Reward Issuances

Namazu Elements Core Features · https://namazustudios.com/docs/namazu-elements-core/reward-issuance/

Reward Issuances represent a promise to grant a user one or more digital goods at a later time. They are commonly used when a reward must be granted *as a result of an event* (for example, completing a quest or redeeming an in-app purchase) but should not be immediately applied to the user's inventory.

A Reward Issuance can be created by either the client or the server and remains in an **ISSUED** state until it is redeemed. Upon redemption, the underlying Item(s) are applied to the user's inventory and the issuance is marked **REDEEMED**.

How Reward Issuances Fit Into Namazu Elements

Reward Issuances act as a bridge between **events** and **inventory changes**:

1. An event occurs (quest completion, IAP receipt validation, leaderboard placement, etc.).
2. A Reward Issuance is created for a specific user, describing *what* they are entitled to receive.
3. The client explicitly redeems the issuance.
4. The associated Item quantity is applied to the user's inventory.

This flow provides strong guarantees against duplicate rewards, supports delayed redemption, and allows the client to present rewards to the user in a controlled way.

Reward Issuance Lifecycle

A Reward Issuance moves through a simple lifecycle:

1. Issued

When first created, a Reward Issuance is placed into the **ISSUED** state. In this state:

- The reward has *not* yet affected the user's inventory.
- The issuance is eligible to be redeemed by the client.
- Only one ISSUED issuance may exist at a time for the same user and context (depending on type; see below).

2. Redeemed

When redeemed:

- The issuance transitions to **REDEEMED**.
- The configured Item and quantity are applied to the user's inventory.
- For **NON_PERSISTENT** issuances, the record may be automatically deleted shortly after redemption.

Core Fields

A Reward Issuance consists of the following key properties:

User

The user who is entitled to redeem the reward.

Item and Quantity

Defines *what* will be granted upon redemption:

- **Item**: The digital good to grant.
- **Item Quantity**: How many units are applied to the user's inventory.

State

The current status of the issuance:

- **ISSUED** – Created and awaiting redemption.
- **REDEEMED** – Successfully redeemed and applied.

The state cannot be modified directly; redemption must be performed through the appropriate redemption endpoint.

Context

A context is a unique string that identifies *why* this issuance exists. Contexts are critical for preventing duplicate rewards.

Examples include:

- A specific quest step completion
- An IAP transaction
- A leaderboard payout

When creating an issuance from the client, the context **must remain stable** across retries. This ensures that if a request is retried (for example, due to a network failure), duplicate issuances are not created.

> Contexts beginning with `SERVER.` are reserved for server-generated issuances and should not be used by clients.

Issuance Types

Reward Issuances support two types that control how duplicates are handled:

NON_PERSISTENT (Default)

- Multiple issuances may occur over time for the same user and context.
- Only one **ISSUED** issuance may exist at a time.
- Once redeemed, a new issuance with the same context may be created.

This is ideal for repeatable rewards such as daily quests or consumable purchases.

PERSISTENT

- Only one issuance may *ever* exist for a given user and context.

- Once created (even if redeemed), future attempts with the same context will be rejected.
- Expiration timestamps are ignored.

This is ideal for one-time rewards such as account-level unlocks or permanent entitlements.

Expiration

Reward Issuances may optionally define an expiration timestamp:

- If the issuance is not redeemed before the expiration time, it becomes invalid and may be deleted.
- Expiration can be extended by updating the timestamp.
- Persistent issuances ignore expiration entirely.

Expiration is useful for limited-time rewards, promotional grants, or time-boxed events.

Metadata and Tags

Metadata

Reward Issuances may include arbitrary metadata, either client-defined or server-generated. Metadata is commonly used to:

- Record additional context about the source event
- Attach mission or progression identifiers
- Store platform-specific purchase details

Tags

Tags provide a lightweight way to categorize or group issuances. They can be used for filtering, analytics, or debugging.

Redemption Results

When redeeming a Reward Issuance, the API returns a **RewardIssuanceRedemptionResult**, which contains:

- The requested issuance ID
- The updated Reward Issuance (on success)
- The affected Inventory Item (on success)
- Error details (on failure)

This allows clients to reliably confirm whether redemption succeeded and what inventory changes occurred.

Code Samples

Below are a few Java examples showing common Reward Issuance flows using the `RewardIssuanceDao`.

Create (or Fetch) an Issuance Idempotently

A key pattern is to treat issuance creation as **idempotent** by using a stable `context` string. If the same request is retried (network error, client restart, etc.), the same issuance should be returned rather than creating duplicates.

```
import dev.getelements.elements.sdk.dao.RewardIssuanceDao;
import dev.getelements.elements.sdk.model.goods.Item;
import dev.getelements.elements.sdk.model.reward.RewardIssuance;
import dev.getelements.elements.sdk.model.user.User;

public class RewardsService {

    private final RewardIssuanceDao rewardIssuanceDao;

    public RewardsService(RewardIssuanceDao rewardIssuanceDao) {
        this.rewardIssuanceDao = rewardIssuanceDao;
    }

    public RewardIssuance issueQuestReward(User user, Item item, int quantity,
```

```
                                        String questId, int stepSequence, int rewardIndex) {

        // For quest/progression-style rewards you typically want a server-generated context.
        // This context is stable and uniquely ties the issuance to a specific progression event.
        String context = RewardIssuance.buildMissionProgressContextString(
                questId,
                stepSequence,
                rewardIndex
        );

        RewardIssuance issuance = new RewardIssuance();
        issuance.setUser(user);
        issuance.setItem(item);
        issuance.setItemQuantity(quantity);
        issuance.setContext(context);
        issuance.setType(RewardIssuance.Type.NON_PERSISTENT);
        issuance.setSource("MISSION_PROGRESS");
        issuance.addTag("quest");
        issuance.addMetadata("questId", questId);
        issuance.addMetadata("sequence", stepSequence);
        issuance.addMetadata("rewardIndex", rewardIndex);

        // Creates the issuance if it doesn't exist, otherwise returns the existing one.
        // Newly-created issuances will be created in the ISSUED state.
        return rewardIssuanceDao.getOrCreateRewardIssuance(issuance);
    }
}
```

Create an Issuance for an IAP Receipt

For purchases, the goal is generally “**never grant this twice**.” Use a context derived from the purchase identifiers.

```
import dev.getelements.elements.sdk.model.reward.RewardIssuance;

public RewardIssuance issueAppleIapReward(User user, Item item, int quantity,
                                          String originalTransactionId, int skuOrdinal) {

    // Uses a hash of (originalTransactionId, itemId, skuOrdinal) to uniquely identify the
purchase reward.
```

```
    String context = RewardIssuance.buildAppleIapContextString(
            originalTransactionId,
            item.getId(),
            skuOrdinal
    );

    RewardIssuance issuance = new RewardIssuance();
    issuance.setUser(user);
    issuance.setItem(item);
    issuance.setItemQuantity(quantity);
    issuance.setContext(context);

    // For most IAP rewards, PERSISTENT is a good default because you want a one-time grant.
    issuance.setType(RewardIssuance.Type.PERSISTENT);
    issuance.setSource("APPLE_IAP");
    issuance.addTag("iap");

    return rewardIssuanceDao.getOrCreateRewardIssuance(issuance);
}
```

Redeem an Issuance

Redemption applies the issuance to the user's inventory and returns the `InventoryItem` that was modified.

```
import dev.getelements.elements.sdk.model.inventory.InventoryItem;
import dev.getelements.elements.sdk.model.reward.RewardIssuance;

public InventoryItem redeem(RewardIssuance issuance) {
    // Once redeemed, the issuance will be applied to the user's inventory.
    // Redemption must be safe to call multiple times without double-crediting.
    return rewardIssuanceDao.redeem(issuance);
}
```

Fetch Outstanding Issuances Then Redeem

A common client/server flow is to fetch all ISSUED issuances, present them to the user, and redeem them when the user accepts.

```
import dev.getelements.elements.sdk.model.Pagination;
import dev.getelements.elements.sdk.model.reward.RewardIssuance;
import dev.getelements.elements.sdk.model.reward.RewardIssuance.State;

import java.util.List;

public void redeemAllIssued(User user) {

    // Fetch a page of outstanding issuances.
    Pagination<RewardIssuance> page = rewardIssuanceDao.getRewardIssuances(
            user,
            0,
            50,
            List.of(State.ISSUED),
            List.of() // optionally filter by tags
    );

    for (RewardIssuance issuance : page.getItems()) {
        rewardIssuanceDao.redeem(issuance);
    }
}
```

Common Use Cases

- **Quest Rewards**: Issue rewards on quest completion and let the client redeem them when presenting a reward screen.
- **In-App Purchases**: Issue rewards only after server-side receipt validation.
- **Promotions**: Grant time-limited rewards with expirations.
- **One-Time Unlocks**: Use persistent issuances to guarantee a reward is only ever granted once.

Reward Issuances provide a flexible, reliable mechanism for granting digital goods while protecting against duplication and ensuring a clean, auditable reward flow.

Code Examples

The following examples demonstrate common server-side flows for creating and redeeming Reward Issuances using the RewardIssuanceDao. These examples assume server-side execution and omit error handling for clarity.

Creating a Reward Issuance

Reward Issuances are typically created in response to a validated event, such as quest completion or a verified in-app purchase.

```
RewardIssuance issuance = new RewardIssuance();

issuance.setUser(user);
issuance.setItem(item);
issuance.setItemQuantity(1);
issuance.setType(RewardIssuance.Type.NON_PERSISTENT);
issuance.setSource("QUEST");
issuance.setContext(
    RewardIssuance.buildContextString(
        "quest",
        questId,
        stepId
    )
);

RewardIssuance created = rewardIssuanceDao.create(issuance);
```

Key points:

- The context must uniquely identify the event that caused the issuance.
- Retrying the same create call with the same context is safe.
- The issuance is created in the **ISSUED** state automatically.

Redeeming a Reward Issuance

Redemption is an explicit action that applies the reward to the user's inventory and transitions the issuance to **REDEEMED**.

```
RewardIssuanceRedemptionResult result = rewardIssuanceDao.redeem(rewardIssuanceId);

if (result.getErrorDetails() == null) {
    RewardIssuance redeemed = result.getRewardIssuance();
    InventoryItem inventoryItem = result.getInventoryItem();

    // Redemption succeeded
} else {
    // Redemption failed
}
```

On successful redemption:

- The issuance state is updated to **REDEEMED**.
- The associated InventoryItem is created or updated.
- NON_PERSISTENT issuances may be deleted shortly after redemption.

Redeeming Multiple Issuances

Some flows (such as bulk reward claiming) may require redeeming multiple issuances at once. This can be done from the RewardIssuanceService, instead of the DAO:

```
List<String> issuanceIds = List.of(id1, id2, id3);

List<RewardIssuanceRedemptionResult> results =
    rewardIssuanceService.redeemRewardIssuances(issuanceIds);

for (RewardIssuanceRedemptionResult result : results) {
    if (result.getErrorDetails() == null) {
        // Success
    } else {
        // Handle failure
    }
}
```

Each issuance is redeemed independently; a failure in one does not prevent others from succeeding.

Idempotency and Safety

Reward Issuances are designed to be safely retried:

- Creating an issuance with the same user and context will not create duplicates.
- Redeeming an already redeemed issuance will return an error without re-applying inventory changes.
- Persistent issuances provide strong guarantees for one-time rewards.

These guarantees make Reward Issuances ideal for distributed systems where retries and partial failures are expected.