

# 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.

## 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:

```
FSolidChain::FCursor SolidCursor = SolidChain->Iterate();
```

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

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

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

With a solid Cursor you can also get a direct (copy-free) reference to a Trait (with a `GetTraitRef()` method):

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

Please note, that the Chain gets disposed automatically when all of the iterating Cursors have finished providing (iterating) the slots. To suppress this behavior use explicit `Retain()`/`Release()` calls for a custom lifetime organization:

```
Chain->Retain(); // Grab the chain.
FChain::FCursor Cursor = Chain->Iterate();
while (Cursor.Provide())
{
    ...
}
// Do some other stuff with the chain.
// It's now guaranteed to not be disposed.
...
Chain->Release(); // Free the chain.
```

## Embedded Cursors

Apparatus provides a way to iterate the chains via embedded (self-allocated) cursors. This is mainly utilized internally, by the Blueprints and generally should be avoided within your C++ code.

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=1641386814>

Last update: **2022/01/05 12:46**

