

# Table of Contents

|  |   |
|--|---|
| 1. Glossary .....                      | 1 |
| 1.1. Core Concepts .....               | 1 |
| 1.2. Multi-Tenancy .....               | 1 |
| 1.3. Security .....                    | 2 |
| 1.4. Data and Persistence .....        | 2 |
| 1.5. REST and APIs .....               | 2 |
| 1.6. Framework Components .....        | 3 |
| 1.7. Annotations .....                 | 3 |
| 1.8. External Integrations .....       | 3 |
| 1.9. Ontology (optional feature) ..... | 4 |

# 1. Glossary

## 1.1. Core Concepts

### **BaseModel**

Abstract base class for all Quantum entities, providing DataDomain, audit fields, ID management, and validation hooks.

### **DataDomain**

Scoping information attached to every model (tenantId, orgRefName, ownerId, etc.) that enables multi-tenant isolation and controlled sharing.

### **DomainContext**

Runtime context representing the current execution scope (tenant, org, user, functional area/domain, action) used by repositories and security rules.

### **Functional Area**

Broad business capability grouping (e.g., Catalog, Collaboration, Identity) used for organizing models and security policies.

### **Functional Domain**

Specific entity type within a functional area (e.g., Product within Catalog, Shipment within Collaboration) used for fine-grained permissions.

### **RuleContext**

Policy evaluation engine that determines allowed actions and contributes data filters based on identity, functional area/domain, and business rules.

## 1.2. Multi-Tenancy

### **Tenant**

Logical partition representing a customer or organization in a multi-tenant system, identified by tenantId.

### **Organization (Org)**

Business unit within a tenant, identified by orgRefName, enabling sub-tenant grouping and sharing.

### **Realm**

Database or data partition, often corresponding to a MongoDB database, that can be selected per request via X-Realm header.

### **Shared Domain**

Data domain accessible across multiple tenants (e.g., public catalogs, partner directories) as opposed to tenant-isolated data.

## 1.3. Security

### Principal

Authenticated user identity with associated roles, tenant membership, and permissions.

### Permission Rule

Declarative policy that matches requests (URL, method, headers, body) and decides ALLOW/DENY with optional data filters.

### Access Resolver

Plugin that computes dynamic access lists (e.g., customer IDs a user can see) for use in permission filters.

### Impersonation

Acting as another user identity for troubleshooting or administrative purposes, controlled by scripts and realm restrictions.

## 1.4. Data and Persistence

### MorphiaRepo

Repository interface extending MongoDB operations with Quantum's DataDomain filtering, validation, and audit capabilities.

### EntityReference

Lightweight reference object containing ID, type, refName, and displayName, used for foreign keys without full object loading.

### StateGraph

Finite state machine definition for model fields, enforcing valid states and transitions (e.g., Order: Draft → Processing → Shipped).

### Completion Task

Persistent work item with status tracking, used for checklists and long-running processes with audit trails.

## 1.5. REST and APIs

### BaseResource

Abstract REST resource class providing consistent CRUD endpoints (find, get, list, save, update, delete) with automatic security and validation.

### Query Language

ANTLR-based filter syntax used across all list endpoints, permission rules, and access resolvers for consistent data querying.

### UIActions

List of actions a user can perform on a specific entity instance, computed based on entity state and user permissions.

## CSV Import/Export

Built-in endpoints for bulk data operations with validation, preview sessions, and error handling.

## 1.6. Framework Components

### ValidationInterceptor

Morphia interceptor that runs Jakarta Bean Validation and defaults DataDomain before persistence.

### SecurityFilter

JAX-RS filter that builds security context (PrincipalContext, ResourceContext) and evaluates permissions for each request.

### DataDomainResolver

Service that determines which DataDomain to assign to new entities based on functional area/domain policies.

### Migration

Versioned database schema and data changes managed by Quantum's migration framework with changeset tracking.

## 1.7. Annotations

### @FunctionalMapping

Class-level annotation declaring a model's functional area and domain, replacing legacy bmFunctionalArea() methods.

### @FunctionalAction

Method-level annotation specifying the action performed by a REST endpoint when it differs from HTTP verb defaults.

### @TrackReferences

Field annotation on @Reference fields that maintains back-reference sets for referential integrity checking.

### @RegisterForReflection

Quarkus annotation ensuring classes are available for reflection in native images.

## 1.8. External Integrations

### JWT Provider

Authentication module that validates JSON Web Tokens and populates security context with user identity and roles.

### OIDC Integration

OpenID Connect support for enterprise identity providers like Keycloak, Auth0, and AWS Cognito.

## Feature Flags

Configuration-driven capability toggles with targeting rules for gradual rollouts and A/B testing.

## Postmark Integration

Email service integration for transactional messaging with template support.

## 1.9. Ontology (optional feature)

### Ontology

A formal description of concepts (classes) and their relationships (properties/predicates) with rules (e.g., property chains) that allow inferring implied facts. See [Ontologies in Quantum](#).

### Predicate / Property

A named relationship (e.g., `placedBy`, `memberOf`, `orderShipsToRegion`). Predicates connect source and destination IDs and can be declared inverse or transitive.

### Property Chain

A rule of the form  $p \sqcap q \Rightarrow r$  meaning if  $(A \dashv p \dashv B)$  and  $(B \dashv q \dashv C)$ , infer  $(A \dashv r \dashv C)$ . Chains make common traversals first-class and fast.

### Edge (materialized)

A persisted tuple (`tenantId`, `src`, `p`, `dst`) representing a relationship, often inferred from rules. Edges live in a dedicated collection and power single-hop queries and policies.

### Ontology Materializer

A component that computes inferred edges for an entity snapshot and upserts them to the edges collection. Keeps edges fresh as data changes.

### ListQueryRewriter

A helper that turns semantic constraints into efficient Mongo queries (e.g., `id IN srcIdsByDst`). Integrates with permission rules and Morphia repos.