**File Review**

**File:** Domain/Entities/XeroOAuth2Token.cs  
**Layer/Type:** Domain – Entity  
**Status:** Reviewed  
**Tokens (approx.):** ~1.6k

**🧾 ELI5**

This file defines a **domain entity** that represents an OAuth2 token from Xero. It securely holds the access/refresh tokens, expiry dates, tenant associations, and provides helper methods to check if the token is expired or needs rotation.

**🎯 Purpose and Role**

* Encapsulates **Xero OAuth2 token lifecycle** within the Domain layer.
* Provides immutability and thread safety by design.
* Serves as the **canonical representation of tokens** passed to Application/Infrastructure layers.
* Enforces **domain rules** around:
  + Access token must exist.
  + Refresh token must exist.
  + Tenants list must be consistent.
  + Expiry and refresh issue times are tracked.

**🔍 Detailed Breakdown**

* **Properties:**
  + AccessToken (string, required).
  + RefreshToken (string, required).
  + ExpiresAtUtc (DateTime).
  + IdToken (optional string).
  + IssuedAtUtc (UTC when issued).
  + RefreshIssuedAtUtc (UTC when refresh was issued).
  + RefreshedAtUtc (nullable, last refresh timestamp).
  + Tenants (read-only collection of XeroTenantInfo).
* **Constructors:**
  + Primary constructor validates inputs (throws ArgumentNullException for null tokens).
  + Initializes tenants safely (empty read-only collection if none provided).
* **Factory Method:**
  + WithTenants(newTenants) – returns a new instance with updated tenant info.
* **Derived Properties/Methods:**
  + IsAccessTokenExpired → checks expiry against current UTC time.

**⚠️ Error Handling & Validation**

* Null checks enforced for accessToken and refreshToken.
* tenants gracefully handled (null → empty list).
* Throws ArgumentNullException when critical parameters are missing.
* Good use of immutability to reduce mutation risks.

**🔐 Security Review**

* Tokens stored as plain strings **in-memory** (acceptable at domain level).
* No logging of secrets inside this file (✅ good).
* Sensitive values (AccessToken, RefreshToken, IdToken) are exposed as public getters – safe within Domain but requires **care when logging** at higher layers.
* Conforms with project’s decision to manage tokens securely via Key Vault/environment variables.

**⚡ Performance & Reliability**

* Token validity checks are O(1).
* No heavy computation or I/O here.
* Thread-safety ensured by immutability (records + read-only collections).
* WithTenants cloning pattern is reliable but could be costly if tenants list grows very large (likely negligible).

**📊 Observability**

* No logging or tracing in Domain layer (correct – belongs in Application/Infrastructure).

**🧪 Testability & Coverage**

Suggested test cases:

1. **Positive:** Create token with valid values → verify properties set correctly.
2. **Negative:** Pass null for access token → expect ArgumentNullException.
3. **Negative:** Pass null for refresh token → expect ArgumentNullException.
4. **Edge:** Expiry time in past → IsAccessTokenExpired = true.
5. **Edge:** Expiry time in future → IsAccessTokenExpired = false.
6. **Update tenants:** Call WithTenants() → ensure new instance has tenants, old remains unchanged.

**🧹 Code Smells**

* **Info:** Exposes raw token strings publicly – must ensure no accidental logging elsewhere.
* **Low:** Multiple overloads/constructors could be consolidated with optional parameters, but current explicitness aids clarity.

**🔧 Refactoring Suggestions**

* **Introduce value objects** for AccessToken, RefreshToken (instead of string) to enforce stronger typing.
  + Effort: **Medium**, Priority: **3**.
* Consider **obfuscating token ToString()** (e.g., AccessToken.Substring(0,4)+"...") for safer debugging.
  + Effort: **Small**, Priority: **4**.
* Add **XML doc comments** to each property for domain-level clarity (esp. RefreshedAtUtc, IssuedAtUtc).
  + Effort: **Quick Win**, Priority: **2**.

**📜 Contracts & Compatibility**

* This entity defines part of the **public domain model**.
* Changing constructor signatures or property names would break Application/Infrastructure contracts.
* WithTenants() is the only safe mutation path – critical to preserve for compatibility.

**🗄️ Data Model Notes**

* Maps directly to persisted token records (likely in PostgreSQL via Dapper).
* ExpiresAtUtc + RefreshIssuedAtUtc align with ADR strategy for refresh windows.
* Tenants association aligns with Xero’s multi-tenant model.

**✅ Confidence**

**High** – Full file was available and cross-checked against project architectural decisions and guidelines.

**File Review**

**File:** Domain/Entities/TokenRecord.cs  
**Layer/Type:** Domain – Persistent Entity  
**Status:** Reviewed  
**Tokens (approx.):** ~300

**🧾 ELI5**

This file defines a **persistable record** that represents an encrypted OAuth2 token for a specific user and Xero tenant. It is designed to map directly to a database table.

**🎯 Purpose and Role**

* Provides the **persistence model** for Xero OAuth2 tokens.
* Encapsulates:
  + Tenant ID (Xero organisation).
  + User ID (application user).
  + Encrypted access and refresh tokens.
  + Expiry and refresh timestamps.
* Acts as the bridge between:
  + XeroOAuth2Token (domain representation).
  + Infrastructure persistence (Dapper → PostgreSQL).

**🔍 Detailed Breakdown**

* Declared as a **C# record** for immutability and value semantics.
* Properties:
  + TenantId (Guid) → Xero tenant/organisation identifier.
  + UserId (Guid) → RoadmApp application user identifier.
  + EncryptedAccessToken (string).
  + EncryptedRefreshToken (string).
  + ExpiresAtUtc (DateTime).
  + RefreshIssuedAtUtc (DateTime).
* Designed for **direct database mapping** (likely a table TokenRecords).

**⚠️ Error Handling & Validation**

* No explicit validation here (assumes enforcement at higher layers).
* Potential risks if EncryptedAccessToken or EncryptedRefreshToken are empty strings.
* Expiry date integrity must be managed at service layer.

**🔐 Security Review**

* Correctly stores **only encrypted tokens** (not raw).
* Secrets are never exposed as plain values here.
* Good separation of **domain usage vs persistence** (encrypted here, decrypted in services only).
* Must ensure strong encryption/decryption logic exists in Infrastructure layer.

**⚡ Performance & Reliability**

* Record is lightweight and efficient for persistence.
* GUID identifiers ensure uniqueness and fast indexing.
* Expiry tracking avoids unnecessary refreshes.

**📊 Observability**

* No logging inside this entity (correct).
* Observability is handled at service level.

**🧪 Testability & Coverage**

Suggested test cases:

1. **Mapping Test:** Insert and retrieve TokenRecord from DB → ensure all fields round-trip correctly.
2. **Integrity Test:** Verify encrypted fields are not empty.
3. **Edge Case:** Expired ExpiresAtUtc vs future date → service should interpret correctly.

**🧹 Code Smells**

* **Info:** Lacks explicit null-checking – relies on DB and encryption service to enforce correctness.
* **Low:** Using string for encrypted tokens means any value can be passed; a stronger type could help.

**🔧 Refactoring Suggestions**

* Introduce **value objects** for EncryptedAccessToken and EncryptedRefreshToken instead of plain string.
  + Effort: **Medium**, Priority: **3**.
* Add **data annotations** (e.g., [Required]) if used in ORM mapping to prevent null values.
  + Effort: **Small**, Priority: **2**.
* Consider a **factory method** to ensure tokens are always encrypted before creating this record.
  + Effort: **Medium**, Priority: **4**.

**📜 Contracts & Compatibility**

* Represents a **database contract** → changes here require schema migration.
* Must remain backward compatible with existing persistence layer.
* Breaking changes to field names/types will require DB updates + Infrastructure refactoring.

**🗄️ Data Model Notes**

* Likely corresponds to table with composite key (TenantId, UserId).
* Expiry and refresh timestamps enable **refresh window strategy** documented in ADR.
* Encryption ensures compliance with security guidelines.

**✅ Confidence**

**High** – Entire file available, mapped against architecture and ADRs.

**File Review**

**File:** Domain/Entities/Organisation.cs  
**Layer/Type:** Domain – Entity  
**Status:** Reviewed  
**Tokens (approx.):** ~120

**🧾 ELI5**

This file defines a simple entity for an **organisation (tenant)** within RoadmApp. It holds an ID, a name, and the creation timestamp.

**🎯 Purpose and Role**

* Represents an **organisation/tenant** in the system.
* Forms the basis for associating users, roles, and tokens to specific organisations.
* Lightweight entity with minimal properties (expandable in future).

**🔍 Detailed Breakdown**

* **Properties:**
  + OrganisationId (Guid) → unique identifier.
  + Name (string) → default empty string; user-supplied organisation name.
  + CreatedAt (DateTime) → record creation timestamp.
* **Implementation notes:**
  + Class instead of record → may be mutable (currently has get; set;).
  + Default Name = string.Empty prevents null issues.

**⚠️ Error Handling & Validation**

* No validation logic.
* Risk: Name can be empty string (valid in DB?).
* CreatedAt is not initialized in constructor – relies on external code to set it.

**🔐 Security Review**

* No sensitive fields.
* Safe for logging.

**⚡ Performance & Reliability**

* Minimal overhead.
* GUIDs ensure uniqueness.
* No performance concerns.

**📊 Observability**

* No logging (correct for Domain).

**🧪 Testability & Coverage**

Suggested test cases:

1. **Positive:** Instantiate with valid values → properties set correctly.
2. **Negative:** Set Name = "" → verify if allowed.
3. **Edge:** Ensure OrganisationId uniqueness enforced at DB layer.

**🧹 Code Smells**

* **Low:** Mutable entity – could allow accidental changes.
* **Info:** CreatedAt not auto-initialized.

**🔧 Refactoring Suggestions**

* Convert to **record** or make properties init-only to enforce immutability.
  + Effort: **Small**, Priority: **3**.
* Initialize CreatedAt with DateTime.UtcNow in constructor.
  + Effort: **Quick Win**, Priority: **4**.
* Add domain validation for Name (non-empty, max length).
  + Effort: **Small**, Priority: **2**.

**📜 Contracts & Compatibility**

* Entity is part of the Domain model → likely maps to an Organisations table.
* Adding validation or immutability may affect persistence and mapping layers.

**🗄️ Data Model Notes**

* OrganisationId → primary key.
* CreatedAt → should default to CURRENT\_TIMESTAMP in DB schema if not set in code.
* Name should probably be constrained (e.g., NOT NULL, length ≤ 255).

**✅ Confidence**

**High** – Full file available; analysis consistent with architectural guidelines.

**File Review**

**File:** Domain/Entities/PasswordResetToken.cs  
**Layer/Type:** Domain – Entity  
**Status:** Reviewed  
**Tokens (approx.):** ~180

**🧾 ELI5**

This file defines a **password reset token** entity used when a user requests a password reset. It contains the user ID, a hashed token, expiry date, and usage tracking fields.

**🎯 Purpose and Role**

* Provides a secure way to handle **password reset flows**.
* Designed to store only the **hashed version** of the reset token (never plaintext).
* Contains lifecycle metadata:
  + When created.
  + When it expires.
  + When (if ever) it was used.
* Forms the persistent record for password reset requests.

**🔍 Detailed Breakdown**

* **Properties:**
  + Id (Guid) – unique identifier of reset token.
  + UserId (Guid) – user requesting the reset.
  + TokenHash (string) – hashed reset token (default = string.Empty).
  + ExpiresAt (DateTime) – expiry timestamp.
  + CreatedAt (DateTime) – creation timestamp.
  + UsedAt (nullable DateTime) – marks if/when the token was used.
* **Implementation Notes:**
  + Class instead of record – mutable via setters.
  + Default constructor allows uninitialized/empty TokenHash.

**⚠️ Error Handling & Validation**

* No built-in validation for:
  + Non-empty TokenHash.
  + Ensuring ExpiresAt > CreatedAt.
* Relies on higher-level logic to enforce constraints.

**🔐 Security Review**

* ✅ Stores **hashed tokens only** (good security practice).
* ❌ Potential issue: no enforcement that TokenHash is non-empty.
* ❌ If not salted/pepper-hashed at Infrastructure layer, could be vulnerable to rainbow table attacks.
* ❌ UsedAt is mutable – could be overwritten accidentally.

**⚡ Performance & Reliability**

* Entity is lightweight.
* Expiry handling is straightforward.
* Proper indexing on UserId + ExpiresAt will be needed in DB for efficient lookups.

**📊 Observability**

* No logging here (correct for Domain).

**🧪 Testability & Coverage**

Suggested test cases:

1. **Positive:** Create token → all properties set as expected.
2. **Negative:** Ensure TokenHash = "" is rejected at service layer.
3. **Expiry:** Verify token considered invalid if ExpiresAt < DateTime.UtcNow.
4. **Usage:** Set UsedAt → ensure token cannot be reused.

**🧹 Code Smells**

* **Medium:** TokenHash default = string.Empty (unsafe default).
* **Low:** Mutable properties allow accidental overwrites.

**🔧 Refactoring Suggestions**

* Enforce immutability via record or init-only properties.
  + Effort: **Small**, Priority: **3**.
* Add **constructor requiring valid TokenHash + ExpiresAt**.
  + Effort: **Small**, Priority: **4**.
* Consider MarkAsUsed() method instead of public setter for UsedAt.
  + Effort: **Medium**, Priority: **4**.

**📜 Contracts & Compatibility**

* Represents **persistent model** for password reset workflow.
* Changing property names would break DB schema mappings.
* Migration needed if introducing stronger token constraints.

**🗄️ Data Model Notes**

* Id → primary key.
* Likely indexed by UserId.
* ExpiresAt should be checked at DB + service level.
* TokenHash column should be NOT NULL.

**✅ Confidence**

**High** – File is small, self-contained, and clearly aligned with domain responsibilities.

**File Review**

**File:** Domain/Entities/PollingSetting.cs  
**Layer/Type:** Domain – Entity  
**Status:** Reviewed  
**Tokens (approx.):** ~330

**🧾 ELI5**

This file defines how an **organisation’s data polling configuration** is stored. It includes the organisation ID, a polling schedule, and an optional fixed run time.

**🎯 Purpose and Role**

* Represents a **per-organisation polling configuration**.
* Used to control **how and when data is retrieved from Xero** (or other external systems).
* Domain-level configuration → independent of storage or UI.

**🔍 Detailed Breakdown**

* **Properties:**
  + OrganisationId (Guid) → link to owning organisation.
  + PollingSchedule (string, default string.Empty) → defines schedule (cron-like format, or similar).
  + RunTime (nullable TimeSpan) → optional fixed execution time within a day.
* **Implementation Notes:**
  + Class with mutable properties (get; set;).
  + No validation to enforce cron expression correctness or time range.
  + Intended for interpretation by Application/Infrastructure layers (likely background jobs).

**⚠️ Error Handling & Validation**

* No validation on PollingSchedule (invalid strings could be stored).
* RunTime could conflict with PollingSchedule if both are set.
* Depends on higher-level logic to enforce constraints.

**🔐 Security Review**

* No sensitive data stored.
* Safe for logging.

**⚡ Performance & Reliability**

* Lightweight configuration object.
* Reliability depends on Application layer correctly interpreting PollingSchedule and RunTime.
* Poorly validated schedules could lead to missed/duplicated polling.

**📊 Observability**

* No logging here (correct for Domain).
* Polling execution should be logged at service level.

**🧪 Testability & Coverage**

Suggested test cases:

1. **Positive:** Set PollingSchedule = "0 0 \* \* \*" (valid cron) → stored correctly.
2. **Negative:** Set PollingSchedule = "INVALID" → higher layer should reject.
3. **Edge:** RunTime = null → schedule-only polling works.
4. **Edge:** RunTime = 23:59:59 → validate day-boundary edge cases.
5. **Conflict:** Both RunTime and PollingSchedule set → ensure rules are defined.

**🧹 Code Smells**

* **Medium:** PollingSchedule as free-form string (no enforcement).
* **Low:** Mutable properties may allow accidental runtime modification.

**🔧 Refactoring Suggestions**

* Introduce a **value object for PollingSchedule** with built-in validation.
  + Effort: **Medium**, Priority: **4**.
* Consider record or init-only properties for immutability.
  + Effort: **Small**, Priority: **3**.
* Clarify contract: is RunTime *in addition to* or *instead of* PollingSchedule?
  + Effort: **Quick Win**, Priority: **2**.

**📜 Contracts & Compatibility**

* Likely persisted in DB table PollingSettings.
* Changes to PollingSchedule format would affect job scheduler logic.
* Must remain backward compatible if schedules are stored as strings.

**🗄️ Data Model Notes**

* OrganisationId → foreign key to Organisations.
* PollingSchedule → string column (no DB-level validation).
* RunTime → nullable, likely stored as SQL time.

**✅ Confidence**

**High** – Entire file available, straightforward design, consistent with polling features described in project docs.

**File Review**

**File:** Domain/Entities/RoadmappRole.cs  
**Layer/Type:** Domain – Entity  
**Status:** Reviewed  
**Tokens (approx.):** ~300

**🧾 ELI5**

This file defines a **role entity** for the RoadmApp application. It represents a named role (e.g., Administrator, Standard User) with optional descriptions and type, used for authorisation.

**🎯 Purpose and Role**

* Encapsulates **application roles** for user authorisation.
* Defines **permissions scope** within the domain layer.
* Works with RoadmappUser and UserOrgRole to enforce access control.
* Provides metadata about each role (ID, name, description, type).

**🔍 Detailed Breakdown**

* **Properties:**
  + RoleId (Guid) → primary key.
  + RoleName (string, default string.Empty) → unique role identifier (e.g., “Admin”).
  + Description (string, default string.Empty) → human-readable explanation.
  + RoleType (string, default string.Empty) → indicates application or organisation scope.
* **Implementation Notes:**
  + Class is mutable via set.
  + No constraints to enforce uniqueness of RoleName.
  + Serves as a **core domain entity** – not tied to Infrastructure.

**⚠️ Error Handling & Validation**

* No validation rules for:
  + Required RoleName.
  + Enforcing max length or restricted characters.
  + Ensuring RoleType is from a valid set (enum would be preferable).

**🔐 Security Review**

* Roles are critical for **authorisation decisions**.
* Current implementation allows free-form strings → risk of misconfiguration or typo-based privilege escalation.
* Stronger typing (enums/constants) would mitigate risks.

**⚡ Performance & Reliability**

* Lightweight entity.
* Reliability depends on proper enforcement of role mappings in DB and services.
* Risk if multiple roles with same RoleName exist without DB constraint.

**📊 Observability**

* No logging in entity (correct).
* Audit logging for role assignments should exist in Application layer.

**🧪 Testability & Coverage**

Suggested test cases:

1. **Positive:** Create RoadmappRole with valid values → properties set correctly.
2. **Negative:** Attempt empty RoleName → should be rejected at service level.
3. **Edge:** RoleType = "Org" vs "App" → enforce valid scope.
4. **Duplicate:** Two roles with same RoleName → should fail at DB/service.

**🧹 Code Smells**

* **Medium:** Free-form RoleType string (should be enum).
* **Low:** Mutable setters allow accidental modifications.
* **Info:** No enforcement of uniqueness on RoleName.

**🔧 Refactoring Suggestions**

* Convert RoleType to **enum** (e.g., Application, Organisation).
  + Effort: **Medium**, Priority: **4**.
* Add domain validation for non-empty RoleName.
  + Effort: **Small**, Priority: **3**.
* Consider record or init-only properties for immutability.
  + Effort: **Small**, Priority: **3**.

**📜 Contracts & Compatibility**

* Represents **authorisation contract** in Domain model.
* Any changes to RoleName values must be coordinated with Application + Infrastructure layers.
* Likely mapped to Roles table → DB constraints should enforce uniqueness.

**🗄️ Data Model Notes**

* RoleId → primary key.
* RoleName should be unique + indexed.
* RoleType should ideally be constrained to enum-like set.

**✅ Confidence**

**High** – Entire file reviewed, consistent with security/authorisation design principles.

**File Review**

**File:** Domain/Entities/RoadmappUser.cs  
**Layer/Type:** Domain – Entity  
**Status:** Reviewed  
**Tokens (approx.):** ~400

**🧾 ELI5**

This file defines the **user account entity** for RoadmApp. It stores identifiers, credentials, and profile data like email and full name, along with status (active/inactive) and an optional Xero user link.

**🎯 Purpose and Role**

* Core **user identity model** within the domain.
* Used for authentication, authorisation, and general user management.
* Maps to persistence for user accounts.
* Provides essential attributes required for login, account status, and integration with Xero.

**🔍 Detailed Breakdown**

* **Properties:**
  + UserId (Guid) → primary key.
  + Username (string, default "") → unique login identifier.
  + Email (string, default "") → contact + identity field.
  + PasswordHash (string, default "") → securely stored hash (not plaintext).
  + IsActive (bool) → account status.
  + FullName (string?, optional).
  + XeroUserId (string?, optional) → maps local user to Xero identity.
* **Implementation Notes:**
  + Uses mutable properties (get; set;).
  + Default empty strings may allow partially constructed objects.
  + No validation for unique username/email.

**⚠️ Error Handling & Validation**

* No built-in validation. Risks include:
  + Empty Username, Email, PasswordHash.
  + Weak or malformed email addresses.
  + Password hashes could be empty unless enforced elsewhere.

**🔐 Security Review**

* ✅ Stores **hashed password** (not plaintext).
* ❌ Uses plain string for password hash (risks accidental logging/memory exposure).
* ❌ No enforcement of **hashing standard** (bcrypt, PBKDF2, Argon2).
* Email and Username must be protected against enumeration attacks in login workflows.

**⚡ Performance & Reliability**

* Lightweight domain entity.
* Password hash verification handled externally.
* DB constraints needed for UserId, Username, Email uniqueness.

**📊 Observability**

* No logging here (correct).
* Login failures and account status events should be logged in Application layer.

**🧪 Testability & Coverage**

Suggested test cases:

1. **Positive:** Create user with valid username, email, hash → properties set correctly.
2. **Negative:** Empty Username or Email → should be rejected at service layer.
3. **Negative:** Empty PasswordHash → must not be persisted.
4. **Edge:** Inactive user → ensure prevented from login.
5. **Integration:** Map XeroUserId → validate consistent with Xero user identity.

**🧹 Code Smells**

* **Medium:** Mutable entity → risks accidental overwrites.
* **Medium:** Password hash as plain string → should be protected by value object/secure type.
* **Low:** Lack of validation for Email format.

**🔧 Refactoring Suggestions**

* Convert to record or init-only properties for immutability.
  + Effort: **Small**, Priority: **3**.
* Introduce **value objects** for Username, Email, PasswordHash.
  + Effort: **Medium**, Priority: **4**.
* Add constructor requiring essential fields (Username, Email, PasswordHash).
  + Effort: **Small**, Priority: **4**.

**📜 Contracts & Compatibility**

* Core domain contract for **user management**.
* Breaking changes here would ripple through authentication, authorisation, and persistence layers.
* DB schema must enforce uniqueness on UserId, Username, Email.

**🗄️ Data Model Notes**

* Likely corresponds to Users table.
* UserId → PK.
* Username + Email → should be indexed unique.
* PasswordHash → should be NOT NULL.

**✅ Confidence**

**High** – File is fully available and matches architectural expectations for domain user entity.

**File Review**

**File:** Domain/Entities/UserOrgRole.cs  
**Layer/Type:** Domain – Entity  
**Status:** Reviewed  
**Tokens (approx.):** ~220

**🧾 ELI5**

This file defines a **link entity** that assigns a role to a user within a specific organisation. It connects users, organisations, and roles together to control access and permissions.

**🎯 Purpose and Role**

* Represents a **many-to-many relationship** between:
  + Users (RoadmappUser).
  + Organisations (Organisation).
  + Roles (RoadmappRole).
* Provides the **authorisation context**: "User X has Role Y in Organisation Z."
* Essential for multi-tenant scenarios where users can belong to multiple organisations with different permissions.

**🔍 Detailed Breakdown**

* **Properties:**
  + OrganisationId (Guid) → identifies the organisation.
  + RoleId (Guid) → identifies the role assigned.
* **Implementation Notes:**
  + Very lightweight class.
  + Expected to be paired with UserId in database (not explicitly included in this class – may be added via relationships or composite keys).
  + Domain logic is minimal, entity mainly serves as **join table mapping**.

**⚠️ Error Handling & Validation**

* No validation logic here.
* Risk: duplicate role assignments unless enforced at DB/service level.
* Risk: missing UserId if not properly modeled elsewhere.

**🔐 Security Review**

* Roles are **core to access control**.
* Current implementation leaves validation to higher layers.
* DB uniqueness constraints needed to prevent role escalation via duplicates.

**⚡ Performance & Reliability**

* Simple entity → negligible performance concerns.
* Reliability depends on DB enforcing uniqueness (e.g., (UserId, OrganisationId, RoleId) composite key).

**📊 Observability**

* No logging (correct).
* Role assignment changes should be logged in Application layer.

**🧪 Testability & Coverage**

Suggested test cases:

1. **Positive:** Assign role to user → entity maps correctly.
2. **Negative:** Duplicate role assignment → should be rejected.
3. **Edge:** Assign same role across multiple organisations → allowed.
4. **Edge:** User with no roles → verify restricted access.

**🧹 Code Smells**

* **Medium:** No UserId property in this entity – may be incomplete for domain representation.
* **Low:** No enforcement against duplicates at domain level.

**🔧 Refactoring Suggestions**

* Add UserId property to make relationship explicit.
  + Effort: **Medium**, Priority: **5**.
* Consider converting to record for immutability.
  + Effort: **Small**, Priority: **2**.
* Add factory method AssignRole(userId, orgId, roleId) to enforce consistency.
  + Effort: **Medium**, Priority: **3**.

**📜 Contracts & Compatibility**

* Maps to join table for authorisation model.
* Likely UserOrgRoles table with composite PK.
* Schema migration required if UserId is missing.

**🗄️ Data Model Notes**

* Should have composite primary key: (UserId, OrganisationId, RoleId).
* Foreign keys to Users, Organisations, and Roles.
* Indexing important for fast access checks.

**✅ Confidence**

**High** – File is small, clearly intended as part of the authorisation join model.

**File Review**

**File:** Domain/Entities/XeroScope.cs  
**Layer/Type:** Domain – Entity  
**Status:** Reviewed  
**Tokens (approx.):** ~260

**🧾 ELI5**

This file defines a **Xero OAuth2 scope** entity. Each scope represents a specific permission (e.g., access to invoices). It tracks scope name, description, whether it’s active, and timestamps for creation and updates.

**🎯 Purpose and Role**

* Represents **authorisation scopes** granted to RoadmApp when integrating with Xero.
* Maps directly to the cfg.xero\_scopes table (configuration layer).
* Provides metadata for permission management.
* Used to determine which API endpoints RoadmApp can call against Xero.

**🔍 Detailed Breakdown**

* **Properties:**
  + Id (long) → unique identifier.
  + ScopeName (string, default "") → Xero-defined scope (e.g., "accounting.transactions").
  + Description (string, default "") → human-readable description of scope.
  + IsActive (bool) → whether scope is enabled.
  + CreatedAt (DateTimeOffset) → when record created.
  + UpdatedAt (DateTimeOffset) → when last updated.
* **Implementation Notes:**
  + Declared as sealed class → prevents inheritance.
  + Default values avoid nulls but allow empty strings.

**⚠️ Error Handling & Validation**

* No built-in validation for:
  + ScopeName format or required values.
  + Preventing duplicate scopes.
* Relies on DB schema and services for integrity.

**🔐 Security Review**

* Scopes define **authorisation boundaries** → critical for security.
* Must ensure only **valid Xero scopes** are allowed (currently free-form string).
* IsActive flag provides good control for enabling/disabling.

**⚡ Performance & Reliability**

* Lightweight entity.
* Scopes are likely cached in memory for performance.
* DB indexes on ScopeName and IsActive recommended for quick lookups.

**📊 Observability**

* No logging here (correct for Domain).
* Scope enable/disable should be auditable at Application layer.

**🧪 Testability & Coverage**

Suggested test cases:

1. **Positive:** Create scope with valid values → persists correctly.
2. **Negative:** Empty ScopeName → should be rejected at service level.
3. **Negative:** Duplicate ScopeName → should fail at DB constraint.
4. **Edge:** Toggle IsActive → ensure system respects change.

**🧹 Code Smells**

* **Medium:** Free-form ScopeName string → risks typos, invalid scopes.
* **Low:** Mutable setters → accidental modifications possible.

**🔧 Refactoring Suggestions**

* Enforce ScopeName validation (regex or enum of supported scopes).
  + Effort: **Medium**, Priority: **5**.
* Convert to record or make properties init-only.
  + Effort: **Small**, Priority: **3**.
* Add domain factory to create only valid scopes.
  + Effort: **Medium**, Priority: **4**.

**📜 Contracts & Compatibility**

* Represents **configuration contract** for OAuth2 scopes.
* Any change in ScopeName must align with Xero’s official scope definitions.
* DB migrations may be required if schema changes.

**🗄️ Data Model Notes**

* Id → PK.
* ScopeName should be unique + indexed.
* CreatedAt/UpdatedAt → track lifecycle (should be set automatically in DB or service).

**✅ Confidence**

**High** – File is fully available, straightforward, and clearly matches the documented Xero integration strategy.

**File Review**

**File:** Domain/Entities/XeroTenantInfo.cs  
**Layer/Type:** Domain – Entity  
**Status:** Reviewed  
**Tokens (approx.):** ~340

**🧾 ELI5**

This file defines the **information about a Xero tenant (organisation)** linked to a token. It stores identifiers, type, and name, and ensures each tenant can be uniquely identified and associated with tokens.

**🎯 Purpose and Role**

* Represents a **single Xero tenant (organisation)** associated with a RoadmApp user’s token.
* Used for multi-tenant handling in Xero integrations.
* Provides the mapping between local records and Xero’s tenant metadata.

**🔍 Detailed Breakdown**

* **Properties:**
  + Id (Guid) → unique identifier for this record.
  + TenantId (Guid) → Xero’s tenant identifier.
  + TenantType (string) → Xero tenant type (e.g., "ORGANISATION").
  + TenantName (string?, optional) → human-readable tenant name.
* **Constructors:**
  + Default constructor initializes:
    - Id = Guid.NewGuid().
    - TenantType = "" (default empty).
  + Overloaded constructor accepts TenantId and TenantType.
* **Implementation Notes:**
  + Lightweight entity.
  + No validation for tenant type string.
  + Intended to be **populated from Xero API** responses.

**⚠️ Error Handling & Validation**

* No enforcement of required fields (TenantType can be empty).
* Risk: invalid or duplicate tenant entries if not validated at service layer.

**🔐 Security Review**

* No sensitive values stored here.
* Safe for logging.
* Integrity critical: incorrect tenant mappings could cause data leakage across organisations.

**⚡ Performance & Reliability**

* Simple entity, negligible performance concerns.
* GUIDs ensure uniqueness.
* Reliability depends on DB enforcing uniqueness of (TenantId).

**📊 Observability**

* No logging in entity (correct).
* Tenant assignment changes should be auditable in Application layer.

**🧪 Testability & Coverage**

Suggested test cases:

1. **Positive:** Instantiate with valid TenantId and TenantType.
2. **Negative:** Empty TenantType → should be rejected at higher layer.
3. **Edge:** Duplicate TenantId across users → ensure DB constraint prevents.
4. **Integration:** Map correctly from Xero API response JSON.

**🧹 Code Smells**

* **Medium:** TenantType is free-form string (should be enum).
* **Low:** Mutable properties → accidental overwrites possible.

**🔧 Refactoring Suggestions**

* Convert TenantType to enum (e.g., Organisation, Practice).
  + Effort: **Medium**, Priority: **4**.
* Add domain validation (non-empty TenantType, TenantName optional).
  + Effort: **Small**, Priority: **3**.
* Consider record or init-only properties for immutability.
  + Effort: **Small**, Priority: **2**.

**📜 Contracts & Compatibility**

* Likely persisted in DB table XeroTenants.
* Tenant IDs must remain stable – any change risks breaking token associations.

**🗄️ Data Model Notes**

* Id → surrogate PK.
* TenantId → natural key (unique, indexed).
* TenantType → should have constraint (enum in code, check in DB).
* TenantName → optional, nullable column.

**✅ Confidence**

**High** – File fully available, straightforward, clearly matches documented multi-tenant Xero integration model.

**File Review**

**File:** Domain/Entities/XeroWebhookEvent.cs  
**Layer/Type:** Domain – Entity  
**Status:** Reviewed  
**Tokens (approx.):** ~120

**🧾 ELI5**

This file defines a **webhook event record** from Xero. Each event has an ID, type, category, tenant ID, and a timestamp for when it was processed.

**🎯 Purpose and Role**

* Represents an **incoming webhook notification** from Xero (e.g., invoice created/updated).
* Captures event metadata for auditing and processing.
* Provides domain-level structure for background services that consume and handle Xero webhooks.

**🔍 Detailed Breakdown**

* **Properties:**
  + Id (Guid) → internal record identifier.
  + EventId (Guid) → unique ID from Xero webhook payload.
  + TenantId (Guid) → identifies affected Xero tenant.
  + EventType (string, default "") → type of event (e.g., INVOICE\_UPDATED).
  + Category (string, default "") → category of event (e.g., INVOICE).
  + ProcessedAtUtc (DateTime) → timestamp when processed.
* **Implementation Notes:**
  + Declared as sealed class.
  + No constructors → relies on persistence layer or mappers.
  + Default empty strings allow unvalidated event types/categories.

**⚠️ Error Handling & Validation**

* No enforcement that EventType and Category are valid values.
* No check to prevent duplicate EventId entries (should be unique across system).
* Assumes DB/service layer handles idempotency.

**🔐 Security Review**

* No sensitive fields.
* Safe for logging.
* Must ensure webhook **signature validation** is enforced at Application layer (not in this entity).

**⚡ Performance & Reliability**

* Lightweight entity.
* Webhook ingestion may be high-volume → DB indexing on EventId and TenantId critical.
* ProcessedAtUtc ensures traceability.

**📊 Observability**

* No logging here (correct).
* Application layer should log event receipt, deduplication, and failures.

**🧪 Testability & Coverage**

Suggested test cases:

1. **Positive:** Create event with valid EventId, TenantId, EventType, Category.
2. **Negative:** Duplicate EventId → should be rejected at DB/service.
3. **Edge:** Empty EventType or Category → should fail validation.
4. **Edge:** ProcessedAtUtc not set → verify service enforces it.

**🧹 Code Smells**

* **Medium:** EventType and Category are free-form strings.
* **Low:** No constructor to enforce required properties.

**🔧 Refactoring Suggestions**

* Convert EventType and Category to enums.
  + Effort: **Medium**, Priority: **4**.
* Add constructor requiring EventId, TenantId, EventType, and Category.
  + Effort: **Small**, Priority: **4**.
* Enforce uniqueness of EventId at domain or persistence level.
  + Effort: **Small**, Priority: **5**.

**📜 Contracts & Compatibility**

* Maps directly to webhook payload contract.
* Must remain compatible with Xero’s webhook schema.
* DB schema should enforce uniqueness of EventId.

**🗄️ Data Model Notes**

* EventId → natural key (unique constraint).
* TenantId → foreign key to XeroTenants.
* ProcessedAtUtc → audit field, should default to NOW() at DB level.

**✅ Confidence**

**High** – File is small and self-contained, clearly represents webhook event metadata.