# Burntbite ECS Engine: Architecture & Implementation Deep-Dive

Jaime Páramo Benítez - Game Programmer

June 11, 2026

## Contents

# 1 Core Architecture: Entity Component System

## 1.1 Architectural Overview

The Burntbite engine moves away from traditional, deep Object-Oriented inheritance trees in favor of a custom Entity Component System (ECS). In this architecture, Entities are simply numeric IDs (`uint32_t`), Components are Plain Old Data (POD) structs, and Systems contain all the operational logic. This decoupling provides immense flexibility when designing new behaviors.

## 1.2 Type-Indexed Component Storage

Rather than using contiguous sparse sets, Burntbite employs a hash-map based storage system mapped dynamically at runtime. The core `ECS` class utilizes `std::type_index` to map component types to an abstract `IStorage` interface.

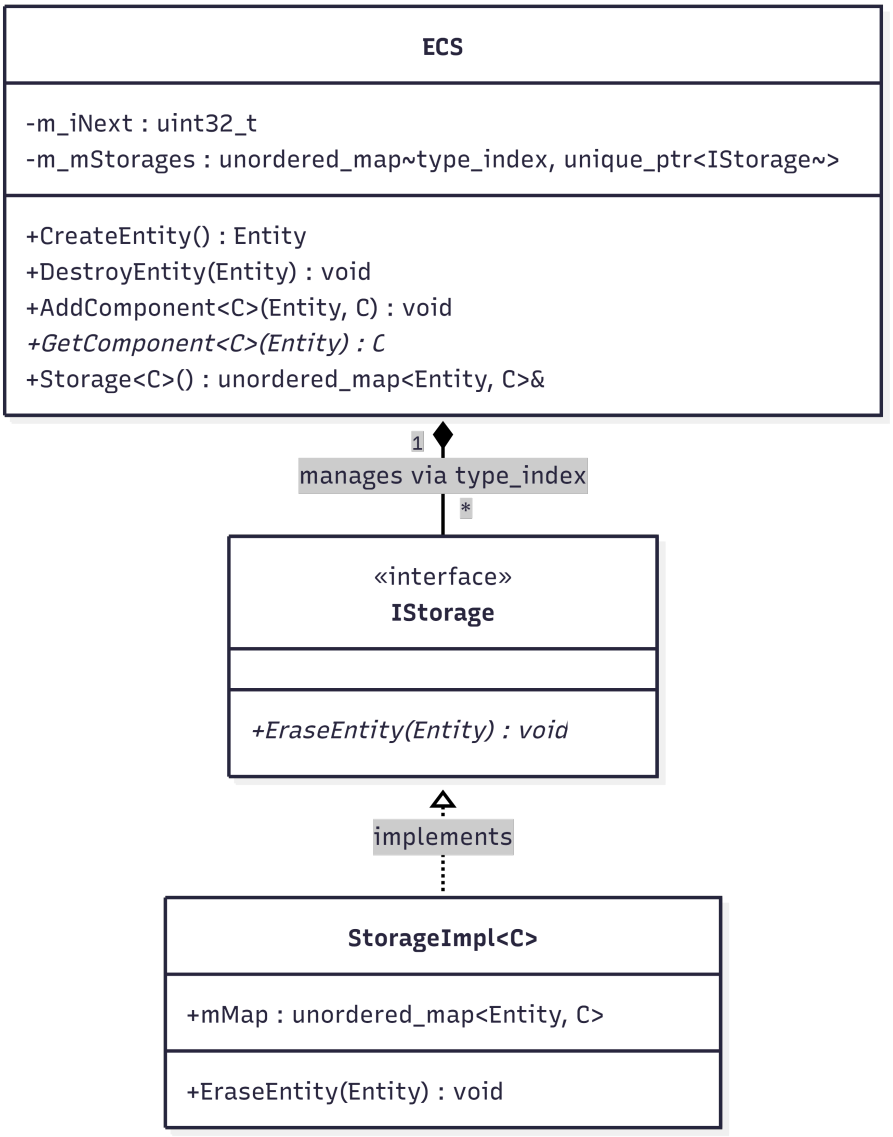


Figure 1: UML Class Diagram of the ECS Storage Architecture

This polymorphic storage approach allows the engine to add entirely new component types without modifying the underlying ECS registry structure. Under the hood, `StorageImpl<C>` maintains an `std::unordered_map<Entity, C>`, mapping the unique entity ID directly to its component data.

```
    struct IStorage {
      virtual ~IStorage() = default;
      virtual void EraseEntity(Entity e) = 0;
    };

    template<typename C>
    struct StorageImpl : IStorage {
      std::unordered_map<Entity, C> mMap;
      void EraseEntity(Entity e) override { mMap.erase(e); }
    };

```

Listing 1: Storage Implementation in ECS.h

# 2 Data-Driven Initialization: EntityFactory

## 2.1 Externalizing Entities via XML

To prevent hardcoding entity configurations, Burntbite utilizes the PugiXML library to construct entities dynamically from external data files. The `EntityFactory` acts as a dedicated assembler.

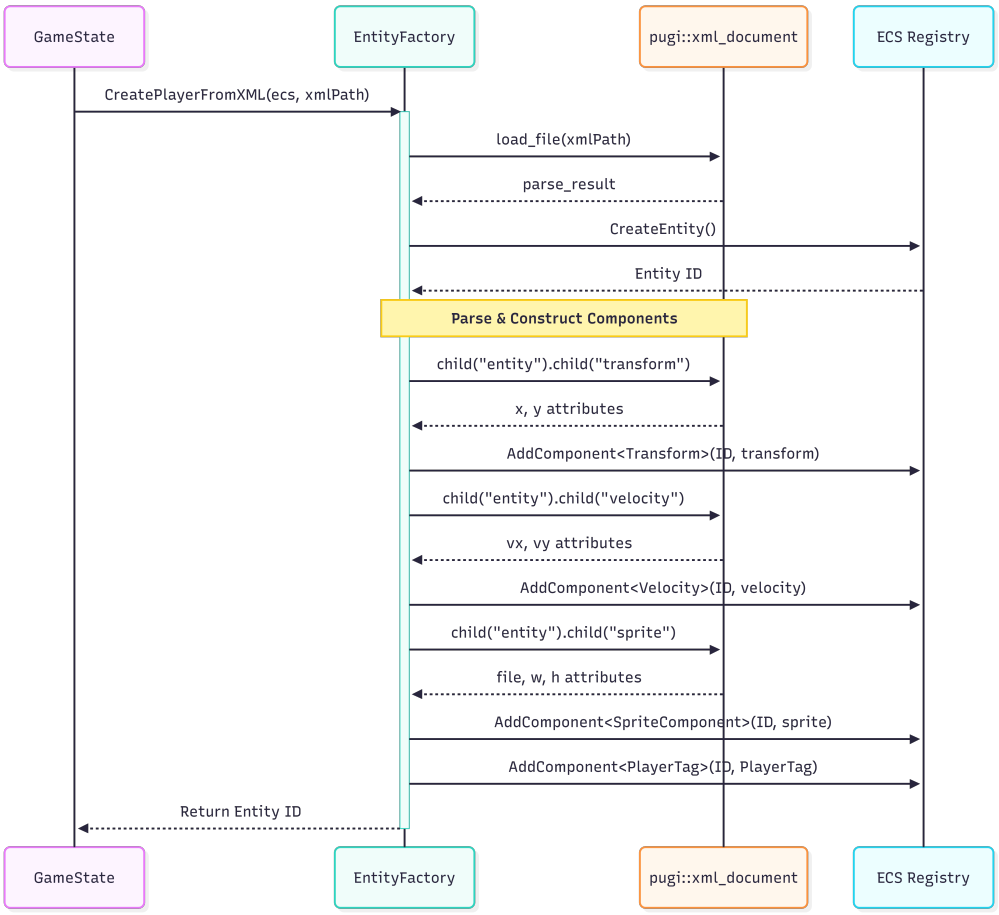


Figure 2: Sequence Diagram of XML Entity Parsing

When a file like `player.xml` is loaded, the factory parses the node attributes and translates them into ECS components. It provides robust fallbacks, ensuring that even if an XML node is missing (e.g., a missing velocity node), the engine safely attaches a default-constructed component to the entity ID.

```
    pugi::xml_node oTransformNode = oNode.child("transform");
    if (oTransformNode) {
      Transform oTransform;
      oTransform.fPosX = oTransformNode.attribute("x").as_float(240.f);
      oTransform.fPosY = oTransformNode.attribute("y").as_float(440.f);

      _rEcs.AddComponent<Transform>(iEntity, oTransform);
    }

```

Listing 2: Parsing and adding a Transform component via PugiXML

# 3 Game State & The Manager Entity

## 3.1 Overcoming Global State in ECS

A persistent challenge in pure ECS architectures is tracking global state such as score, high scores, or universal spawn timers, which do not logically belong to physical actors.

To solve this elegantly without breaking the ECS paradigm, the `GameState` utilizes a "Manager Entity." During initialization, it creates a headless entity ID specifically to hold the `SpawnTimer` and `Score` components. Systems, such as the `SpawnSystem`, simply query this specific entity to dictate the global rules of the game.

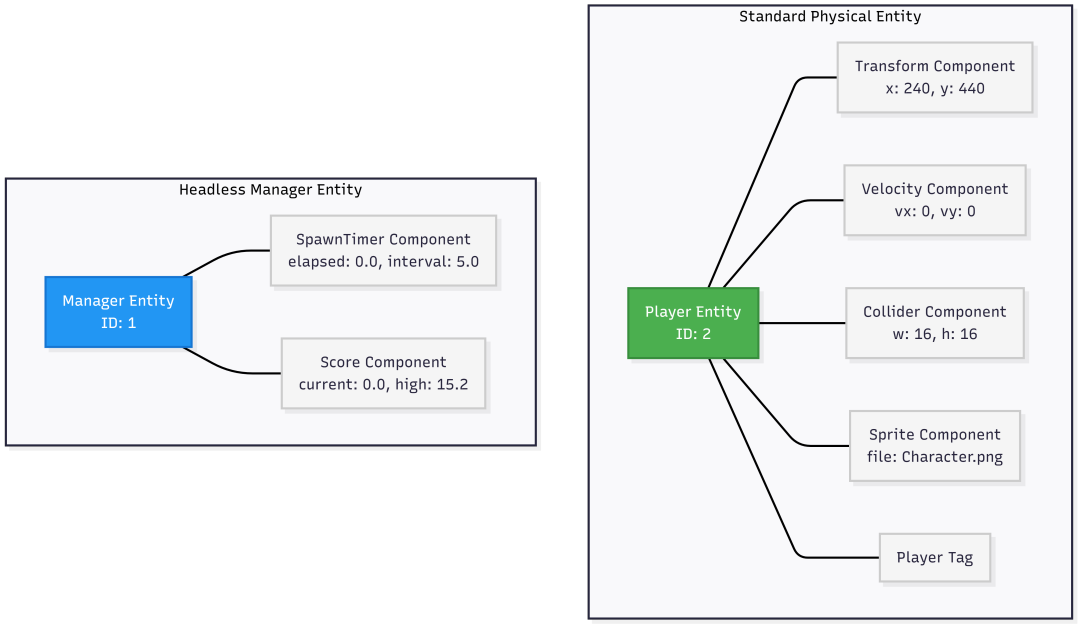


Figure 3: Component Association Diagram of the Manager Entity

## 3.2 Aggressive Memory Sweeping

When the player dies, the engine must reset the board without destroying the persistent Manager Entity or causing memory leaks. The `ResetForNewGame()` function aggressively iterates through all physical component maps (`Transform`, `Collider`, `Velocity`), collecting any entity ID that does not match the `m_iManager` ID, and systematically erases them.

```
    auto& rTransforms = _rEcs.Storage<Transform>();
    std::vector<Entity> lTransformsToRemove;

    for (auto& rKeyValue : rTransforms) {
      if (rKeyValue.first != m_iManager) {
        lTransformsToRemove.push_back(rKeyValue.first);
      }
    }

    for (Entity iEntity : lTransformsToRemove) {
      rTransforms.erase(iEntity);
    }

```

Listing 3: Aggressive sweeping in GameState::ResetForNewGame