A Draft StoreGate Tutorial

P. Calafiura

EDM WorkShop @ LBNL

May 9, 2000

Example 2:
StoreGate, Multiple Algorithms

- **Concepts:**

- **Part 1:**
  - **Getting to the StoreGate**
  - **Getting the event header from the StoreGate, and accessing its components**
- **Part 2:**
  - **Creating a DataObject**
  - **Writing DataObject to StoreGate**
  - **Reading DataObject from StoreGate**
  - **Chaining multiple algorithms**

Example 2:
StoreGate, Multiple Algorithms

- **Part 3:**
  - **Creating a ContainedObject**
  - **Writing a vector of ContainedObjects to StoreGate**
  - **Reading vector of ContainedObjects from StoreGate**

Files in StoreGate/example

- **ReadData.cxx, ReadData.h**
- **WriteData.cxx, WriteData.h**
- **DataObj.h, ContObj.h**

Example 2:
The StoreGate

- Event information is organized by type.
- There are several ways to get objects out of the StoreGate. These include the retrieveObject and retrieveAll templated methods from the StoreGateSvc. The first returns a DataHandle to the first object of specified type (optionally specifying a Key to match). The second returns a DataHandleList with all matching DataObjects.
- We prefer to use the more general retrieveAll.
- When an object is registered with the StoreGate, the StoreGate (actually the TDS) becomes the owner of the object. DON'T CALL THE DELETE METHOD OF THE OBJECT.

Example 2: Part 1
Getting to the StoreGate

- **Start with the ReadData algorithm skeleton**
- **Add the headers for the EventInfo, the StoreGate service, and the Data Handle:**

```
#include “EventAthena/EventInfo.h”
#include "EventStructure/EventID.h"
#include “StoreGate/StoreGateSvc.h”
#include “StoreGate/DataHandle.h”
```

- **Add a StoreGateSvc pointer data member to both the ReadData and WriteData algorithms**

```
StoreGateSvc* m_storeGate;
```

- **Add this to the initialize method of both algs**

```
StatusCode sc =
   service("StoreGateSvc", m_storeGate);
```

- **use the Atlas development version of Gaudi libs**

Example 2: Part 2
Creating a DataObject

- **To be registered in the StoreGate, an object must be a DataObject, ie it must inherit from DataObject.**
- **You must also define a static constant Class ID, which must be unique for each class. In MyDataObj.h:**

```
#include "Gaudi/Kernel/DataObject.h"
const static CLID CLID_MDO = 214222;
class MyDataObj: public DataObject {
 public:
   MyDataObj(): DataObject(){};
   static CLID& classID() { return CLID_MDO; }
   virtual CLID& clID() { return CLID_MDO; }
   int val() { return m_dat; }
   void val( int i) { m_dat = i; }
   virtual ~MyDataObj(){};
 private:
   int m_dat;
};
```

Example 2: Part 2
Writing DataObject to StoreGate

- **In the execute() method of WriteData, create a DataObject of type MyDataObj, and set its member data:**

```
MyDataObj *dobj = new MyDataObj;
dobj->val(10);
```

- **Register it in the event StoreGate:**

```
StatusCode sc =
  m_storeGate->registerObject(dobj);
if ( sc.isFailure(){ print error message }
```

Example 2: Part 2
Read DataObject from StoreGate

- **Objects in the StoreGate can be accessed via a DataHandle<>, which is templated in the type of the object, and has a usage syntax similar to a normal pointer. It also provide an iterator range to go through all elements of a collection**
- **In the execute() method of ReadData, get a SmartDataPtr of type MyDataObj from the event StoreGate, with its location specified by the property variable:**

```
DataHandle<MyDataObj> dobj;
if (
(m_storeGate>retrieveObject(dobj)).isFailure()
  ) print error message
```

- **Print value of object member data:**

```
log << MSG::INFO << "Data Object Val: "
    << dobj->val() << endreq;
```

Example 2: Part 2
DataObject Locations

- **Create a Key object, get a specific MyDataObj, print out its contents**

```
SillyKey bla(m_DataObjLoc);
DataHandle<MyDataObj> dobj3;
if (
(m_storeGate->retrieveObject(dobj3,
bla)).isSuccess() )
  log << MSG::INFO << bla << " Data Object Val:
" << dobj3->val() << endreq;
 else print error;
```

- **This key is silly because you still see the TDS location you're writing to. A key should be always a "well known" object.**

Example 2: Part 2
Chaining Multiple Algorithms

- **In order to execute more than one algorithm, all that must be modified is the "TopAlg" property of the Application Manager in the jobOptions file:**

```
ApplicationMgr.TopAlg = { "WriteData",
  "ReadData"} ;
```

- **Modify the GNUmakefile.in so that WriteData.cxx gets compiled:**

```
> cd ~/Workshop/src41/ControlExamples/GaudiTutorial
libGaudiTutorial.so_SRC = src/ReadData.cxx
  src/WriteData.cxx
```

- **Compile and link using "gmake install"**
- **run the executable in the run directory**

Example 2: Part 3
Creating a Contained Object

- **If you want to store a collection of objects, such as a group of hits, in the StoreGate, you should use an ObjectVector or an ObjectList.**
- **Objects must inherit from ContainedObject if they are to be stored in an ObjectVector/List:**

```
#include "Gaudi/Kernel/ContainedObject.h"
const static CLID CLID_MYCO = 214223;
class MyContObject: public ContainedObject {
 public:
   MyContObject(): ContainedObject(){};
   static CLID& classID() { return CLID_MYCO; }
   virtual CLID& clID() { return CLID_MYCO; }
   void set(float t, int i) { m_time = t; m_channelID= i;}
   int id() { return m_channelId; }
.....
 private:
  int m_channelId;
  float m_time;
};
```

Example 2: Part 3
Writing a Contained Object

- **In the execute() method of WriteData, create and fill a ContainedObject with the hits:**

```
MyContObj *mco_1 = new MyContObj;
mco_1->set(11.3, 1);
....
```

- **Create an ObjectVector on the heap, and fill it with a contained object:**

```
ObjectVector<MyContObj> *cobj = new
  ObjectVector<MyContObj>;
cobj->push_back(mco_1);
cobj->push_back(mco_2);
...
```

Example 2: Part 3
Writing a Contained Object

- **Register the ObjectVector with the StoreGate, and check to see that all went well:**

```
if (
(m_storeGate->registerObject(cobj)).isFailure()
  ) { print error mesg };
```

- **No need to specify the location of the object to be written. Its type will suffice to retrieve it if it all you care about. Otherwise add your own key to it (e.g. a string)**

Example 2: Part 3
Dealing with Multiple Objects

- **You can register as many object instances of a given type as you want. The StoreGate will assign them unique IDs and can return either all of them or the one of given ID or key value.**
- **Get all instances of MyDataObj and list them**

```
DataHandleList<MyDataObj> dobjs;
if (StatusCode::SUCCESS !=
 m_storeGate->retrieveAll(dobjs) )
  return( StatusCode::FAILURE);

DataHandleList<MyDataObj>::iterator iObj =
 dobjs.begin();
while(iObj != dobjs.end())
  log << MSG::INFO
      << (*iObj++)->val() << endreq;
```

Example 2: Part 3
Reading Contained Object

- **In the execute() method of ReadData, get a SmartDataPtr that points to the contained object in the StoreGate:**

```
ObjectVector<MyContObj>::iterator it;
DataHandle< ObjectVector<MyContObj> > vec;
if (StatusCode::SUCCESS !=
 m_storeGate->retrieveObject(vec) )
  return( StatusCode::FAILURE);
```

Iterate through the vector, printing out the values of its members:

```
for (it=vec->begin(); it!=vec->end(); it++) {
  float time = (*it)->time();
     int ID     = (*it)->id();
  log << MSG::INFO << "Time: " << time << "
  ID: " << ID << endreq;
```