Lesser-Prime Upgrades

- several ~~years~~ decades of use experience
- sub-optimal constructs
- unfortunately, Fat Bastard, v4, used to be reserved for large interface changes
- coincides with Auger Upgrade
- manpower?

Managers

- current interface impractical
- bloated to cover most cases
- readable only with excessive use of macros
- pure abstract virtual base class VManager
- any modification to abstract methods requires implementation in **all** Managers!
- can be humongous task, scary; need to modify **~70** headers!

VManager Data Interface

```
VMANAGER_ABSTRACT_GETDATA(double);
VMANAGER_ABSTRACT_GETDATA(int);
//VMANAGER_ABSTRACT_GETDATA(unsigned long long int);
VMANAGER_ABSTRACT_GETDATA(std::string);
VMANAGER_ABSTRACT_GETDATA(std::vector<double>);
VMANAGER_ABSTRACT_GETDATA(std::vector<int>);
VMANAGER_ABSTRACT_GETDATA(std::vector<std::string>);
VMANAGER_ABSTRACT_GETDATA(std::vector<bool>);
VMANAGER_ABSTRACT_GETDATA(std::list<double>);
VMANAGER_ABSTRACT_GETDATA(std::list<int>);
VMANAGER_ABSTRACT_GETDATA(std::list<std::string>);
VMANAGER_ABSTRACT_GETDATA(std::list<std::pair<int, int>>);
VMANAGER_ABSTRACT_GETDATA(utl::TabulatedFunction);
VMANAGER_ABSTRACT_GETDATA(std::map<int, utl::TabulatedFunction>);
VMANAGER_ABSTRACT_GETDATA(utl::TabulatedFunctionComplexLgAmpPhase);
VMANAGER_ABSTRACT_GETDATA(std::map<std::string, double>);
VMANAGER_GETDATA_DENIED(std::vector<std::vector<int>>);
```

Performance Issues

- in most cases the interface is a very effective bottleneck
- data needs to be described with string componentProperty, string componentName, IndexMap componentIndex
- modeled on XML tag structure
- in early days only scalar values
- added TabulatedFunction, vector<int>, vector<double>, map
- your data has to fit into existing categories
- dynamic polymorphism: abstract method in base, implementations everywhere
- templated virtual function?

NA61 Shine Solution

- can we have more complex returns without interface bloat?
- pass native classes to be filled by managers?
- class VManager::Handle
- glorified void*
- single interface to get data
- no need to updates to all managers

```
EStatus
GetData(Handle& returnData,
        const std::string& component,
        const std::string& property,
        const IndexMap& index = IndexMap())
  const
{ … }
```

Handle

```
class Handle {
public:
  template<typename T> Handle(T& t) : fData(&t), …

  template<typename T> T Get() { return *fData; }

  std::string GetTypeIdName() const
  { return utl::TypeId::Demangle(fTypeIdMangledName); }

  template<typename T> bool Is() const
  { return fTypeIdMangledName == utl::TypeId::MangledName<T>(); }

  private:
    void* fData;
    std::string fTypeIdMangledName;
};
```

Handle Client-side

```
DManagerRegister::GetInstance().GetData(
  tpc,
  tpc.GetComponentType(),
  property,
  tpc.GetIndexMap()
);
```

Handle Server-side

```
VManager::EStatus
TPCPadGeometryFixedManager::GenericGetData(
  VManager::Handle& returnData,
  const string& component,
  const string& property,
  const VManager::IndexMap& index
)
  const
{
  if (returnData.Is<TPC>() &&
      component == "TPC" &&
      property == "tpcPadGeometry")) {
    TPC& tpc = returnData.Get<TPC>();
    FillTPCGeometry(calibB, tpc);
  }
}
```

Object Decorations

```
class TPC : public Component {
public:
  TPC() : Component(“TPC”, “”,) { }
  …
};
```

Some More

CLHEP → Eigen

Geometry ← Eigen

SdFastTankSimulator ← update custom photon tracker