Trait

Traits are low-level data blocks (components) comprising subjects. Unlike Details, Traits are plain UStruct data. They are managed manually by the framework in a cache-efficient manner and mainly targeted towards the runtime performance.

Changing Traits

The Trait data is what called value-type in some languages. The behavior is somewhat native for UE environment. This basically means that in order to change a Trait's data you have to read it, change it afterwards and finally write it back re-applying it to the Subject. This procedure is necessary to guarantee the atomic and secure nature of the Trait types. Don't be too much scared by this copying routine since the procedure has some very high chances of being cache-local.

Garbage Collection Notes

As it was already said earlier Apparatus uses its own low-level memory model to store Traits efficiently. This comes at a certain cost. Basically you have to manually guarantee the safety for references to other UObjects (Actors, Components, etc) within your Traits, since those should be refernce-counted by the Garbage Collector which is absent.

Luckily, you can do it quite easily by referencing the same objects a certain asset or object via some global GC-managed UObject instance and retaining them this way. This global object may also be added to root to be retained. You may also add the referenced objects to root explicitly, before assigning them to Trait properties.

Referencing other Subjects via Subject Handles is perfectly fine though and is managed by Apparatus itself. Only remember that those Handles are like weak references. They don't hold the Subject referenced, they just invalidate themselves when the Subject is destroyed (despawned).

Creating Traits

All of the Unreal Engine's UStructs should be available within Apparatus automatically. If you can't see your struct somewhere in the Traits list, double-click on it in the Content Browser so it gets loaded. Generally speaking you should do it only once, since it gets loaded automatically if there are any references to it.

C++ Workflow

You should mainly refer to the official Unreal Engine manual on creating UStructs.

You basically create a header (`.h`) file and optionally a source (`.cpp`) file. An example of such header-only Trait would be:

```
#pragma once

#include "CoreMinimal.h"
#include "Moving.generated.h"

/**
 * The state of being moved with a certain velocity.
 */
USTRUCT(BlueprintType)
struct MY_API FMoving
{
    GENERATED_BODY()

  public:

    UPROPERTY(BlueprintReadWrite, EditAnywhere)
    float VelocityX = 0;

    UPROPERTY(BlueprintReadWrite, EditAnywhere)
    float VelocityY = 0;
};
```

You could omit the 🔗 UPROPERTY specifications but with the fields exposed like that you can now use MyTrait both in your C++ code and within BPs.