

Enchaining

Enchaining is a process of selecting a subset of Chunks (or Belts) based on a certain [Filter](#) criteria. Once they are Enchained the corresponding Belts and Chunks are becoming locked, i.e. any structural changes become minimized for the sake of consistent [Iterating](#).

C++ Workflow

That's pretty basic, really. You don't create (instantiate) Chains manually but those are actually managed by the [Machine class](#). Assuming you've already assembled the needed [Filter](#) all you have to do is to call a global Mechanism method named [Enchain](#) passing it the applied filter.

```
FChain& Chain = Mechanism->Enchain(Filter).Get();
```

You're now ready to [iterate](#) or [operate](#) the resulting Chain, but there is more.

Solid Chains

You can also chain to a special type of Chains called *solid*. Solid chains provide some additional features like getting direct references to Traits and Concurrency but they also limit the operations possible on the Subjects to only non-structural ones. That is, you can't add or remove any traits to/from the Subjects during some active solid enchainings. Enchaining to a solid chain is as easy as:

```
FSolidChain& Chain = Mechanism->EnchainSolid(Filter).Get();
```

Chunks Proxies

If you want to Iterate the Chunks directly, you would have to Enchain them into a list of Chunks Proxies.

In order to do that, a special overloaded [method](#) is present within the API. It accepts an [array](#) of Chunks Proxies as its argument second argument. The first one is of course, a Filter.

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

Permanent link:
<http://turbanov.ru/wiki/en/toolworks/docs/apparatus/enchaining>

Last update: **2022/07/18 18:12**



