

# Iterating

In order to implement your actual Mechanic logic you have to be able to process all of the [Filter](#)-matching Subjects (and Subjectives, which are in turn - Subjects). For the purpose of effectiveness and consistency this is not done directly but via [Chains](#). You iterate those Chains iterating all the Subjects and Subjectives inside them.

Chains are iterated with the help of Cursors. Those are very much like container iterators. Multiple Cursors are available but this is more of a multi-threading future-proofing. For now, you should really use a single default Cursor only.

## C++ Workflow

Iterating a Chain is done through a special type of object called *Cursor*. You can have as many of those, but usually one is enough:

```
FChain::FCursor Cursor = Chain->Iterate();
```

The solid-variant would logically be like so:

```
FChain::FSolidCursor SolidCursor = SolidChain->Iterate();
```

When you got the Cursor needed, you can construct a simple while loop like so:

```
while (Cursor.Operate())
{
    auto Trait = Cursor.GetTrait<FMyTrait>();
    ...
}
```

The `.Operate()` method prepares the needed state and would return `false` when there are no more slots available.

With a solid Cursor you can get a direct (copy-free) reference to a Trait:

```
while (SolidCursor.Operate())
{
    auto& Trait = SolidCursor.GetTraitRef<FMyTrait>();
    ...
}
```

```
}
```

## Embedded Cursors

Apparatus provides a way to iterate the chains via embedded (self-allocated) cursors also.

Your basic loop is quite simple. It's just a `while` statement with a single condition:

```
while (Chain.BeginOrAdvance())
{
    ...
}
```

Inside this loop you can implement some actual logic. Using the [Subject's](#) direct or [Chain's](#) utility methods:

```
while (Chain.BeginOrAdvance())
{
    FSubjectHandle Subject = Chain.GetSubject();
    UMyDetail* MyPosition = Chain.GetDetail<UMyDetail>();
    FMyTrait    MyVelocity;
    Chain.GetTrait(MyVelocity);
    MyPosition->X += MyVelocity.VelocityX * DeltaTime;
    MyPosition->Y += MyVelocity.VelocityY * DeltaTime;
    ...
    MyVelocity.VelocityX = 0;
    MyVelocity.VelocityY = 0;
    Subject.SetTrait(MyVelocity);
}
```

When the Chain Cursor is gone past the last available Subject/Subjective the Chain becomes disposed and the previously locked Chunks and Belts become unlocked again, applying all the pending changes (if there are any).

From:

<http://turbanov.ru/wiki/> - **Turbopedia**

Permanent link:

<http://turbanov.ru/wiki/en/toolworks/docs/apparatus/iterating?rev=1630186182>

Last update: **2021/08/29 00:29**

