Filtering

Filtering represents a way to narrow down the iterating (or operating) process down to Entities with only certain combinations of Components (or the lack of). The filtering specification is represented by a special data structure called *Filter*. Filtering can be both *including* (positive) and *excluding* (negative). In Apparatus you can combine any types of Components in the filters, i.e. both Traits and Details can be used at the same time within the same Filter.

Including

The including filtering specifies what Components a Subject (or Subjective) must certainly have in order to be matched by the filter (and thereby processed by the Mechanic). The including filterting of the Details supports the inheritance model, so if you include some base Detail class in the Filter, the descendants will match.

Excluding

The excluding filtering is exactly the opposite of the including one. It specifies a list of Components (Traits and Details) that must not exist within the Subject or Subjective in order to be matched by the filter.

The positive and negative parts of the filters can and should be used along with each other. Apparatus is heavily optimized for all sorts of filtering involved, so it doesn't exactly "search" for data but uses some fast masking procedures internally. Matching against a Filter of 100 Details is roughly the same as matching agains a single-Detail Filter.

C++ Workflow

What you do is you basically create a FFilter instance. There are different constructors, makers 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:

```
FFilter MyFilter = FFilter::Make<UMyDetail, FMyTrait>();
```

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

API documentation for filters.