an NLT offer. Configured per dealer via `PATCH /v1/dealers/{id}/nlt-settings`. The mark-up is the dealer's commercial margin on the rental; it is not a fee DealerMAX collects. Changes to mark-up propagate to AI-surfaceable pricing within the network index-refresh window and are visible through subsequent polling reads. See NLT settings API.

NLT down-payment tiers

Three configurable scenarios — `low`, `medium`, `high` — for the up-front advance payment a consumer can make on an NLT contract. Each tier is configured per dealer via the NLT settings API and returned by the catalog API as euro amounts at the offer list price. In v1, the `quotations[]` array carries the headline priced cells, while the three down-payment scenario amounts are returned separately for UI toggles and comparison. See NLT settings API.

NLT quotations

The headline set of price points returned for a single NLT offer: 3 contract durations (typically 24, 36, 48 months) x 6 mileage-per-year tiers (typically 10k, 15k, 20k, 25k, 30k, 40k km/year), for up to 18 quotations per offer. Each quotation is calculated using the dealer's NLT agency mark-up and the headline down-payment tier; the three down-payment scenario amounts are returned separately. See NLT API.

NLWeb

A Microsoft-led open standard for natural-language search over web-published content, announced at Microsoft Build 2025. NLWeb endpoints accept conversational queries and return structured results that conform to a published JSON schema. DealerMAX operates an NLWeb-compliant `/ask` endpoint at `apimax.azcore.it/public/ask` that serves both `mode=list` (semantic search across the network corpus) and `mode=summarize` (LLM-generated answers grounded in the corpus). See the NLWeb specification for the standard.

O

OpenAPI

The OpenAPI Specification 3.1, used as the single source of truth for the partnermax HTTP surface. The machine-readable enterprise contract is maintained in `docs/appendix/openapi.json` and shared during approved onboarding; the unrestricted public OpenAPI on the developer portal is the Dealer Public API contract. SDKs, documentation, and SDK-test fixtures are generated from the PartnerMAX spec. Drift between hand-written docs and the spec is treated as a documentation bug.

P

parent_id

A self-referential foreign key on the `utenti` table that establishes the hierarchical relationship between admin, partner, and dealer accounts. A partner record's `parent_id` points to the root admin account; a dealer record's `parent_id` points to the partner that provisioned it. The hierarchy is enforced at the database layer and at every authorization check in the API. See Security and Compliance — Authorization model.

Partner

A software vendor (or in some cases a large network operator) that resells DealerMAX infrastructure to its own automotive-dealer customers. The partner provisions dealers through partnermax, configures each dealer's NLT economics, and consumes catalog data on behalf of those dealers. If you are reading this, you are most likely a partner. See Brand Guidelines — Permitted vs non-permitted language patterns.

Partner key

The master API key issued to a partner account. Scoped to all dealers under the partner; a single key can read or modify any of the partner's dealers. Per-dealer scoping is not currently available — every key has full partner-scope access — though scoped keys are on the roadmap. See Authentication.

partnermax

The developer-facing layer of the DealerMAX platform. The service this documentation describes. Lowercase by convention; capitalize "Partnermax" only at the start of a sentence.

Presenza Citabile

Italian for "Citable Presence". The outcome the AIO discipline aims at: a dealer's information being structured such that AI surfaces — ChatGPT, Claude, AI Overviews in search, AI-powered assistants embedded in other products — can cite the dealer accurately when consumers ask relevant questions. Leave in Italian even in English copy; provide the English gloss on first use. See Brand Guidelines — Brand terminology.

R

RFC 7807

The IETF standard “Problem Details for HTTP APIs”. The error-response format partnermax returns on 4xx and 5xx responses. Every error includes `type` (URI), `title`, `status`, `detail`, `instance`, and partnermax-specific extension fields. See Errors.

RFC 8594

The IETF standard “The Sunset HTTP Header Field” together with the `Deprecation` header convention. partnermax sets these headers on endpoints scheduled for removal, with `Sunset` indicating the calendar date of removal and `Deprecation` indicating the deprecation status. See [Versioning and Deprecation].

RFC 9239

The IETF draft “RateLimit Header Fields for HTTP” defining the `RateLimit-Limit`, `RateLimit-Remaining`, and `RateLimit-Reset` headers. partnermax returns these on every authenticated response so client SDKs can adapt their request pacing. The standard is in active draft revision at IETF; partnermax tracks the current revision. See Rate Limits.

S

Sandbox

The isolated test environment served from the **same host** as production (`api.dealermax.app`). Routing between sandbox and production is determined by the API key’s prefix band — `pmk_sand_*` resolves against a separate Supabase database and a Stripe test-mode account, `pmk_live_*` resolves against production. Use sandbox for development, integration testing, and CI; never run customer traffic through it. See Sandbox.

SDK (Software Development Kit)

Generated client libraries that wrap the partnermax HTTP surface in idiomatic per-language constructs. The Python SDK (`partnermax` on PyPI, repo `github.com/DealerMax-app/partnermax-python`) and TypeScript / JavaScript SDK (`partnermax` on npm, repo `github.com/DealerMax-app/partnermax-node`) are generated from the OpenAPI spec using Stainless and released under the Apache License 2.0. See Versioning.

Stainless

The SDK code-generation platform used to produce the partnermax SDKs from its OpenAPI 3.1 spec. Generated SDKs are ergonomic, type-safe, and remain in sync with the spec across releases. The same approach is used by Anthropic, OpenAI, and Cloudflare for their public SDKs.

subscription gating

The mechanism by which API key validity is tied to the partner’s DealerMAX subscription state. When the subscription is in `active`, `trialing`, or any state during the grace period, API access is permitted. When the subscription enters terminal `canceled` and the grace period has expired, all partner-scoped API keys are revoked at the edge.

Subprocessor

A third-party service DealerMAX uses to operate the platform — Supabase, Stripe, Postmark, Railway, Cloudflare, Sentry, OpenAI. Each subprocessor is bound by a Data Processing Agreement that incorporates appropriate transfer safeguards. The current list is published at `https://developers.dealermax.app/.well-known/subprocessors.json` and reproduced in Security and Compliance — Sub-processor list.

Supabase

The managed PostgreSQL and object-storage provider underpinning the partnermax data layer. partnermax data resides in Supabase's EU region (`eu-central-1`, Frankfurt). Supabase is a SOC 2 Type 2 attested subprocessor. See Security and Compliance — Data residency.

U

utenti

Italian for "users". The master account table in the underlying DealerMAX schema that joins partners, dealers, and administrative accounts together via the parent_id hierarchy. The table is a relational implementation detail you do not interact with directly through the API, but it is referenced in some operational contexts (for example, audit log entries cite the `utenti.id` of the actor). See Security and Compliance — Authorization model.

W

W3C traceparent

The W3C Trace Context standard header propagated through partnermax requests for distributed tracing. The header is set by the partner SDK (or by the partner application if not using the SDK), forwarded through partnermax services, and recorded in the audit log under `trace_id`. Use the same `traceparent` end-to-end to correlate a partner-side log line with a partnermax audit entry. See the W3C Trace Context specification.

Related documentation

- Getting Started
- Authentication
- API Reference
- Webhooks
- Errors
- Security and Compliance
- Brand Guidelines