Software Architecture Documentation

Interface Documentation

A Comprehensive Guide to Documenting APIs, Events, Protocols, and System Boundaries

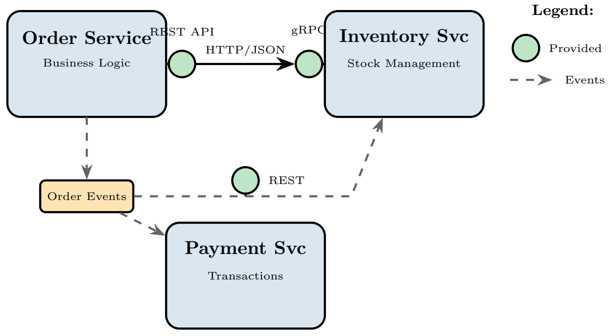
Architecture Documentation Series

Based on OpenAPI, AsyncAPI, IEEE Standards, and Industry Best Practices

December 4, 2025

Abstract

Interface documentation describes the contracts through which architectural elements interact—the APIs, events, protocols, and data formats that define system boundaries. Well-documented interfaces enable independent development, testing, and evolution of system components while ensuring interoperability and maintainability. This comprehensive guide establishes principles and practices for documenting all types of interfaces: REST APIs, GraphQL endpoints, gRPC services, message queues, event streams, file formats, and hardware protocols. The document covers interface taxonomy, operation specifications, data schemas, protocol details, error handling, quality of service, security requirements, versioning strategies, and governance processes. Whether designing public APIs, internal service contracts, or integration interfaces, this guide provides the foundation for clear, complete, and consumable interface documentation.



Contents

1 Introduction to Interface Documentation

1.1 Definition and Purpose

An interface is a boundary across which two architectural elements interact. Interface documentation describes what consumers need to know to successfully use an interface—the operations available, data formats expected, protocols used, and behaviors guaranteed.

Definition

Interface Documentation is the specification of the externally visible properties of an architectural element's boundary—the contract that defines how other elements can interact with it, including the syntax (operations, data types), semantics (behavior, constraints), and pragmatics (protocols, quality attributes) of that interaction.

Interface documentation serves several critical purposes. First, it enables **independent development** by allowing teams to build consumers and providers in parallel based on agreed contracts. Second, it supports **integration testing** by providing the specification against which implementations are validated. Third, it facilitates **evolution** by making change impact analysis possible through explicit contract versioning. Fourth, it enables **discovery** by helping developers find and understand available capabilities. Fifth, it ensures **interoperability** by precisely defining exchange formats and protocols.

1.2 The Cost of Poor Interface Documentation

Warning

Consequences of Inadequate Interface Documentation:
Integration failures: Teams discover incompatibilities late in development when integration is attempted, causing delays and rework.
Breaking changes: Without clear contracts, changes inadvertently break consumers, causing production incidents.
Duplicate implementations: Teams build redundant services because they cannot discover existing capabilities.
Support burden: Interface owners spend excessive time answering questions that documentation should address.
Security vulnerabilities: Unclear authentication and authorization requirements lead to inconsistent security implementations.
Performance problems: Undocumented rate limits and quotas cause cascading failures under load.

1.3 Interface Documentation in Context

Interface documentation complements the Element Catalog by providing detailed specifications for how elements interact. While the catalog describes what an element is and does, interface documentation describes how to interact with it.

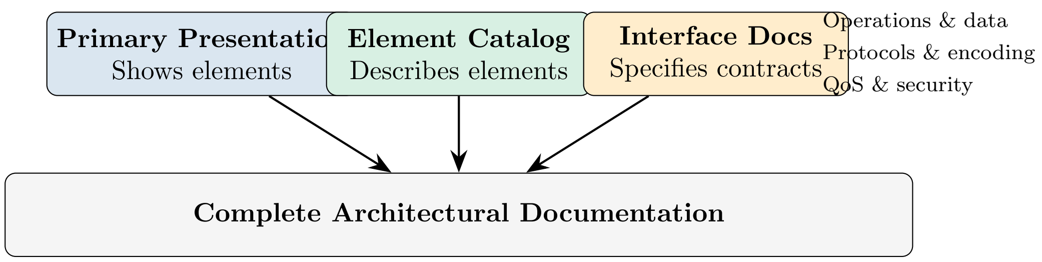


Figure 1: Interface Documentation within Architecture Documentation

1.4 Standards and Specifications

Interface documentation builds on established standards. OpenAPI (formerly Swagger) provides the specification for REST API documentation. AsyncAPI extends similar concepts to event-driven interfaces. gRPC/Protocol Buffers define service interfaces for RPC. GraphQL Schema Definition Language specifies GraphQL APIs. WSDL documents SOAP web services. IEEE 830 and ISO/IEC/IEEE 29148 provide general requirements specification guidance.

2 Interface Taxonomy

2.1 Interface Classification

Interfaces can be classified along several dimensions to guide documentation approach and depth.

2.1.1 By Direction

Table 1: Interface Classification by Direction

Direction	Description	Examples
Provided	Interface offered by element for others to consume	REST API; gRPC service; event publisher
Required	Interface that element needs from others	Database connection; external API client
Bidirectional	Two-way communication channel	WebSocket; message queue with reply

2.1.2 By Interaction Pattern

Table 2: Interface Classification by Interaction Pattern

Pattern	Characteristics	Technologies
Request-Response	Synchronous; caller waits for result	REST, gRPC, GraphQL
Fire-and-Forget	Asynchronous; no response expected	Message queue publish

Pattern	Characteristics	Technologies
Publish-Subscribe	One-to-many event distribution	Kafka, RabbitMQ, SNS
Streaming	Continuous data flow	gRPC streaming, WebSocket, SSE
Callback/Webhook	Asynchronous notification to caller	HTTP webhooks
Polling	Consumer periodically checks for updates	REST polling, long-polling

2.1.3 By Technology

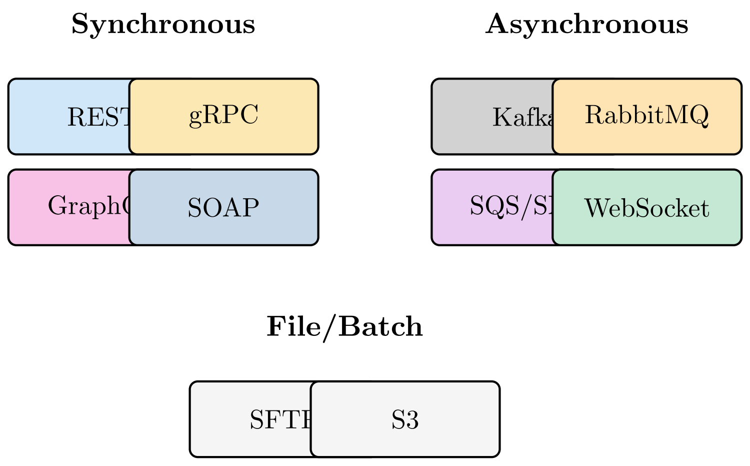


Figure 2: Interface Technologies by Category

2.1.4 By Audience

Table 3: Interface Classification by Audience

Audience	Description	Documentation Emphasis
Public	External developers; third parties	Comprehensive; tutorials; SDKs
Partner	Trusted external organizations	Detailed; access provisioning
Internal	Teams within organization	Essential details; assume context
Private	Same team/component	Minimal; may be implicit

2.2 Interface Style Comparison

Table 4: Interface Style Comparison

Aspect	REST	GraphQL	gRPC	Async Events
Protocol	HTTP	HTTP	HTTP/2	AMQP, Kafka
Encoding	JSON/XML	JSON	Protobuf	JSON, Avro, Protobuf
Contract	OpenAPI	SDL	.proto	AsyncAPI, Avro Schema
Coupling	Loose	Loose	Tight	Very loose
Latency	Medium	Medium	Low	Variable
Best For	Web APIs; CRUD	Flexible queries	Microservices	Event-driven

3 Interface Catalog

3.1 Purpose of the Catalog

The Interface Catalog provides a comprehensive inventory of all documented interfaces, enabling discovery and navigation. It serves as an index to detailed interface specifications.

3.2 Interface Registry

Table 5: Interface Registry

ID	Name	Owner	Type	Status	Description
IF-001	Order API	Order Svc	REST	**[STABLE]**	Order lifecycle management
IF-002	Order Events	Order Svc	Kafka	**[STABLE]**	Order state change events
IF-003	Inventory API	Inv Svc	gRPC	**[STABLE]**	Stock queries and reservations
IF-004	Payment API	Pay Svc	REST	**[STABLE]**	Payment processing
IF-005	Payment Webhooks	Pay Svc	Webhook	**[STABLE]**	Payment status callbacks
IF-006	User API	User Svc	REST	**[STABLE]**	User profile management
IF-007	Auth API	Auth Svc	REST	**[STABLE]**	Authentication endpoints
IF-008	Search API	Search Svc	REST	**[BETA]**	Product search queries

Continued on next page

ID	Name	Owner	Type	Status	Description
IF-009	Analytics Events	Analytics	Kafka	[STABLE]	User behavior events
IF-010	Notification API	Notify Svc	REST	[STABLE]	Send notifications
IF-011	File Upload API	Storage Svc	REST	[STABLE]	File upload/download
IF-012	GraphQL Gateway	API Gateway	GraphQL	[BETA]	Unified query interface

3.3 Interface Dependency Map

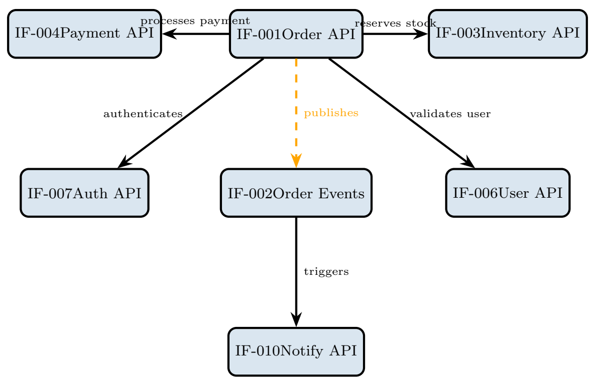


Figure 3: Interface Dependency Map

4 Documentation Conventions

4.1 Naming Conventions

Best Practice

Interface Naming Standards:
Interface IDs: Use pattern `IF-NNN` with sequential numbering.
API Names: Use descriptive names indicating domain and purpose (e.g., "Order Management API").
Endpoints: Use lowercase, hyphen-separated paths (e.g., `/order-items`).
Operations: Use verb-noun format for RPC-style (e.g., `CreateOrder`), noun for resources.
Events: Use past-tense verbs indicating what happened (e.g., `OrderCreated`).
Fields: Use camelCase for JSON, snake_case for databases.

4.2 Common Data Formats

4.2.1 Standard Response Envelope

Listing 1: Standard API Response Envelope

```
{
  "data": { ... },            // Response payload (on success)
  "meta": {
    "requestId": "uuid",      // Correlation ID for tracing
    "timestamp": "ISO8601",   // Response timestamp
    "version": "v1"           // API version
  },
  "pagination": {             // For list responses
    "page": 1,
    "pageSize": 20,
    "totalItems": 150,
    "totalPages": 8
  }
}
```

4.2.2 Standard Error Format

Listing 2: Standard Error Response

```
{
  "error": {
    "code": "VALIDATION_ERROR",     // Machine-readable code
    "message": "Invalid input",     // Human-readable message
    "details": [                    // Detailed field errors
      {
        "field": "email",
        "code": "INVALID_FORMAT",
        "message": "Must be valid email"
      }
    ],
    "requestId": "uuid",            // For support reference
    "documentation": "https://..."  // Link to docs
  }
}
```

4.3 HTTP Status Code Standards

Table 6: HTTP Status Code Usage

Code	Meaning	When to Use
200	OK	Successful GET, PUT, PATCH
201	Created	Successful POST creating resource
202	Accepted	Request accepted for async processing
204	No Content	Successful DELETE or PUT with no response body
400	Bad Request	Malformed request syntax or invalid parameters

Code	Meaning	When to Use
401	Unauthorized	Missing or invalid authentication
403	Forbidden	Authenticated but not authorized
404	Not Found	Resource does not exist
409	Conflict	Conflict with current state (e.g., duplicate)
422	Unprocessable	Validation error on well-formed request
429	Too Many Requests	Rate limit exceeded
500	Internal Error	Unexpected server error
502	Bad Gateway	Upstream service error
503	Service Unavailable	Temporarily unavailable (maintenance)
504	Gateway Timeout	Upstream service timeout

4.4 Authentication Standards

Table 7: Authentication Methods

Method	Use Case	Implementation
Bearer Token (JWT)	User authentication; API access	`Authorization: Bearer <token>`
API Key	Service-to-service; simple clients	`X-API-Key: <key>` header
OAuth 2.0	Delegated authorization	Authorization code or client credentials flow
mTLS	Service mesh; high security	Client certificate validation
HMAC Signature	Webhooks; request integrity	`X-Signature: <hmac>` header

5 REST API Documentation

5.1 REST API Specification Template

This section provides the complete template for documenting REST APIs.

IF-001: Order Management API

A. General Information

Interface ID:	IF-001
Name:	Order Management API
Owning Element:	Order Service (`SVC-ORDER-001`)
Interface Type:	Provided
Technology:	REST over HTTP/1.1, JSON
Base URL:	`https://api.example.com/v1/orders`
Version:	v1.3.0
Status:	**[STABLE]**
OpenAPI Spec:	`/specs/order-api-v1.yaml`

B. Responsibilities and Role

The Order Management API provides the primary interface for order lifecycle management. It enables clients to create orders, query order status, modify orders before fulfillment, and cancel orders. The API serves as the entry point for all order-related operations in the e-commerce platform.

Key Use Cases:

- Customer checkout: Create new order from shopping cart
- Order tracking: Query order status and history
- Order modification: Update shipping address before shipment
- Order cancellation: Cancel order with refund initiation
- Admin operations: List and search orders for support

Stakeholders:

- Web/Mobile frontend teams
- Customer service applications
- Partner integration systems
- Analytics and reporting systems

C. Operations

Endpoint	Description
POST `/orders`	Create a new order
GET `/orders`	List orders with filtering and pagination
GET `/orders/{id}`	Get order details by ID
PATCH `/orders/{id}`	Update order (limited fields)
DELETE `/orders/{id}`	Cancel order
GET `/orders/{id}/items`	Get order line items
POST `/orders/{id}/items`	Add item to order (draft only)
GET `/orders/{id}/history`	Get order status history
POST `/orders/{id}/actions/confirm`	Confirm pending order
POST `/orders/{id}/actions/ship`	Mark order as shipped

6 Event Interface Documentation

6.1 Event Interface Specification Template

IF-002: Order Events

A. General Information

Interface ID:	IF-002
Name:	Order Events
Owning Element:	Order Service (`SVC-ORDER-001`)
Interface Type:	Provided (Publisher)
Technology:	Apache Kafka
Topic:	`orders.events.v1`
Version:	v1.2.0
Status:	**[STABLE]**
AsyncAPI Spec:	`/specs/order-events-v1.yaml`

B. Responsibilities and Role

The Order Events interface publishes domain events for all significant order state changes. It enables downstream services to react to order lifecycle events without tight coupling to the Order Service.

Key Use Cases:

- Inventory: Release reservations on cancellation
- Payments: Initiate refunds on cancellation
- Notifications: Send customer communications
- Analytics: Track order funnel metrics
- Search: Update order search index

C. Event Types

Event Type	**Trigger**	**Key Data**
`OrderCreated`	New order submitted	Full order details
`OrderConfirmed`	Payment successful	Order ID, confirmation time
`OrderUpdated`	Order modified	Changed fields, version
`OrderShipped`	Shipment created	Tracking number, carrier
`OrderDelivered`	Delivery confirmed	Delivery timestamp
`OrderCancelled`	Order cancelled	Cancellation reason
`OrderRefunded`	Refund processed	Refund amount, method

D. Event Schema

Event Envelope

```
{
  "eventId": "evt_abc123xyz",
  "eventType": "OrderCreated",
  "eventVersion": "1.0",
```

7 gRPC Interface Documentation

7.1 gRPC Service Specification

IF-003: Inventory Service (gRPC)

A. General Information

Interface ID:	IF-003
Name:	Inventory Service
Owning Element:	Inventory Service (`SVC-INV-001`)
Interface Type:	Provided
Technology:	gRPC over HTTP/2
Proto Package:	`com.example.inventory.v1`
Proto File:	`/protos/inventory/v1/inventory.proto`
Version:	v1
Status:	**[STABLE]**

B. Service Definition

Listing 4: inventory.proto

```
syntax = "proto3";

package com.example.inventory.v1;

service InventoryService {
  // Check stock availability for products
  rpc CheckAvailability(CheckAvailabilityRequest)
      returns (CheckAvailabilityResponse);

  // Reserve stock for an order
  rpc ReserveStock(ReserveStockRequest)
      returns (ReserveStockResponse);

  // Release previously reserved stock
  rpc ReleaseReservation(ReleaseReservationRequest)
      returns (ReleaseReservationResponse);

  // Commit reservation (deduct from inventory)
  rpc CommitReservation(CommitReservationRequest)
      returns (CommitReservationResponse);

  // Stream inventory updates
  rpc WatchInventory(WatchInventoryRequest)
      returns (stream InventoryUpdate);
}
```

C. Message Types

```
message CheckAvailabilityRequest {
  repeated string product_ids = 1;
  string warehouse_id = 2;  // Optional; all warehouses if empty
}

message CheckAvailabilityResponse {
  repeated ProductAvailability availability = 1;
```

8 Webhook Interface Documentation

IF-005: Payment Webhooks

A. General Information

Interface ID:	IF-005
Name:	Payment Webhooks
Owning Element:	Payment Service (`SVC-PAY-001`)
Interface Type:	Provided (Outbound)
Technology:	HTTP POST callbacks
Version:	v1
Status:	**[STABLE]**

B. Webhook Events

Event Type	**Description**
`payment.succeeded`	Payment successfully captured
`payment.failed`	Payment attempt failed
`payment.refunded`	Refund processed
`payment.disputed`	Chargeback initiated

C. Webhook Payload

```
{
  "id": "whk_evt_abc123",
  "type": "payment.succeeded",
  "created": "2024-01-15T10:30:00Z",
  "data": {
    "paymentId": "pay_xyz789",
    "orderId": "ord_def456",
    "amount": 11447,
    "currency": "USD",
    "status": "succeeded",
    "paymentMethod": {
      "type": "card",
      "last4": "4242",
      "brand": "visa"
    }
  }
}
```

D. Signature Verification

All webhooks include HMAC signature for verification:

```
# Headers sent with webhook
X-Webhook-Signature: sha256=abc123...
X-Webhook-Timestamp: 1705315800
X-Webhook-Id: whk_evt_abc123
```

Verification Steps:

1. Extract timestamp and signature from headers
2. Reject if timestamp > 5 minutes old (replay protection)

9 Interface Governance

9.1 Interface Lifecycle

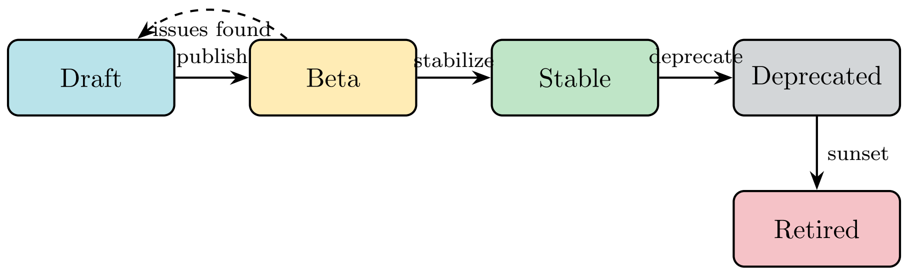


Figure 4: Interface Lifecycle States

9.2 Breaking Change Policy

Warning

Breaking Changes Require:

- New major version (v1 → v2)
- 12 months deprecation notice for public APIs
- 6 months for internal APIs
- Migration guide documentation
- Consumer notification

9.3 Interface Review Checklist

☐ All operations documented with request/response schemas

☐ Error codes and handling documented

☐ Authentication and authorization specified

☐ Rate limits and quotas defined

☐ Versioning strategy documented

☐ Examples provided for all operations

☐ Machine-readable spec (OpenAPI/AsyncAPI) available

☐ Contract tests implemented

☐ Traceability to requirements established

10 Appendix A: OpenAPI Template

Listing 5: OpenAPI Template (openapi.yaml)

```yaml
openapi: 3.1.0
info:
  title: Service Name API
  version: 1.0.0
  description: |
    Detailed description of the API.
  contact:
    name: API Support
    email: api-support@example.com

servers:
  - url: https://api.example.com/v1
    description: Production

security:
  - bearerAuth: []

paths:
  /resources:
    get:
      summary: List resources
      operationId: listResources
      tags: [Resources]
      parameters:
        - name: page
          in: query
          schema:
            type: integer
            default: 1
      responses:
        '200':
          description: Success
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ResourceList'

components:
  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
  schemas:
    ResourceList:
      type: object
      properties:
        data:
          type: array
          items:
            $ref: '#/components/schemas/Resource'
```

11 Appendix B: Glossary

API	Application Programming Interface; contract for programmatic interaction
AsyncAPI	Specification for documenting event-driven APIs
Endpoint	A specific URL path that accepts requests
gRPC	Google Remote Procedure Call; high-performance RPC framework
Idempotency	Property where multiple identical requests have same effect as one
OpenAPI	Specification for documenting REST APIs (formerly Swagger)
Payload	The data content of a request or response
Protocol	Rules governing communication between systems
Rate Limit	Maximum number of requests allowed per time period
Schema	Definition of data structure and types
Webhook	HTTP callback triggered by an event

12 Appendix C: References

1. OpenAPI Initiative. (2023). *OpenAPI Specification 3.1.* https://spec.openapis.org
2. AsyncAPI Initiative. (2023). *AsyncAPI Specification 2.6.* https://www.asyncapi.com
3. Google. (2023). *API Design Guide.* https://cloud.google.com/apis/design
4. Masse, M. (2011). *REST API Design Rulebook.* O'Reilly Media.
5. Clements, P., et al. (2010). *Documenting Software Architectures.* Addison-Wesley.
6. Lauret, A. (2019). *The Design of Web APIs.* Manning Publications.
7. gRPC Authors. (2023). *gRPC Documentation.* https://grpc.io/docs
8. Fielding, R. (2000). "Architectural Styles and the Design of Network-based Software Architectures." Doctoral dissertation, UC Irvine.