Filters

Filters present a way to narrow down the iterating process down to Subjects and Subjectives with only certain combinations of Traits and Details. Filtering can be both including (positive) and excluding (negative). Including means that a Subject (or Subjective) must definitely have Traits (or Details) from the list. Excluding means otherwise - it must *not* have any of those.

The positive and negative parts can and should be used along with each other. Remember, that Apparatus is heavily optimized for all sorts of filtering involved, it doesn't exactly "search" for data but uses some fast masking procedures. Testing with a Filter of 100 Details is roughly the same as testing with a single one.

C++ Workflow

What you do is you basically create a FFilter instance. There are different constructors and methods available for this struct and you can use the most fitting for your case. As an example, let's create a Filter of one Trait and one Detail, valid for Booted Subjects only (this is the default):

```
#include "Filter.h"

...

FFilter MyFilter(EBootFilter::Booted);
Filter.Include<UMyDetail>();
Filter.Include<FMyTrait>();
```

You can now use the assembled filter to enchain the Belts or Chunks and iterate on them.

API documentation for filters.