Deferred Operations (Deferreds)

It’s no secret and if fact by design, that you can’t execute methods that change a Subject’s structure when using the Solid semantics. That essentially means you can only change the state of the individual Traits themselves, but not add nor remove the Traits themselves. The main advantage of the Solid enchaining is that it provides for the concurrent Operating and it would certainly be great to have more flexibility while evaluating the multi-threaded processing.

So there come handy the deferred operations (or *Deferreds* for short). Like the naming implies, those are not executed immediately but are instead delayed for a later, more suitable occasion.

Setting Traits

Say, we are implementing a real-time strategy game. A user can select multiple units and assign them orders (tasks). Send them marching to a destination point, for example. We could defer the Trait setting operation while Operating concurrently on the selected units, using the corresponding API method. Check out this exemplary snippet:

```
FVector Destination = GetUserClickedPoint(); // Retrieve the currently user-clicked point on the map.
auto SolidChain = Mechanism->EnchainSolid(TFilter<FUnit, FSelected>()); // Enchain all of the selected units.
SolidChain->OperateConcurrently([Destination](FSolidSubjectHandle Unit) // Process the selected units in a parallel fashion.
{
    Unit.SetTraitDeferred(FMoveToPointOrder{Destination}); // Defer-assign a trait that's telling the unit to move to the needed point.
});
```