Unifying DataHandle

Charles Leggett

Gaudi Workshop 2016

What is a DataHandle

- semi-abstract mechanism to identify and access things in the Event Store
  - hides implementation of actual retrieve / record
  - typeless
- use DataHandles to manage data dependencies between components
  - overload Ath/GaudiAlgorithm::declareProperty(DataHandle<T>)
- actual usage inside of user Algorithms is typed
- separate VarHandle from VarHandleKey for re-entrancy

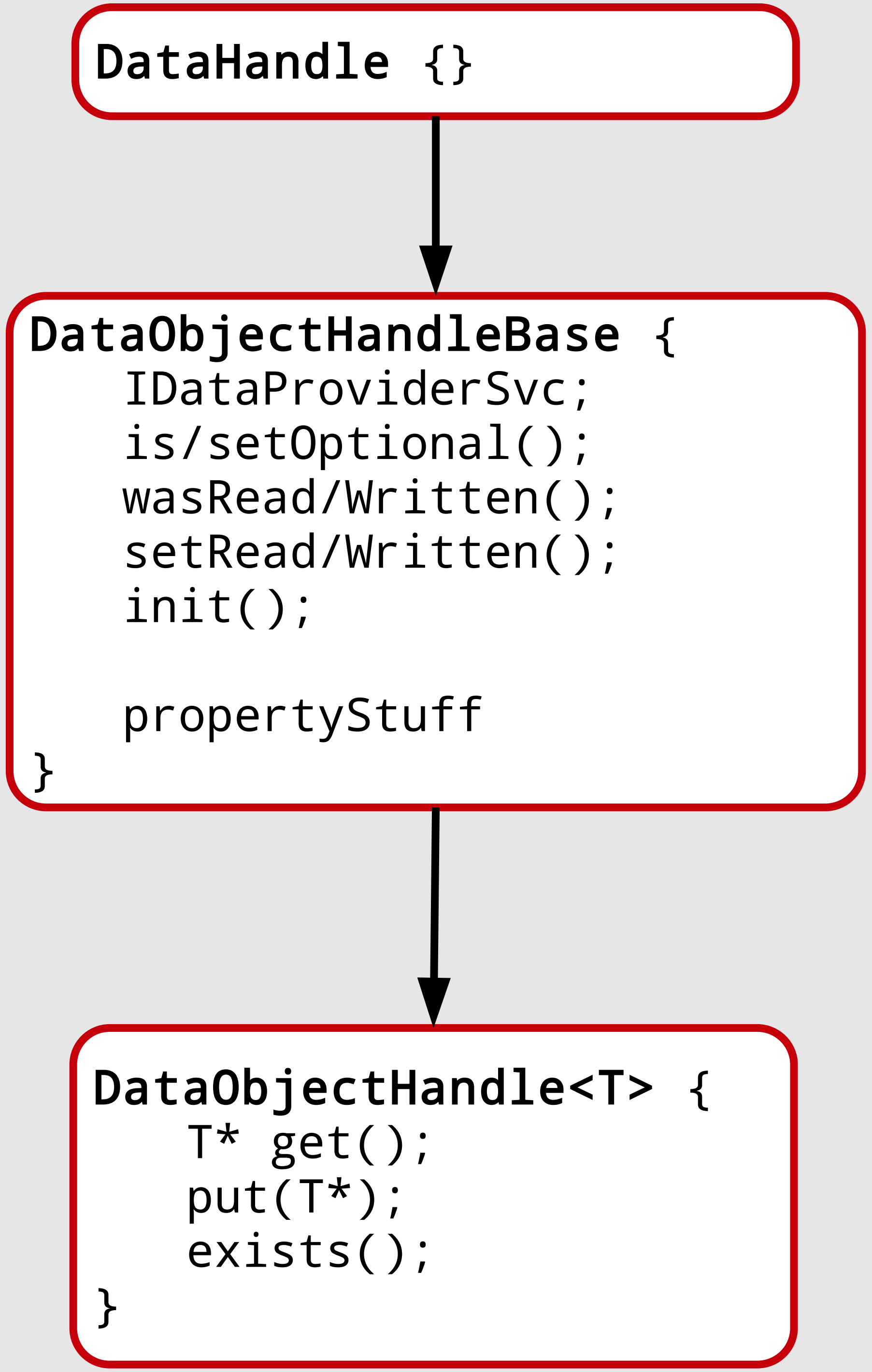
GaudiKernel/DataHandle

```
DataHandle {
   Mode
   Owner
   Key  (DataObjID)

   commit();
   reset();
}
```

```
DataObjID {
   Path/Key
   Type / ClassID
}
```

GaudiKernel/DataObjectHandle (Typed)



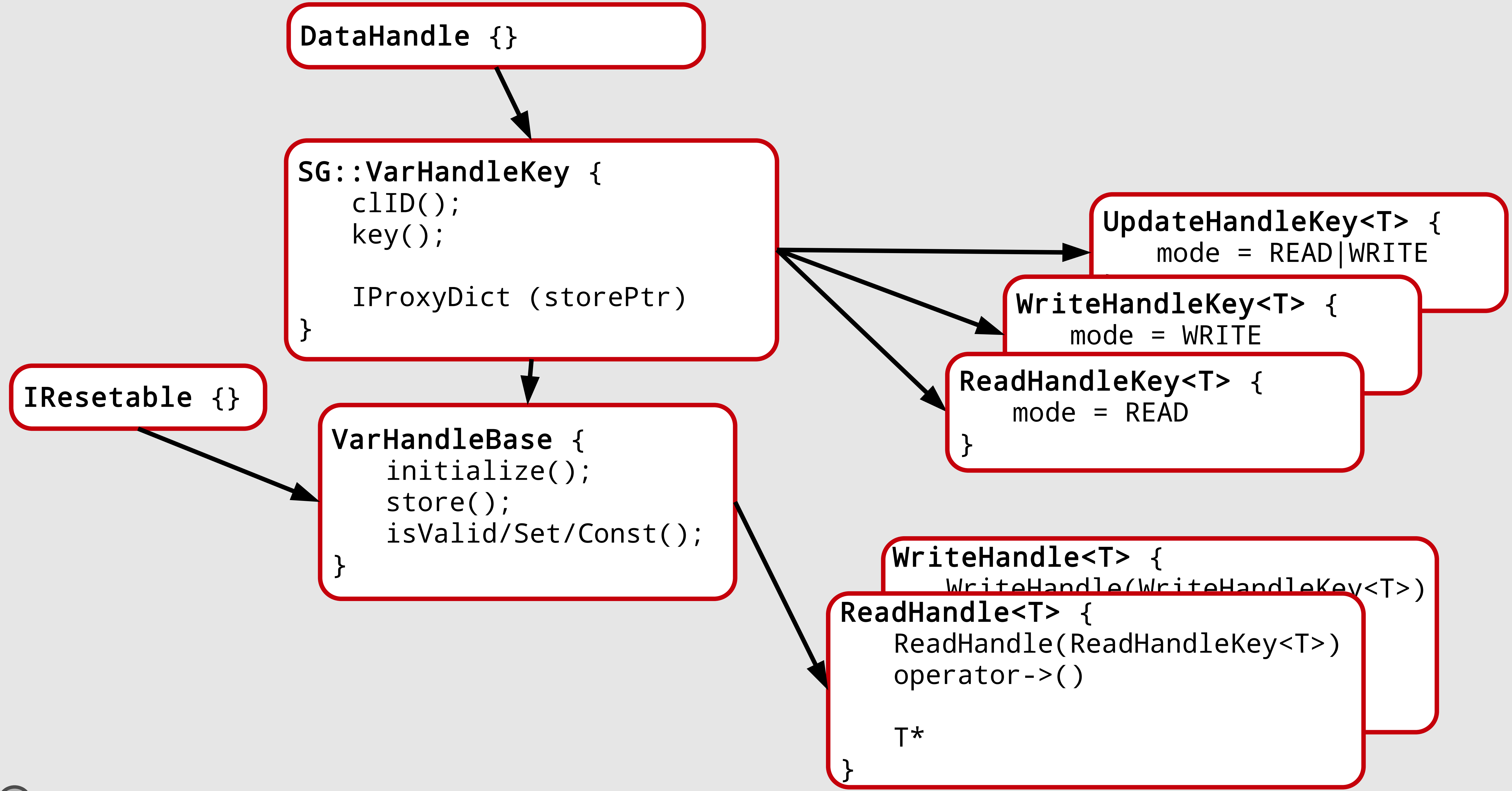
```
DataHandle {}
```

```
DataObjectHandleBase {
   IDataProviderSvc;
   is/setOptional();
   wasRead/Written();
   setRead/Written();
   init();

   propertyStuff
}
```

```
DataObjectHandle<T> {
   T* get();
   put(T*);
   exists();
}
```

SG::VarHandles



```
DataHandle {}
```

```
SG::VarHandleKey {
    clID();
    key();

    IProxyDict (storePtr)
}
```

```
IResetable {}
```

```
VarHandleBase {
    initialize();
    store();
    isValid/Set/Const();
}
```

```
UpdateHandleKey<T> {
    mode = READ|WRITE
```

```
WriteHandleKey<T> {
    mode = WRITE
```

```
ReadHandleKey<T> {
    mode = READ
}
```

```
WriteHandle<T> {
    WriteHandle(WriteHandleKey<T>)
```

```
ReadHandle<T> {
    ReadHandle(ReadHandleKey<T>)
    operator->()

    T*
}
```

ReadHandle

```
SG::ReadHandle<T> {
   ReadHandle(key, storeName);
   ReadHandle(ReadHandleKey<T>);
   ReadHandle(ReadHandleKey<T>,
              EventContext);

   bool isValid();

   const T* operator->();
   const T& operator*();
}
```

```
class MyAlg : public AthAlgorithm {
   SG::ReadHandleKey<MyObj> m_rdh;
}

MyAlg::MyAlg( const string& name, ISvcLocator* pSvc ) :
  ::AthAlgorithm( name, pSvc ), m_rdh("a1") {
  declareProperty("Key_RDH",m_rdh);
}

StatusCode MyAlg::initialize() {
  ATH_CHECK( m_rdh.initialize() );
}

StatusCode MyAlg::execute() {
  SG::ReadHandle<MyObj> rh(m_rdh);
  if (!rh.isValid()) {
    error() << retrieving " << m_rdh.key()) << endmsg;
    return StatusCode::FAILURE;
  }
  info() << "read: " << rh.key() << " = " << rh->val() );
}
```

WriteHandle

```
SG::WriteHandle<T> {
    WriteHandle(key, storeName);
    WriteHandle(WriteHandleKey<T>);
    WriteHandle(ReadHandleKey<T>, EventContext);

    bool isValid();

    const T* operator->();
    const T& operator*();

    record(std::unique_ptr<T>)
    recordNonConst(std::unique_ptr<T>);
}
```

```
class MyAlg : public AthAlgorithm {
    SG::WriteHandleKey<MyObj> m_wrh;
}

MyAlg::MyAlg( const string& name, ISvcLocator* pSvc ) :
  ::AthAlgorithm( name, pSvc ), m_wrh("a1") {
  declareProperty("Key_WRHH",m_wrh);
}

StatusCode MyAlg::initialize() {
  ATH_CHECK( m_wrh.initialize() );
}

StatusCode MyAlg::execute() {
  SG::WriteHandle<MyObj> wh(m_wrh);
  ATH_CHECK( wh.record( CxxUtils::make_unique< MyObj >
                      ( MyObj(_theValue_ ) ) );

  info() << wh.key() << " = " << wh->val() );
  ATH_CHECK(wh.isValid());
}
```

VarHandleArray

```
VarHandleKeyArray {
   assign( vector<string> );
   vector<VarHandleKey*> keys();
}
```

```
VarHandleKeyArrayCommon<T> : public VarHandleKeyArray,
                             public vector<T> {
   assign( vector<string> );
   vector<VarHandleKey*> keys();
}
```

```
ReadHandleKeyArray<T> : public VarHandleKeyArrayCommon< ReadHandleKey<T> > {
   vector<ReadHandle<T>> makeHandles();
}
```

Unification

- Why separation of DataObjectHandle and VarHandle?
  - DataObjectHandle uses Gaudi EventDataSvc get/put
    - integrated with GaudAlg
  - SG::VarHandle uses StoreGate
    - integrated with AthAlgorithm
- Can we unite them?
- differences:
  - get/put vs record/retrieve for EventStore integration
    - would need a common interface
  - VarHandleKeys
    - IProxyDict
  - optional()
  - alternativeDataNames()