Channel Filter Service Interface

Brandon Eberly

September 8, 2015

Introduction

- Piggy-backed off of work done by Gianluca
  - See talk at July 28 coordination meeting, though many class names and file locations have changed

- Used "**provider&service**" model established for pedestal database readout

- Created two implementations of **provider&service** for **larsoft**, one service implementation for **uboonecode**

- Modified `ChannelFilter` to be a wrapper for the **provider&service** interface.
  - `ChannelFilter` is "deprecated", but experiments can continue to use it
  - Functionality is identical, but experiments will need to change their fcl

- Feature branches:

•**larevt**	feature/Issue1083
•**lareventdisplay**	feature/eberly_channelfilter
•**uboonecode**	feature/eberly_channelfilter

ChannelFilter Provider Interface

- See `larevt/CalibrationDBI/Interface/IChannelFilterProvider.h`. Public interface:

```
/// Returns whether the specified channel is physical and connected to wire
virtual bool IsPresent(DBChannelID_t channel) const = 0;

/// Returns whether the specified channel is bad in the current run
virtual bool IsBad(DBChannelID_t channel) const = 0;

/// Returns whether the specified channel is noisy in the current run
virtual bool IsNoisy(DBChannelID_t channel) const = 0;

/// Returns whether the specified channel is physical and good
virtual bool IsGood(DBChannelID_t channel) const = 0;
```

Channel statuses that must always be implemented

```
/// Returns a status integer
virtual unsigned short Status(DBChannelID_t channel) const
  { return 99;}
```

Allows implementation of additional statuses

```
/// Returns a copy of set of good channel IDs for the current run
virtual DBChannelSet_t const GoodChannels() const = 0;

/// Returns a copy of set of bad channel IDs for the current run
virtual DBChannelSet_t const BadChannels() const = 0;

/// Returns a copy of set of noisy channel IDs for the current run
virtual DBChannelSet_t const NoisyChannels() const = 0;
```

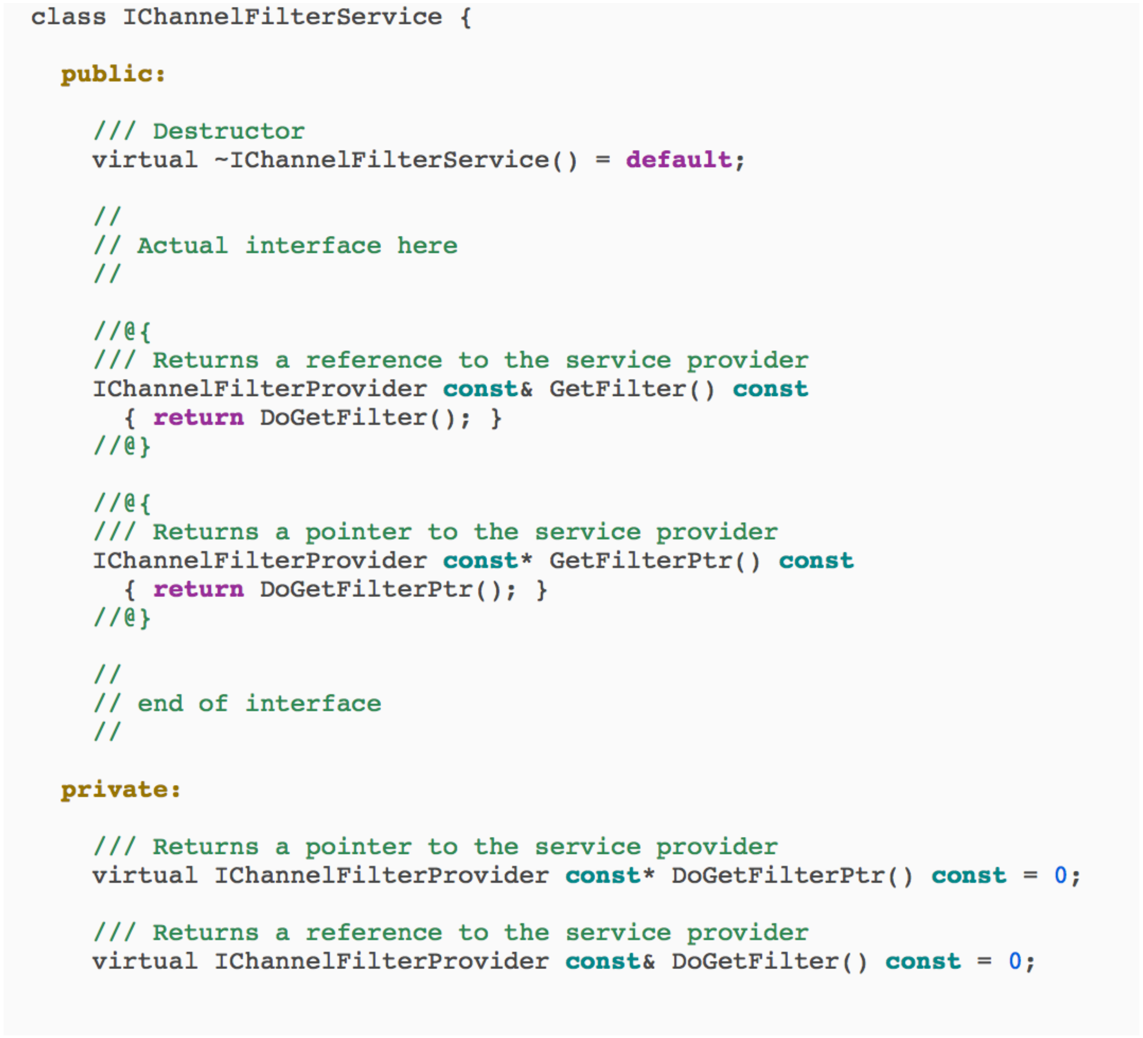
Get `std::set<ChannelID>` for implemented statuses

```
/// Prepares the object to provide information about the specified time
/// @return whether information is available for the specified time
virtual bool Update(DBTimeStamp_t ts) = 0;
```

`Update(...)` for database caching

ChannelFilter Service Interface

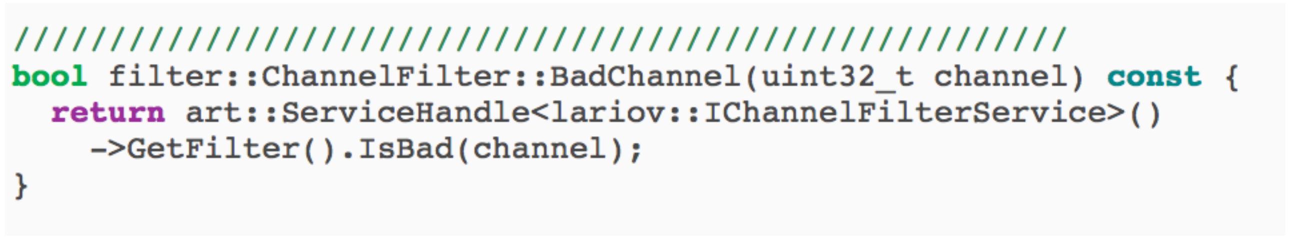
See `larevt/CalibrationDBI/Interface/IChannelFilterService.h`:



```
class IChannelFilterService {

  public:

    /// Destructor
    virtual ~IChannelFilterService() = default;

    //
    // Actual interface here
    //

    //@{
    /// Returns a reference to the service provider
    IChannelFilterProvider const& GetFilter() const
      { return DoGetFilter(); }
    //@}

    //@{
    /// Returns a pointer to the service provider
    IChannelFilterProvider const* GetFilterPtr() const
      { return DoGetFilterPtr(); }
    //@}

    //
    // end of interface
    //

  private:

    /// Returns a pointer to the service provider
    virtual IChannelFilterProvider const* DoGetFilterPtr() const = 0;

    /// Returns a reference to the service provider
    virtual IChannelFilterProvider const& DoGetFilter() const = 0;
```

ChannelFilter deprecation

- `ChannelFilter` is now a wrapper for the previous interfaces. For example:



```
////////////////////////////////////////////////////////
bool filter::ChannelFilter::BadChannel(uint32_t channel) const {
  return art::ServiceHandle<lariov::IChannelFilterService>()
    ->GetFilter().IsBad(channel);
}
```

- `larevt/Filters/SimpleChannelFilter` and `larevt/Filters/SimpleChannelFilterService` are implementations of the interface that preserve the previous functionality of `ChannelFilter` (fcl-configurable list of bad/noisy channels)

- Example: if you are Argoneut, add these lines to your fcl file (if this doesn't work, let me know – similar file provided for bo, not sure what Gianluca did for Dune):

```
#include "channelfilter_argoneut.fcl"    #located in larevt/Filters/
services.user.IChannelFilterService:  @local::channel_filter_argoneut
```

Single-IOV Implementation for Database

- See in `larevt/CalibrationDBI:` `IOVData/ChannelStatus.h`
  `Providers/SIOVChannelFilterProvider.*`
  `Services/SIOVChannelFilterService_service.cc`

- Provider inherits from provider interface and `DatabaseRetrievalAlg` (latter provides hooks to conditions database)
    - Service calls `Update(...)` before each event is processed (`PreProcessEvent`)

- Channel statuses are ordered from "worst" to "best", to allow cutting on `IChannelFilterProvider::Status()` in larsoft algorithms

```
enum chStatus {kDISCONNECTED=0, kDEAD=1, kLOWNOISE=2, kNOISY=3, kGOOD=4, kUNKNOWN=5};
```

- Channel statuses `kLOWNOISE` and `kDEAD` both map to `IsBad`

```
/// Returns whether the specified channel is bad in the current run
bool IsBad(DBChannelID_t channel) const override {
  return this->GetChannelStatus(channel).IsDead() || this->GetChannelStatus(channel).IsLowNoise();
}
```

- Provider has function `AddNoisyChannel(...)` to allow the service that owns it to modify the list of noisy channels (useful if channel noise varies by event)
    - **larsoft** service implementation does not use this; **uboonecode** impl does

Other Changes

- Channel ID and Timestamp types used by database interfaces now hide behind typedefs
  - `larevt/CalibrationDBI/Interface/CalibrationDBIFwd.h`

  - The ChannelID is changed from `uint64_t` to `uint32_t`, matching what is used internally by **art/larsoft** for channelIDs (OKed by Jon Paley)

  - Detector Pedestal interfaces and implementations updated to use the typedefs

- Some internal changes to the detector pedestal implementation
  - Remove a try/catch block that was hit every time conditions were requested (told that this was slow)
  - Fill default values (if used) in constructor using list of channels in geometry
  - These changes do not change functionality, unless you were in the habit of asking for default values for channels that do not exist...

Next

- Feature branches are ready to be merged into develop
  - MicroBooNE fcl files are configured to use its implementation of the service

- Might need data product to store event-by-event noisy channel information
  - Allow **uboonecode** `ChannelFilter` implementation to retrieve this data product, rather than determine noisy channels internally
  - Maybe **larsoft** implementation would use this too

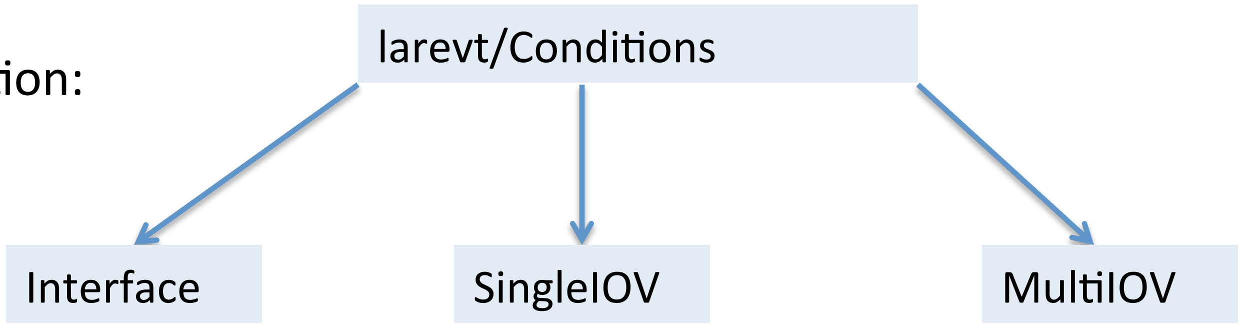
- Some interest in MicroBooNE for channel statuses that vary by TDC (e.g. half a wire is noisy)
  - If needed, we can overload the provider interface with an optional argument.

- Working on interface for **PMT conditions**
(larevt feature/eberly_PmtGainDBI – only compiles if feature/Issue1083 is merged in)
  - Might be very uboone-specific, so I might just move this work over to **uboonecode**

Organization/Naming Discussion

- Gianluca suggested I talk about how to organize our conditions interface code
    - We have a couple interfaces already, with more to come. They should all live together

- My suggestion:



Holds all abstract interfaces for data providers and services, along with typedefs and utility classes used in the interfaces

Holds all files related to the default single-IOV database implementation of the interfaces

Holds all files related to the default multiple-IOV database implementation of the interfaces

Use a single namespace for all classes in larevt/Conditions: **larcond**

- If you approve, we can start a feature branch to make these changes

Backup

Provider&Service Model

One DataProviderInterface per database folder

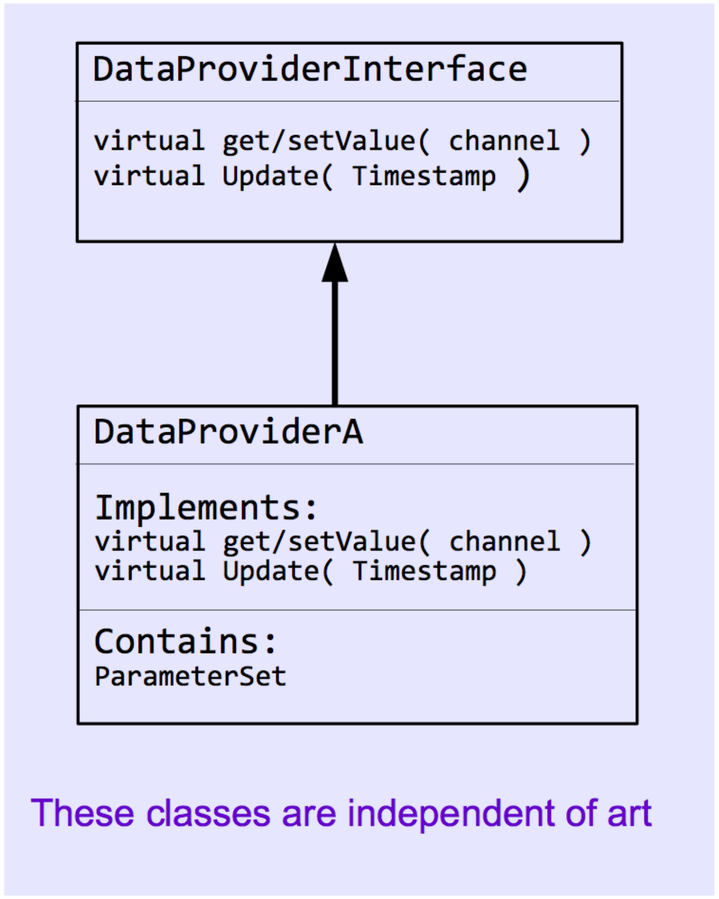
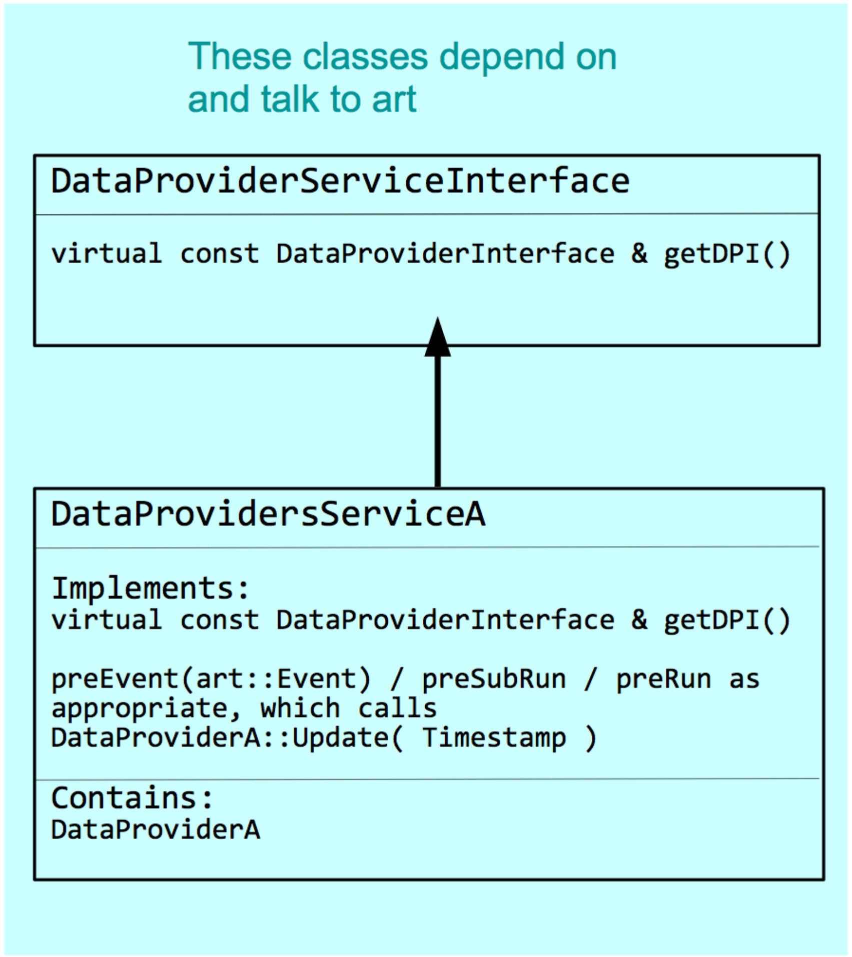

DataProviderInterface
virtual get/setValue(channel)
virtual Update(Timestamp)
DataProviderA
Implements:
virtual get/setValue(channel)
virtual Update(Timestamp)
Contains:
ParameterSet
These classes are independent of art

diagram by E. Snider

One service interface per database folder


These classes depend on and talk to art
DataProviderServiceInterface
virtual const DataProviderInterface & getDPI()
DataProvidersServiceA
Implements:
virtual const DataProviderInterface & getDPI()
preEvent(art::Event) / preSubRun / preRun as appropriate, which calls DataProviderA::Update(Timestamp)
Contains:
DataProviderA