Fetching Calibration Data: Internals

A typical client algorithm for calibration data has to make at least two calls to the Calibration Data Service to insure that a dataset appropriate for the current event time will be in the Calibration TDS (aka TCDS), the first to get a pointer to the data in the TCDS (and create it from its persistent form if it's not already there), the second to verify that data already present in the TCDS is still applicable to current event (and if not update it with a new data set which is). The first is done with a `retrieveObject` call, or simply by declaring a SmartDataPtr. The second is done with an `updateObject` call.

The following pictures explain some of the internals of each of these operations.
Create a Calibration Object

client

m_calibDataSvc->retrieveobject("/Calib/ACD_Eff/vanilla", pObj)

Ask TCDS data service for pointer to object

CalibDataSvc (DataSvc)

pLoader->createObject(pAddress , pObj)

If object not already in TCDS, ask loader (Persistency service) to load it. So-called address is a descriptor including enough information to guide conversion

PersistencySvc

pSvc->createObject(pAddress , pObj)

Ask format-specific conversion service (MySQL) to create object

CalibMySQLCnvSvc

m_meta->findBest(&ser, calibType, eventTime, ...);
m_meta->getReadInfo(ser, &physFmt, &version, &ident);
m_persSvc->createObj(tmpAddress, pObj);

Search meta dbs for best match; get info needed to retrieve calib bulk data, ask Persistency Svc to create corresponding obj.
Create a Calibration Object (cont'd)

PersistencySvc

```
pSvc->createObject(pAddress ,pObj)
```

Ask format-specific conversion service (XML, later also ROOT) to create object

CalibXMLCnvSvc (ConversionSvc)

```
pCnv->createObject(pAddress ,pObj)
```

Find the right converter for this data type, this physical format, and ask it to create the object

XmlTest1Cnv : XmlBaseCnv

XmlBaseCnv fills info common to all calib data (validity interval, metadata serial number), parses XML file and passes DOM_Document node to specific converter, which fills in remainder of data.
Update a Calibration Object

- **client**: `m_calibDataSvc->updateObject(pMyObj)`
  - Ask TCDS data service to see if already-retrieved object is current w.r.t. current event.

- **CalibDataSvc (DataSvc)**: `pLoader->updateObj(pAddress ,pObj)`
  - Ask loader (Persistency service) to check for currency and update if necessary. So-called address is a descriptor including enough information to guide conversion.

- **PersistencySvc**: `pSvc->updateObject(pAddress ,pObj)`
  - Ask format-specific conversion service (MySQL) to check for currency and update object if necessary.
Update a Calibration Object (cont'd)

If old calibration object validity interval includes current event time, simply return ptr to old object. Else...

```c
CalibMySQLCnvSvc
```

```c
m_meta->findBest(&ser, calibType, eventTime, ...);
m_meta->getReadInfo(ser, &physFmt, &version, &ident);
m_persSvc->createObj(tmpAddress, pNewObj);
```

Search meta dbs for best match; get info needed to retrieve calib bulk data, ask Persistency Svc to create corresponding obj, which will be stored in a different place from the original

```c
PersistencySvc
```

```c
pSvc->createObject(pAddress ,pObj)
```

Ask format-specific conversion service (XML, later also ROOT) to create object
Update a Calibration Object (cont'd)

**CalibXMLCnvSvc**

`pCnv->createObject(pAddress ,pObj)`

Find the right converter for this data type, this physical format, and ask it to create the object.

**XmlTest1Cnv : XmlBaseCnv**

XmlBaseCnv fills info common to all calib data (validity interval, metadata serial number), parses XML file and passes DOM_Document node to specific converter, which fills in remainder of data.

**CalibMySQLCnvSvc**

```cpp
CalibBase* pBase=dynamic_cast<CalibBase*>(pObj);
CalibBase* pNewBase=dynamic_cast<CalibBase*>(pNewObj);
pBase->update(pNewBase);
```

Call virtual `update(CalibBase* other)`. Must cast first since this function doesn't exist for DataObject. Typical calib. class implementation will `dynamic_cast` arg. to its own type, invoke `CalibBase::update` to properly handle base class members, then do deep copy of its own members.