Subjects. The Low-Level Entities

Subjects are the foundational lightweight entities managed by Apparatus. They are mostly UE-independent and consist of Traits and Flags.

Subject Handles

Subjects are not used directly and their internals are hidden deep from the framework user's perspective. Instead a special concept called *Subject Handle* is introduced. It's really much like a weak pointer in terms of Unreal. When you despawn a Subject all of the handles that are currently referencing it become automatically invalid. Internally this is managed through a generation-based referencing technique.

Subjective Layer

A Subject can have an additional higher-level dimension called Subjective. Subjectives are UE-managed objects (UObjects) which may contain the high-level Details in their composition. Subjectives with Details are generally more flexible and have additional features implemented as compared to raw Subjects with Traits. This comes at a cost of being not as memory-/cache-efficient and potentially less performant. Please note however, that the Subjective layer is optional and you may establish your project's logic entirely on Subjects if you want.

Subjects without the Subjective layer involved are called *barebone* Subjects. Both Subjective-based Subjects and barebone Subjects are commonly referred to as just *Subjects*. That's because every Subjective is actually a Subject internally and embeds it as an essential part. This is something like an inheritance, so every Subjective can also have Traits and Flags as part of its composition.

Spawning

Spawning is a process of creating a Subject as part of a Mechanism.

C++ Workflow

In order to spawn a new Subject within the Mechanism, you should call one of the SpawnSubject methods. The simplest one would be:

```
FSubjectHandle Subject = Machine::SpawnSubject();
```

If you want to spawn a Subject with some Traits initially attached to it, use the special templated version of the method:

```
FBurning Burning{10, 15.5f};
FSword Sword{2};
FSubjectHandle BurningSword = Machine::SpawnSubject(Burning, Sword);
```

This would efficiently pre-allocate a Slot for the Subject in the correct Chunk and initialize it according to the Traits supplied as the arguments.

Despawning

The *despawning* process is exactly the opposite of spawning and basically means destroying of a Subject. Destroying an already despawned (or invalid) Subject Handle is a legal operation that does nothing and reports no errors.

C++ Workflow

In order to destroy a Subject in your C++ code, use the Despawn method provided by the Handle structure. Do it like so:

```
void PickPowerup(FSubjectHandle Player, FSubjectHandle Powerup)
{
  // Add health/energy/strength to the player...
  ...
  // Remove the item from the world:
  Powerup.Despawn();
}
```