NUROX API SPECIFICATION LANGUAGE (NASL)

Technical Specification

Document ID	:	NRX-NASL-TS-001
Version	:	1
Status	:	Draft
Effective on	:	15 Jul 2025
Classification	:	nuroX Proprietary & Confidential

nuroX
Dubai

Sign-Off Page

(All signatories below confirm they have examined the document and authorized its release at the stated revision. Hand-written or digital signatures are equally acceptable.)

Document ID	**NXR-NASL-TS-001**	**Revision**	**1.0**
Title	*NUROX API Specification Language (NASL) – Technical Spec*		
Classification	Proprietary & Confidential		
Effective Date	15 Jul 2025		

Role	Name & Position	Signature	Date
Principal Author	**Walid Abdelal** *Chairman, Chief Vision & Science Officer*		
Reviewed by	**Omar Safwat** *Chief Technology Officer*		
QA / CM Check	**Jeff Heisler** *Software Configuration Manager*		
Endorsed by	**Abdulraouf Ismail** *Chief Executive Officer*		
Release Authority	**Jeff Heisler** Chief Product & Configuration Officer		

0 Foreword (non-normative)

This specification was developed by nuroX in cooperation with domain experts from software configuration management, DevOps SMEs, and enterprise-grade software developers. It aims to fill the discoverability and contract-management gaps left by existing code-first and schema-centric approaches (e.g., OpenAPI, gRPC-Protobuf). The language defined herein—NASL, the NUROX API Specification Language—is designed for global use across industries that rely on REST-style or REST-like HTTP interfaces.

1 Scope

This document specifies:

- **Syntax and semantics** of NASL files expressed in YAML 1.2 (§5-§7).
- **Rules for declaring** endpoints, parameters, request/response payloads, versioning, and non-functional hints such as cacheability and idempotency (§6).
- **Conformance requirements** for tooling (validators, code generators, linter plugins) and for NASL documents that claim compliance (§8).
- **Interoperability mapping** guidelines to widely-adopted specifications such as OpenAPI 3.1, AsyncAPI 3.0, GraphQL SDL, and gRPC Protobuf (§9).

The specification **does not** define transport security, authentication, or message encryption; it references existing standards (TLS 1.3, OAuth 2.1, OIDC 2.0) for those concerns.

2 Normative References

The following referenced documents are indispensable for the application of this specification. For dated references, only the edition cited applies; for undated references, the latest edition of the referenced document (including any amendments) applies.

Ref ID	Title	Publisher
[RFC 9110]	*HTTP Semantics*	IETF (RFC Editor)
[RFC 8259]	*The JavaScript Object Notation (JSON) Data Interchange Format*	IETF (IETF Datatracker)
[YAML 1.2]	*YAML Ain't Markup Language – Version 1.2*	yaml.org (yaml.org)
[JSON-Schema-Core 2020-12]	*A Media Type for Describing JSON Documents*	JSON Schema WG / IETF draft 2020-12 (json-schema.org)
[RFC 8927]	*JSON Type Definition (JTD)*	IETF (RFC Editor)
[OAS 3.1]	*OpenAPI Specification Version 3.1*	OpenAPI Initiative (OpenAPI Initiative Publications)
[IANA-JSON]	*Media-type registration application/json*	IANA (IANA)

[RFC 3986]	*Uniform Resource Identifier (URI): Generic Syntax*	IETF (IETF Datatracker)
[AsyncAPI 3.0]	*AsyncAPI Specification v3.0*	AsyncAPI Initiative (asyncapi.com)
[SAE EIA-649C]	*Configuration Management Standard*	SAE International (SAE International)

3 Terms, Definitions, and Abbreviations

TERM	DEFINITION
CACHEABLE	Response may be stored by clients or intermediaries in accordance with RFC 9111.
CONFORMANCE CLASS	Subset of NASL functionality against which a product or document can claim compliance (see §8).
EBNF	Extended Backus-Naur Form—a family of metasyntax notations, any of which can be used to express a context-free grammar.
ENDPOINT	A unique pair of path + HTTP method exposed by a service.
NASL	*NUROX API Specification Language*—a YAML-based, contract-first description language for HTTP APIs.
PROJECTION	Tailored response model optimised for a consumer's context (e.g., dashboard widget).

4 Introduction

4.1. Problem Statement

In code-first ecosystems (e.g., .NET minimal APIs) the contract for an endpoint like

```
app.MapGet("/api/scrm/dashboard", ([AsParameters] DashboardQuery q) => …);
```

is **implicit**—developers must delve into source files, IDE hover docs, or generated Swagger to learn the required query parameters and payload shape. This hampers:

- **Contract-first development** and parallel front-/back-end work.
- **UX design reviews** by non-developers.
- **Automated compliance checks** (e.g., GDPR field exposure, PCI masking).

4.2. Objective of NASL

NASL makes the contract **explicit and centrally governed**. A NASL excerpt:

```
endpoints:
  - id: dashboard-summary
    path: "/"
    method: GET
    parameters:
      - name: dateFrom  ; type: datetime ; default: "30 days ago"
      - name: dateTo    ; type: datetime ; default: "now"
      - name: entityType; type: enum[all,supplier,customer] ; default: all
    response: DashboardSummaryView
    cacheable: true
```

delivers a self-contained, diff-friendly artefact that:

- **Accelerates** front-end mocking and CI tests.
- **Enables** static validation before runtime.
- **Provides** a single source of truth for auditors, architects, and integrators.

5 Overview (Informative)

NASL is deliberately *small-surface, big-impact*: five design pillars (Table 5-1) map cleanly onto the stages you already run through in product design, implementation and release. The diagram on the previous page shows each pillar lighting-up a tangible component in the tool-chain.

5.1. Feature-to-Component Map

#	NASL PILLAR	WHAT IT LOOKS LIKE IN THE FILE	WHICH COMPONENT(S) USE IT	CONCRETE PAY-OFF
1	**YAML 1.2 grammar**	*Indent-only, comment-friendly* docspath: "/orders/{id}"	**Design Tools → NASL Spec → Document Parser**	UX or BA can diff & review the contract in Git **before** any code exists.
2	**Strict type system**	type: enum[jet, avgas, diesel]union[Card, Wire, Crypto]	**Schema Validator → Type System → Code Generator**	Build breaks the moment a wrong value sneaks in; generated DTOs carry the same guarantees into .NET 9 & TS 5.
3	**Non-functional metadata**	cacheable: trueidempotent: PUTtimeout: 3s	**API Generator → Generated Artifacts**	Ops can wire HTTP caching & retries straight from the spec—no tribal knowledge.
4	**Tool-chain-agnostic**	$dialect: nasl/2025-07	**Transpiler targets:** .NET 9, Rust 1.80, TS 5.x	Same NASL file produces controllers, clients *and* human docs. Language wars avoided.
5	**Semantic versioning**	service.version: v1.2.0deprecated: 2026-01-31	**Registry → Backend / Frontend build gates**	SBOMs & pipelines know exactly which breaking changes land when; old clients keep compiling against v1.1.* until ready.

5.2. Happy-Path Through the Ecosystem

1. Design kick-off

A UX designer in Figma sketches a new "Widget 1" widget → exports an *endpoint checklist* into the repo. *A systems engineer* writes the matching NASL snippet.

2. Validation & Registry

A commit hook calls the Validation CLI. *Pass* → the spec is version-tagged, signed, and stored in the NASL Registry under dashboard/1.2.0.nasl.yaml.

3. Code & Doc Generation

The same CI job invokes:

- Code Generator → .cs minimal-API stubs + strong-typed Widget1Query record.
- Documentation Generator → HTML site & internal PDF.
- API Generator → OpenAPI 3.1 overlay for external integrators.

  Artifacts land in *Generated Artifacts* and flow to both Backend and Frontend pods.

4. Implementation

Backend devs open Widget1Controller.cs containing empty handler stubs and TODO markers.
Front-end devs import @nurox/sdk/dashboard generated from the same spec—no manual typings needed.

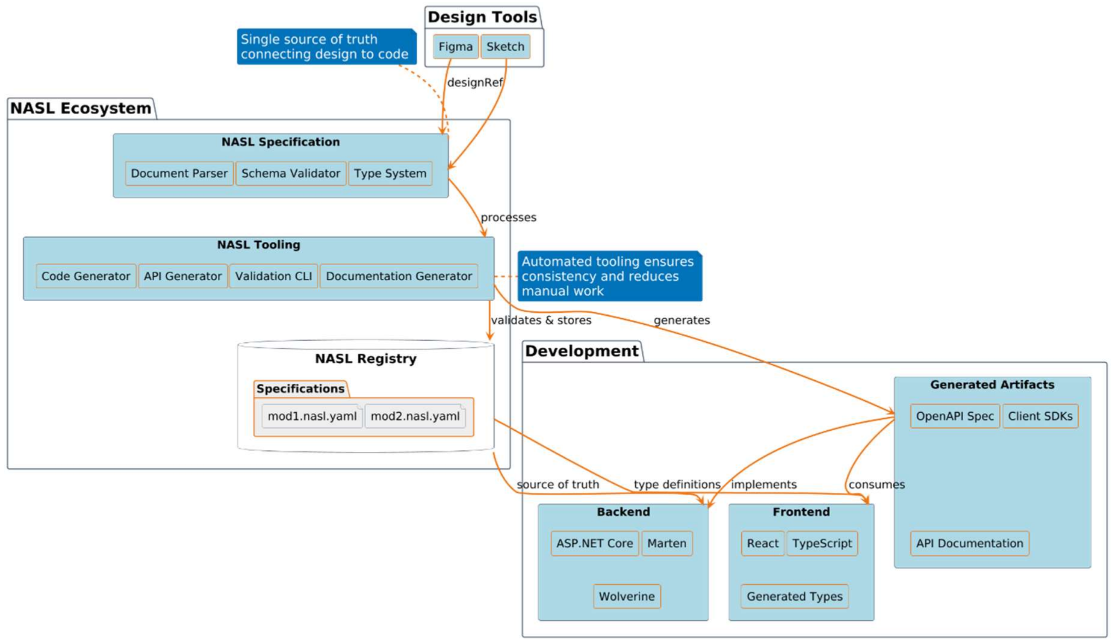
5. Runtime

Live pods fetch the signed spec from the Registry at boot:

- If a *breaking* major version appears (v2 .x), pods can refuse to start, alerting Ops.
- Non-breaking minors (v1.3.0) hot-load if the strict type check passes.

6. Governance & Audit

Because every NASL file carries Classification: Proprietary & Confidential, DLP scanners can flag leaks; CM tooling (SAE EIA-649C compliant) traces every deployed artifact back to its exact spec hash.



6 NASL EBNF Grammar (Normative)

6.1. Grammar

6.1.1. Top-level document structure

```
(* NASL - nuroX API Specification Language EBNF Grammar *)
(* Version 1.0 *)

(* Top-level document structure *)
<nasl-document> ::= <yaml-header> <specification> <authorship> <platforms> <api>
<endpoints>
                    [<projections>] [<data-contracts>] [<widget-bindings>]

<yaml-header> ::= <comment> <comment>
<comment> ::= "#" {<any-char>} <newline>
```

6.1.2. NASL-related

```

(* Specification section *)
<specification> ::= "specification:" <newline> <indent> <spec-fields>
<spec-fields> ::= <version-field> <module-field> <interface-field> <design-ref-
field>
                  [<description-field>] [<tags-field>]

<version-field> ::= "version:" <string> <newline>
<module-field> ::= "module:" <pascal-case-identifier> <newline>
<interface-field> ::= "interface:" <pascal-case-identifier> <newline>
<design-ref-field> ::= "designRef:" <uri> <newline>
<description-field> ::= "description:" <string> <newline>
<tags-field> ::= "tags:" <string-array> <newline>

(* Authorship section *)
<authorship> ::= "authorship:" <newline> <indent> <authorship-fields>
<authorship-fields> ::= <principal-field> <witness-field> <created-at-field>
                        <last-modified-field> <change-log-field>

<principal-field> ::= "principal:" <email> <newline>
<witness-field> ::= "witness:" <email> <newline>
<created-at-field> ::= "createdAt:" <iso-datetime> <newline>
<last-modified-field> ::= "lastModified:" <iso-datetime> <newline>
<change-log-field> ::= "changeLog:" <newline> <change-log-entries>

<change-log-entries> ::= {<change-log-entry>}
<change-log-entry> ::= <indent> "-" <change-entry-fields>
<change-entry-fields> ::= "date:" <iso-date> <newline>
                          "author:" <email> <newline>
                          "witness:" <email> <newline>
                          "changes:" <string> <newline>
                          ["version:" <version-string> <newline>]
```

6.1.3. Target Platform

```
(* Platforms section *)
<platforms> ::= "platforms:" <newline> {<platform-entry>}
<platform-entry> ::= <indent> "-" "type:" <platform-type> <newline>
                     [<indent> "versions:" <string-array> <newline>]
                     [<indent> "breakpoints:" <breakpoint-array> <newline>]
                     [<indent> "capabilities:" <string-array> <newline>]

<platform-type> ::= "responsive-web" | "ios-app" | "android-app" | "desktop-app" |
"api-only"
<breakpoint-array> ::= "[" <breakpoint> {"," <breakpoint>} "]"
<breakpoint> ::= "mobile" | "tablet" | "desktop" | "wide"
```

6.1.4. API Configuration

```
(* API Configuration *)
<api> ::= "api:" <newline> <indent> <api-fields>
<api-fields> ::= <base-field> <authentication> <versioning> [<rate-limit>]
[<cors>]

<base-field> ::= "base:" <api-path> <newline>
<api-path> ::= "/" {<path-segment>}
<path-segment> ::= <kebab-case-identifier> "/"

<authentication> ::= "authentication:" <newline> <indent> <auth-fields>
<auth-fields> ::= "type:" <auth-type> <newline>
                  "rationale:" <string> <newline>
                  ["config:" <config-object> <newline>]

<auth-type> ::= "jwt-bearer" | "api-key" | "oauth2" | "cookie-session" | "mutual-
tls"

<versioning> ::= "versioning:" <newline> <indent> <versioning-fields>
<versioning-fields> ::= "strategy:" <versioning-strategy> <newline>
                        "rationale:" <string> <newline>
                        ["config:" <config-object> <newline>]

<versioning-strategy> ::= "header" | "url-path" | "query-param" | "content-
negotiation"
```

6.1.5. Projections

```
(* Projections *)
<projections> ::= "projections:" <newline> {<projection-entry>}
<projection-entry> ::= <indent> "-" <projection-fields>
<projection-fields> ::= "name:" <pascal-case-identifier> <newline>
                        ["type:" <projection-type> <newline>]
                        "sources:" <string-array> <newline>
                        "refreshStrategy:" <refresh-strategy> <newline>
                        ["refreshRate:" <duration> <newline>]

<projection-type> ::= "marten-aggregate" | "event-view" | "sql-view" | "cached-
view"
<refresh-strategy> ::= "real-time" | "event-driven" | "scheduled" | "on-demand"
```

6.1.6. Endpoints

```
(* Endpoints *)
<endpoints> ::= "endpoints:" <newline> {<endpoint-entry>}
<endpoint-entry> ::= <indent> "-" <endpoint-fields>
<endpoint-fields> ::= "id:" <kebab-case-identifier> <newline>
                      "path:" <endpoint-path> <newline>
                      "method:" <http-method> <newline>
                      "description:" <string> <newline>
                      "platforms:" <platform-array> <newline>
                      [<parameters>]
                      [<body>]
                      <response>
                      [<endpoint-metadata>]

<http-method> ::= "GET" | "POST" | "PUT" | "PATCH" | "DELETE"
<platform-array> ::= "[" ("all" | <platform-list>) "]"
<platform-list> ::= <string> {"," <string>}

<parameters> ::= "parameters:" <newline> {<parameter-entry>}
<parameter-entry> ::= <indent> "-" <parameter-fields>
<parameter-fields> ::= "name:" <identifier> <newline>
                       "type:" <type-spec> <newline>
                       "location:" <param-location> <newline>
                       ["required:" <boolean> <newline>]
                       ["default:" <value> <newline>]
                       ["validation:" <string> <newline>]

<param-location> ::= "query" | "path" | "header"

<body> ::= "body:" <newline> <indent> <body-fields>
<body-fields> ::= "type:" <type-reference> <newline>
                  ["required:" <boolean> <newline>]

<response> ::= "response:" <newline> <indent> <response-fields>
<response-fields> ::= "type:" <type-reference> <newline>
                      ["status:" <http-status> <newline>]
                      ["cacheable:" <boolean> <newline>]
                      ["ttl:" <integer> <newline>]
```

6.1.1. Data Contracts

```
(* Data Contracts *)
<data-contracts> ::= "data-contracts:" <newline> {<contract-entry>}
<contract-entry> ::= <pascal-case-identifier> ":" <newline> <indent> <contract-
fields>
<contract-fields> ::= ["description:" <string> <newline>]
                      "fields:" <newline> {<field-entry>}

<field-entry> ::= <indent> <camel-case-identifier> ":" <newline> <indent> <field-
spec>
<field-spec> ::= "type:" <type-spec> <newline>
                 ["description:" <string> <newline>]
                 ["required:" <boolean> <newline>]
                 ["widget:" <kebab-case-identifier> <newline>]
```

6.1.2. Type Specifications

```
(* Type Specifications *)
<type-spec> ::= <primitive-type> | <array-type> | <enum-type> |
                <range-type> | <nullable-type> | <type-reference>

<primitive-type> ::= "string" | "number" | "boolean" | "datetime" | "date" |
"time"
<array-type> ::= <type-spec> "[]"
<enum-type> ::= "enum[" <enum-values> "]"
<enum-values> ::= <identifier> {"," <identifier>}
<range-type> ::= "number[" <number> "-" <number> "]"
<nullable-type> ::= <type-spec> "?"
<type-reference> ::= <pascal-case-identifier>
```

6.1.3. UI Widget Bindings

```
(* Widget Bindings *)
<widget-bindings> ::= "widget-bindings:" <newline> {<widget-entry>}
<widget-entry> ::= <kebab-case-identifier> ":" <newline> <indent> <widget-fields>
<widget-fields> ::= "dataPath:" <json-path> <newline>
                    "updateFrequency:" <update-frequency> <newline>
                    ["pollingInterval:" <duration> <newline>]
                    ["clickAction:" <string> <newline>]

<update-frequency> ::= "real-time" | "polling" | "manual"
<json-path> ::= "$" {"." <identifier> | "[" <integer> "]"}
```

6.1.4. Common Rules

```
(* Common Rules *)
<identifier> ::= <letter> {<letter> | <digit> | "_"}
<pascal-case-identifier> ::= <upper-letter> {<letter> | <digit>}
<camel-case-identifier> ::= <lower-letter> {<letter> | <digit>}
<kebab-case-identifier> ::= <lower-letter> {<lower-letter> | <digit> | "-"}

<string> ::= '"' {<any-char-except-quote>} '"'
<integer> ::= ["-"] <digit> {<digit>}
<number> ::= <integer> ["." <digit> {<digit>}]
<boolean> ::= "true" | "false"
<email> ::= <identifier> "@" <identifier> "." <identifier>
<uri> ::= <scheme> "://" <host> {<path-segment>}
<iso-datetime> ::= <iso-date> "T" <time> "Z"
<iso-date> ::= <year> "-" <month> "-" <day>
<duration> ::= <integer> ("s" | "m" | "h" | "d")
<version-string> ::= <integer> "." <integer> ["." <integer>]

<value> ::= <string> | <number> | <boolean> | <null>
<string-array> ::= "[" [<string> {"," <string>}] "]"
<config-object> ::= "{" {<identifier> ":" <value>} "}"
```

6.1.5. Character Classes

```
(* Character Classes *)
<letter> ::= <upper-letter> | <lower-letter>
<upper-letter> ::= "A" | "B" | ... | "Z"
<lower-letter> ::= "a" | "b" | ... | "z"
<digit> ::= "0" | "1" | ... | "9"
<newline> ::= "\n"
<indent> ::= "  " | "\t"
```

6.2. EBNF Tree

6.2.1. NASL Document

```
NASL Document
│
├── YAML Header (required)
│   ├── File path comment
│   └── NASL version comment
│
├── specification (required)
│   ├── version: string
│   ├── module: PascalCase
│   ├── interface: PascalCase
│   ├── designRef: URI
│   ├── description: string (optional)
│   └── tags: string[] (optional)
│
├── authorship (required)
│   ├── principal: email
│   ├── witness: email
│   ├── createdAt: ISO datetime
│   ├── lastModified: ISO datetime
│   └── changeLog: array
│       └── entry
│           ├── date: ISO date
│           ├── author: email
│           ├── witness: email
│           ├── changes: string
│           └── version: string (optional)
│
├── platforms (required)
│   └── platform[]
│       ├── type: enum
│       │   ├── responsive-web
│       │   ├── ios-app
│       │   ├── android-app
│       │   ├── desktop-app
│       │   └── api-only
│       ├── versions: string[] (optional)
│       ├── breakpoints: enum[] (optional)
│       │   ├── mobile
│       │   ├── tablet
│       │   ├── desktop
│       │   └── wide
│       └── capabilities: string[] (optional)
│
```

```
├── api (required)
│   ├── base: path (/api/...)
│   ├── authentication
│   │   ├── type: enum
│   │   │   ├── jwt-bearer
│   │   │   ├── api-key
│   │   │   ├── oauth2
│   │   │   ├── cookie-session
│   │   │   └── mutual-tls
│   │   ├── rationale: string
│   │   └── config: object (optional)
│   ├── versioning
│   │   ├── strategy: enum
│   │   │   ├── header
│   │   │   ├── url-path
│   │   │   ├── query-param
│   │   │   └── content-negotiation
│   │   ├── rationale: string
│   │   └── config: object (optional)
│   ├── rateLimit (optional)
│   │   ├── requests: integer
│   │   └── window: duration
│   └── cors (optional)
│       ├── origins: string[]
│       └── methods: HTTP method[]
│
├── endpoints (required)
│   └── endpoint[]
│       ├── id: kebab-case
│       ├── path: string
│       ├── method: enum
│       │   ├── GET
│       │   ├── POST
│       │   ├── PUT
│       │   ├── PATCH
│       │   └── DELETE
│       ├── description: string
│       ├── platforms: array
│       │   └── "all" | platform-type[]
│       ├── parameters (optional)
│       │   └── parameter[]
│       │       ├── name: identifier
│       │       ├── type: type-spec
│       │       ├── location: enum
│       │       │   ├── query
│       │       │   ├── path
│       │       │   └── header
│       │       ├── required: boolean (optional)
│       │       ├── default: value (optional)
│       │       └── validation: string (optional)
│       ├── body (optional)
│       │   ├── type: type-reference
│       │   └── required: boolean (optional)
│       │
│        ├── response
│        │   ├── type: type-reference
│        │   ├── status: HTTP status (optional)
│        │   ├── cacheable: boolean (optional)
│        │   ├── ttl: integer (optional)
│        │   ├── realtime: boolean (optional)
│        │   ├── paginated: boolean (optional)
│        │   └── pageSize: integer (optional)
│        ├── handler: string (optional)
│        ├── projection: string (optional)
│        ├── widgets: string[] (optional)
│        ├── authorization: string (optional)
│        ├── rateLimit: object (optional)
│        ├── deprecated: boolean (optional)
│        └── deprecationNotice: string (optional)
│
├── projections (optional)
│   └── projection[]
│        ├── name: PascalCase
│        ├── type: enum (optional)
│        │   ├── marten-aggregate
│        │   ├── event-view
│        │   ├── sql-view
│        │   └── cached-view
│        ├── sources: string[]
│        ├── refreshStrategy: enum
│        │   ├── real-time
│        │   ├── event-driven
│        │   ├── scheduled
│        │   └── on-demand
│        ├── refreshRate: duration (optional)
│        ├── retention: duration (optional)
│        └── indexes: string[] (optional)
│
├── data-contracts (optional)
│   └── [TypeName: PascalCase]
│        ├── description: string (optional)
│        └── fields
│            └── [fieldName: camelCase]
│                ├── type: type-spec
│                │   ├── Primitives
│                │   │   ├── string
│                │   │   ├── number
│                │   │   ├── boolean
│                │   │   ├── datetime
│                │   │   ├── date
│                │   │   └── time
│                │   └── Complex Types
│                │       ├── Type[]           (array)
│                │       ├── enum[...]       (enumeration)
│                │       ├── number[min-max] (range)
│                │       ├── Type?           (nullable)
│                │       └── TypeReference   (custom type)
│                ├── description: string (optional)
│                ├── required: boolean (optional)
│                ├── validation: string (optional)
│                ├── widget: kebab-case (optional)
│                └── example: value (optional)
```

```
└── widget-bindings (optional)
    └── [widget-id: kebab-case]
        ├── dataPath: JSONPath
        ├── updateFrequency: enum
        │   ├── real-time
        │   ├── polling
        │   └── manual
        ├── pollingInterval: duration (optional)
        ├── clickAction: string (optional)
        ├── permissions: string[] (optional)
        ├── platforms: string[] (optional)
        └── config: object (optional)
```

6.2.2. Visual Type System Tree

```
Type System
│
├── Primitive Types
│   ├── string
│   ├── number
│   ├── boolean
│   ├── datetime
│   ├── date
│   └── time
│
├── Composite Types
│   ├── Array: Type[]
│   ├── Enum: enum[value1, value2, ...]
│   ├── Range: number[min-max]
│   └── Nullable: Type?
│
├── Type References
│   └── PascalCaseTypeName
│
└── Value Types
    ├── string: "text"
    ├── number: 123, 45.67
    ├── boolean: true, false
    ├── null
    └── object: { key: value }
```

6.2.3. Naming Convention Tree

```
Naming Conventions
│
├── PascalCase (Types, Modules, Interfaces)
│   └── Examples: UserProfile, SCRM, Dashboard
│
├── camelCase (Fields, Properties)
│   └── Examples: firstName, activeDeliveries, totalRevenue
│
├── kebab-case (IDs, Endpoints, Widgets)
│   └── Examples: get-dashboard, active-deliveries, user-profile
│
└── SCREAMING_SNAKE_CASE (Constants - in code, not NASL)
    └── Examples: MAX_RETRIES, DEFAULT_TIMEOUT
```

6.3. NASL Class Diagram (informative)

Annex A provides the respective UML-class re5presentation of this language.

7 Conformance Requirements

A product, service, or document **MAY** advertise itself as **"NASL-Conformant"** only when *all* requirements in § 6.1–§ 6.5 are met.
Where the word **MUST** appears, fulfilment is mandatory for *every* conformance class; **SHOULD** marks strong recommendations; **MAY** marks optional features.

7.1. Schema-Validity Test (C-01)

a) A NASL file MUST validate against the normative JSON Schema published at https://spec.nurox.ai/nasl/2025-07/schema/core.json.
b) Validation is performed with the JSON Schema 2020-12 processor operating in *strict* mode (no implicit type coercion).
c) Implementers SHOULD expose CLI/CI targets named nasl-validate to automate this test.

7.2. Reference-Resolution Test (C-02)

a) $ref keywords MUST resolve:
b) Relative paths inside the same document.
c) Cross-document URIs rooted at the NASL Registry (nasl://registry/…) or HTTP(S).
d) Resolvers MUST NOT access file-system paths outside the spec root (.. escapements are illegal).
e) Cyclic $ref chains MUST raise a validation error.

7.3. Conformance Classes & Feature Matrix (C-03)

Class	Core Goals	Mandatory Sections / Keywords	Optional in lower classes?
A — Core Syntax	Basic contract-first interop	specification, authorship, platforms, **all** endpoints *without* body/response complex types	*N/A*
B — Extended Types	Typed payloads & data-contracts	Everything in Class A **plus** data-contracts, complex type-spec, nullable, enum, range, array syntax	Yes
C — Non-Functional	Runtime & ops metadata	All of Class B **plus**: cacheable, ttl, realtime, paginated, rateLimit, cors, platforms.breakpoint, projections section	Yes

D — Widget Binding *(future)*	UX glue layer	Class C **plus** widget-bindings	Yes

Claiming a higher class automatically implies conformance to all lower classes.

Implementer rule-of-thumb:

a) CLI validators MUST fail if a document declares conformance: B but omits any B-level keyword.
b) Runtime libraries SHOULD ignore unknown keywords unless they start with x-nasl-, in which case they SHOULD surface them via an extension mechanism.

7.4. Forward-Compatibility Rule (C-04)

Parsers MUST:

a) Ignore unknown top-level keys and endpoint-level keywords that are not defined in the declared $schema version.
b) Preserve the order and verbatim content of unrecognized key/value pairs when round-tripping (load → modify → save).
c) Fail hard only when a future keyword conflicts with a required one (e.g., two path keys).

7.5. Document-Structure Checklist (Informative)

SECTION	REQ'D IN CLASS	NOTES
YAML HEADER COMMENTS	A	Must show file path & NASL version
SPECIFICATION.VERSION	A	Follows *semver* (major.minor.patch)
AUTHORSHIP.PRINCIPAL	A	Valid RFC 5322 email
API.AUTHENTICATION.TYPE	A	Selection from fixed enum
BODY.TYPE & RESPONSE.TYPE	B	type-reference **MUST** point into data-contracts
RESPONSE.CACHEABLE	C	Boolean; ttl optional but integer seconds if present
PROJECTIONS	C	If projection.type: sql-view then indexes **SHOULD** be non-empty
WIDGET-BINDINGS	D	Requires dataPath JSONPath pointer

A convenient **one-page PDF checklist** is provided in Annex B for audit teams.

7.6. Conformance Statement Template

```
Product XYZ version … claims NASL Class B Conformance.
Validation results:
   - C-01: Pass (schema hash a1b2c3d…)
   - C-02: Pass (all $ref resolved)
   - C-03: Class B features complete
   - C-04: Parser ignores unknown x-acme-feature extension
```

Vendors SHOULD attach this statement (or machine-readable JSON) to release artefacts for automated supply-chain scanning.

7.7. Non-Conformance Handling

Any violation of **C-01**–**C-04 MUST** raise a NASL-ERR-CONFORMANCE error with:

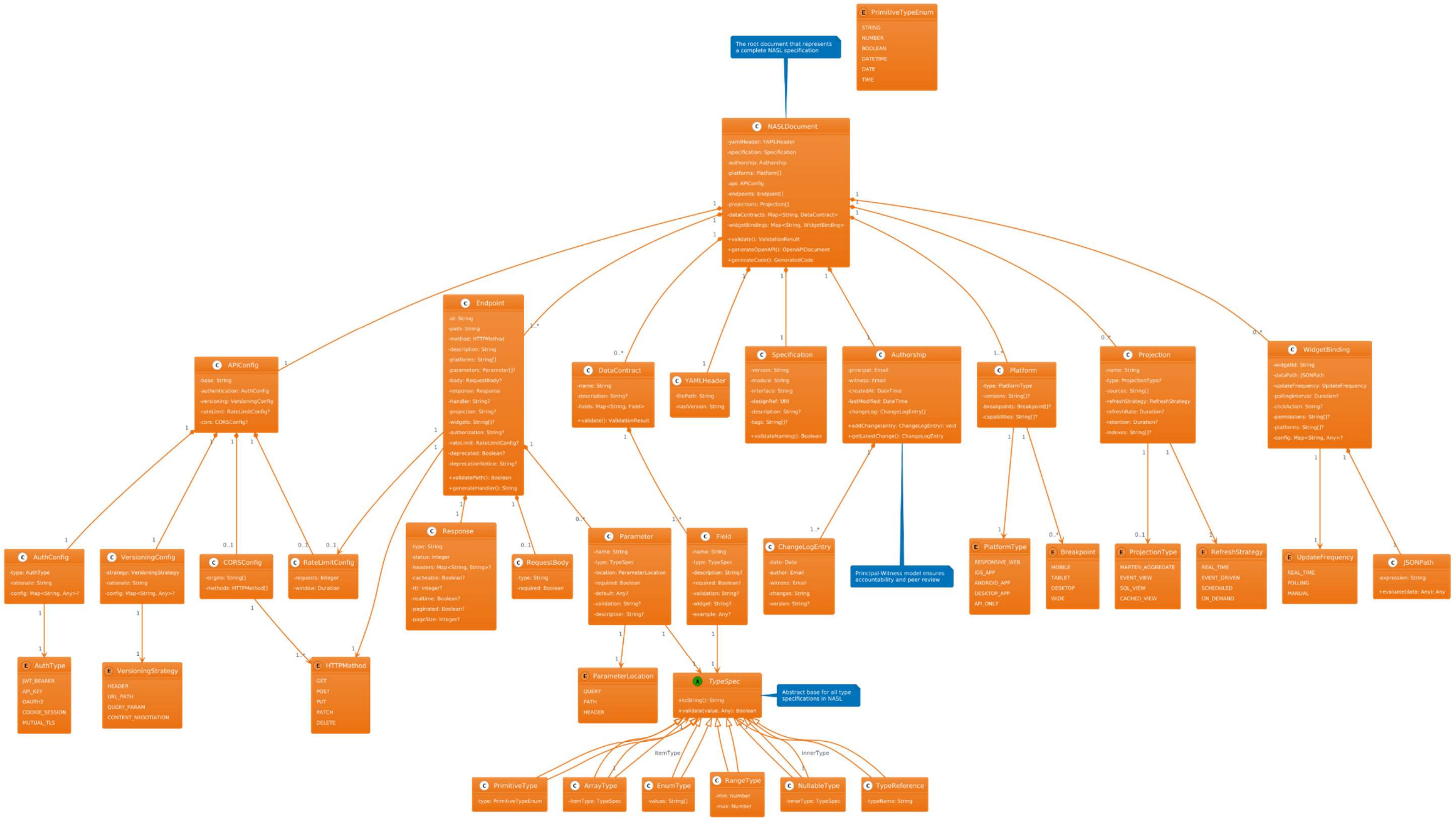
```
{
  "code": "NASL-ERR-C02-REF",
  "message": "$ref resolution failed at '#/endpoints/3/response/type'",
  "severity": "fatal",
  "doc": "https://spec.nurox.ai/nasl/errors#C02"
}
```

Pass the four core tests, pick a class (A-D), and you are safely “NASL-Conformant.” Everything else is gravy—and future-proof.

8 Change Log (To be auto-generated)

Rev	Date	Section(s)	Change	Author
0.1	2025-07-15	All	Initial industry draft	WA

ANNEX A
NASL UML-Class Diagram



ANNEX B
NASL COMNFORMANCE AUDIT CHECKLIST

1) Core Conformance Tests

Ref	Pass / Fail	Item
C-01	☐	**Schema Validity** – file validates against *core JSON Schema 2025-07* with a JSON-Schema 2020-12 strict processor.
C-02	☐	**$ref Resolution** – all $ref links resolve (relative or nasl:// / https://), no cycles, no .. path escapes.
C-03	☐	**Conformance Class** – declared Class **A / B / C / D** is present *and* every mandatory keyword for that class exists; higher-class keywords used only when prerequisites met.
C-04	☐	**Forward-Compatibility** – parser ignores unknown future keywords and preserves them on round-trip.

2) Document Structure Checks

i. YAML header comment shows *file path* and *NASL version*.
ii. specification block contains: version, module, interface, designRef.
iii. authorship block contains: principal, witness, createdAt, lastModified.
iv. platforms section lists at least one platform entry.
v. api section defines base path and authentication.type.
vi. Each endpoint has: id, path, method, response.type.

Class-specific additions

CLASS	EXTRA ITEMS TO VERIFY
B +	data-contracts section exists; all type-reference values resolve.
C +	Non-functional flags valid (cacheable, ttl, rateLimit, cors, etc.). If projections present → refreshStrategy & sources mandatory; indexes recommended for sql-view.
D	widget-bindings present → dataPath (JSONPath) & updateFrequency valid; pollingInterval required when updateFrequency = polling.

3) Audit Sign off

Auditor: ____________________ Date: ___________

Result: ☐ Pass ☐ Minor NC ☐ Major NC

Keep this checklist as a markdown snippet inside your repo (/docs/audit/annex-d.md) so CI pipelines can link directly to the requirements.