Potential Changes to Gaudi / Athena I/O

Can ATLAS I/O use Gaudi?

- As part of I/O Re-Development I would consider three possible strategies:
  1. Keep current AthenaPOOL, APR/POOL, ROOT layer, but slimmed down to what ATLAS really needs
  2. Develop a more direct ROOT CnvSvc from scratch (or starting from what was pretty much abandoned in AthenaRootComps).
  3. This may be included by 1. or 2.: Use the existing Gaudi RootCnvSvc as base for Athena persistence to ROOT
- I have been investigating 3. for a little bit by trying to write (and later read) simple toy Athena objects (not xAOD) via Gaudi

JobOptions

```
## basic job configuration
import AthenaCommon.AtlasUnixGeneratorJob

## get a handle on the default top-level algorithm sequence and
service manager
from AthenaCommon.AlgSequence import AlgSequence
topSequence = AlgSequence()
from AthenaCommon.AppMgr import ServiceMgr as svcMgr

from Configurables import Gaudi__RootCnvSvc as RootCnvSvc
from Configurables import ( PersistencySvc, FileRecordDataSvc,)
# aliased names
from Configurables import Gaudi__MultiFileCatalog as FileCatalog
FileCatalog = FileCatalog(Catalogs = [
"xmlcatalog_file:GaudiOutput.xml" ])
svcMgr += CfgMgr.Gaudi__MultiFileCatalog("FileCatalog")

from Configurables import Gaudi__IODataManager as IODataManager
IODataManager = IODataManager(CatalogType =
"Gaudi::MultiFileCatalog/FileCatalog")
svcMgr += CfgMgr.Gaudi__IODataManager("IODataManager")

if not hasattr (svcMgr, 'EventPersistencySvc'):
    svcMgr += CfgMgr.EvtPersistencySvc( "EventPersistencySvc" )
svcMgr.EventPersistencySvc.CnvServices += [ "Gaudi::RootCnvSvc" ]
if not hasattr (svcMgr, 'ProxyProviderSvc'):
    svcMgr += CfgMgr.ProxyProviderSvc()

# Output setup: AthenaOutput with PoolTool (for DataHeader) and Gaudi
CnvSvc (no POOL/APR)
from OutputStreamAthenaPool.OutputStreamAthenaPoolConf import
AthenaPoolOutputStreamTool
Stream1_Tool = AthenaPoolOutputStreamTool( "Stream1_Tool" )
Stream1_Tool.CnvSvc = "Gaudi::RootCnvSvc"
from AthenaServices.AthenaServicesConf import AthenaOutputStream
Stream1 = AthenaOutputStream( "Stream1" )
Stream1.OutputFile = "PFN:GaudiOutput.root"
Stream1.WritingTool = Stream1_Tool

# Application setup
from AthenaCommon.AppMgr import theApp
# - I/O
theApp.OutStreamType = "AthenaOutputStream"
theApp.addOutputStream( Stream1 )
Stream1.ItemList += [ "EventInfo#*" ]
Stream1.ItemList += [ "ExampleHitContainer#MyHits" ]

# - Algorithms
from AthenaPoolExampleAlgorithms.AthenaPoolExampleAlgorithmsConf
import AthPoolEx__WriteData
topSequence += AthPoolEx__WriteData( "WriteData" )

# - Events
theApp.EvtMax   = 10
theApp.EvtSel   = "NONE" # do not use any event input

# Set output level threshold (2=DEBUG, 3=INFO, 4=WARNING, 5=ERROR,
6=FATAL)
svcMgr.MessageSvc.OutputLevel = 0
svcMgr.MessageSvc.defaultLimit = 10000000
```

Augment Gaudi DataObject to handle ATLAS DataBucket

- This modification should allow ATLAS to use a DataObject wrapper class (such as DataBucket)

```
diff --git a/GaudiKernel/GaudiKernel/DataObject.h
b/GaudiKernel/GaudiKernel/DataObject.h
index 3f1c66c4..28101aab 100644
--- a/GaudiKernel/GaudiKernel/DataObject.h
+++ b/GaudiKernel/GaudiKernel/DataObject.h
@@ -55,11 +55,15 @@ public:
   virtual unsigned long addRef();
   /// release reference to object
   virtual unsigned long release();
+  /// Retrieve void pointer to object, allows override in derived
classes (e.g. ATLAS DataBucket)
+  virtual void* object();
   /// Retrieve reference to class definition structure
   virtual const CLID& clID() const;
   /// Retrieve reference to class definition structure (static
access)
   static const CLID& classID();
+  /// Return the type_info for the object, allows override in
derived classes (e.g. ATLAS DataBucket)
+  virtual const std::type_info& tinfo() const;
   /// Retrieve DataObject name. It is the name when registered in
the store.
   const std::string& name() const;
```

```
diff --git a/GaudiKernel/src/Lib/DataObject.cpp
b/GaudiKernel/src/Lib/DataObject.cpp
index aab6cb7f..0af002a8 100644
--- a/GaudiKernel/src/Lib/DataObject.cpp
+++ b/GaudiKernel/src/Lib/DataObject.cpp
@@ -64,6 +64,11 @@ unsigned long DataObject::addRef()  {
   return ++m_refCount;
 }

+/// Retrieve void pointer to object
+void* DataObject::object() {
+  return this;
+}
+
 /// Retrieve Pointer to class definition structure
 const CLID& DataObject::clID() const   {
   return CLID_DataObject;
@@ -74,6 +79,11 @@ const CLID& DataObject::classID()    {
   return CLID_DataObject;
 }

+/// Return the @c type_info for the object
+const std::type_info& DataObject::tinfo() const {
+  return typeid(*this);
+}
+
 /// Retrieve DataObject name. It is the name when included in the
store.
 const std::string& DataObject::name() const {
   return m_pRegistry ? m_pRegistry->name() :
_sDataObjectCppNotRegistered;
```

Handle DataBucket etc.

```
diff --git a/RootCnv/src/RootCnvSvc.cpp
b/RootCnv/src/RootCnvSvc.cpp
index a6c87b39..d6a07ede 100644
--- a/RootCnv/src/RootCnvSvc.cpp
+++ b/RootCnv/src/RootCnvSvc.cpp
@@ -153,7 +153,7 @@ IConverter* RootCnvSvc::createConverter(long
typ,const CLID& wanted,const ICnvFa
 // ConversionSvc overload: Load the class (dictionary) for the
converter
 void RootCnvSvc::loadConverter(DataObject* pObject) {
   if (pObject) {
-    string cname = System::typeinfoName(typeid(*pObject));
+    string cname = System::typeinfoName(pObject->tinfo());
     if( log().level() <= MSG::DEBUG )
       log() << MSG::DEBUG << "Trying to 'Autoload' dictionary for
class " << cname << endmsg;
     TClass* cl = s_classesNames[cname];
@@ -175,7 +175,7 @@ TClass* RootCnvSvc::getClass(DataObject*
pObject) {
   i=s_classesClids.find(pObject->clID());
   if ( i != s_classesClids.end() ) return i->second;

-  string cname = System::typeinfoName(typeid(*pObject));
+  string cname = System::typeinfoName(pObject->tinfo());
   throw runtime_error("Unknown ROOT class for object:"+cname);
   return nullptr;
 }
```

```
diff --git a/RootCnv/src/RootDataConnection.cpp
b/RootCnv/src/RootDataConnection.cpp
index 666ba3d6..7165c987 100644
--- a/RootCnv/src/RootDataConnection.cpp
+++ b/RootCnv/src/RootDataConnection.cpp
@@ -462,7 +462,7 @@ CSTR RootDataConnection::empty() const {
 pair<int,unsigned long>
 RootDataConnection::saveObj(CSTR section, CSTR cnt, TClass* cl,
DataObject* pObj, int buff_siz, int split_lvl,bool fill) {
   DataObjectPush push(pObj);
-  return save(section,cnt,cl,pObj,buff_siz,split_lvl,fill);
+  return save(section,cnt,cl,pObj-
>object(),buff_siz,split_lvl,fill);
 }

 /// Save object of a given class to section and container
```

Get Gaudi out of the way

```
diff --git a/GaudiCommonSvc/src/PersistencySvc/PersistencySvc.cpp
b/GaudiCommonSvc/src/PersistencySvc/PersistencySvc.cpp
index 21fbc94f..de5c6a7d 100644
--- a/GaudiCommonSvc/src/PersistencySvc/PersistencySvc.cpp
+++ b/GaudiCommonSvc/src/PersistencySvc/PersistencySvc.cpp
@@ -74,7 +74,7 @@ StatusCode PersistencySvc::makeCall( int typ,
IOpaqueAddress*& pAddress, DataObj
     StatusCode status( StatusCode::FAILURE, true );
     switch ( typ ) {
     case CREATE_OBJ:
-      pObject = nullptr;
+      //pObject = nullptr; // Don't be nasty
       status  = svc->createObj( pAddress, pObject );
       break;
     case FILL_OBJ_REFS:
```

```
diff --git a/GaudiKernel/src/Lib/ConversionSvc.cpp
b/GaudiKernel/src/Lib/ConversionSvc.cpp
index 9b41af3c..33ac816a 100644
--- a/GaudiKernel/src/Lib/ConversionSvc.cpp
+++ b/GaudiKernel/src/Lib/ConversionSvc.cpp
@@ -44,7 +44,7 @@ StatusCode ConversionSvc::makeCall(int typ,
       if ( cnv )   {
         switch(typ)   {
         case CREATE_OBJ:
-          pObject = nullptr;
+          //pObject = nullptr; // Don't be nasty
           status = cnv->createObj(pAddress, pObject);
           break;
         case FILL_OBJ_REFS:
```

```
diff --git a/RootCnv/src/RootCnvSvc.cpp
b/RootCnv/src/RootCnvSvc.cpp
index a6c87b39..d6a07ede 100644
--- a/RootCnv/src/RootCnvSvc.cpp
+++ b/RootCnv/src/RootCnvSvc.cpp
@@ -465,7 +465,7 @@ StatusCode
RootCnvSvc::i__fillRepRefs(IOpaqueAddress* /* pA */, DataObject*
pObj

 // Read existing object. Open transaction in read mode if not
active
 StatusCode RootCnvSvc::i__createObj(IOpaqueAddress* pA,
DataObject*& refpObj)  {
-  refpObj = nullptr;
+  //refpObj = nullptr;
   if ( !pA ) return S_FAIL;
   RootDataConnection* con = nullptr;
   const string*  par  = pA->par();
@@ -473,7 +473,7 @@ StatusCode
RootCnvSvc::i__createObj(IOpaqueAddress* pA, DataObject*& refpObj)
{
   StatusCode sc =
connectDatabase(par[0],IDataConnection::READ,&con);
   if ( sc.isSuccess() ) {
     ipar[0] = (unsigned long)con;
-    DataObject* pObj = nullptr;
+    DataObject* pObj = refpObj;
     size_t len = par[1].find('/',1);
     string section = par[1].substr(1,len==string::npos ?
string::npos : len-1);
```

Don’t be quite so LHCB’y

```
diff --git a/RootCnv/src/RootCnvSvc.cpp b/RootCnv/src/RootCnvSvc.cpp
index a6c87b39..d6a07ede 100644
--- a/RootCnv/src/RootCnvSvc.cpp
+++ b/RootCnv/src/RootCnvSvc.cpp
@@ -313,7 +313,7 @@ StatusCode RootCnvSvc::connectOutput(CSTR db_name)  {
 StatusCode  RootCnvSvc::commitOutput(CSTR dsn, bool /* doCommit */) {
   if ( m_current ) {
     size_t len = m_currSection.find('/',1);
-    string section = m_currSection.substr(1,len==string::npos ? string::npos : len-1);
+    string section = len==string::npos ? "Event" : m_currSection.substr(1, len-1); // If no '/' is found use default. FIXME: not hardcoded
     TBranch* b = m_current->getBranch(section, m_currSection);
     if ( b ) {
       Long64_t evt = b->GetEntries();
@@ -407,10 +407,10 @@ StatusCode RootCnvSvc::i__createRep(DataObject* pObj, IOpaqueAddress*& refpAddr)
   if ( !pObj ) return error("createRep> Current Database is invalid!");
   CLID       clid = pObj->clID();
   IRegistry* pR   = pObj->registry();
-  string     p[2] = {m_current->fid(), pR->identifier()};
+  string     p[2] = {m_current->fid(), pR->identifier().empty() ? "Default" : pR->identifier()}; // Handle Default keys
   TClass*    cl   = (clid == CLID_DataObject) ? m_classDO : getClass(pObj);
   size_t     len  = p[1].find('/',1);
-  string     sect = p[1].substr(1,len==string::npos ? string::npos : len-1);
+  string     sect = len==string::npos ? "Event" : p[1].substr(1, len-1);
   pair<int,unsigned long> ret =
     m_current->saveObj(sect,p[1],cl,pObj,m_bufferSize,m_splitLevel,true);
   if ( ret.first > 1 || (clid == CLID_DataObject && ret.first==1) ) {
@@ -433,9 +433,9 @@ StatusCode RootCnvSvc::i__fillRepRefs(IOpaqueAddress* /* pA */, DataObject* pObj
       StatusCode status = dataMgr->objectLeaves(pObj, leaves);
       if ( status.isSuccess() )  {
         RootRef ref;
-        const string& id    = pR->identifier();
+        const string& id    = pR->identifier().empty() ? "Default" : pR->identifier();
         size_t        len   = id.find('/',1);
-        string        sect  = id.substr(1,len==string::npos ? string::npos : len-1);
+        string        sect  = len==string::npos ? "Event" : id.substr(1, len-1);
         LinkManager* pLinks = pObj->linkMgr();
         for(auto &i : leaves ) {
           if ( i->address() ) {
```

Bottom line

- After these changes to Gaudi, I was able to write some dummy toy objects, but reading them back into a DataBucket, seems to be even harder.
- At this point, Gaudi really provides only LHCB persistence, with many assumptions that are not true for ATLAS.