

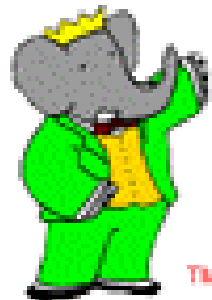
The Emt Configuration Packages

Philip Clark

University of Bristol



The BaBar Collaboration



Purpose of the packages

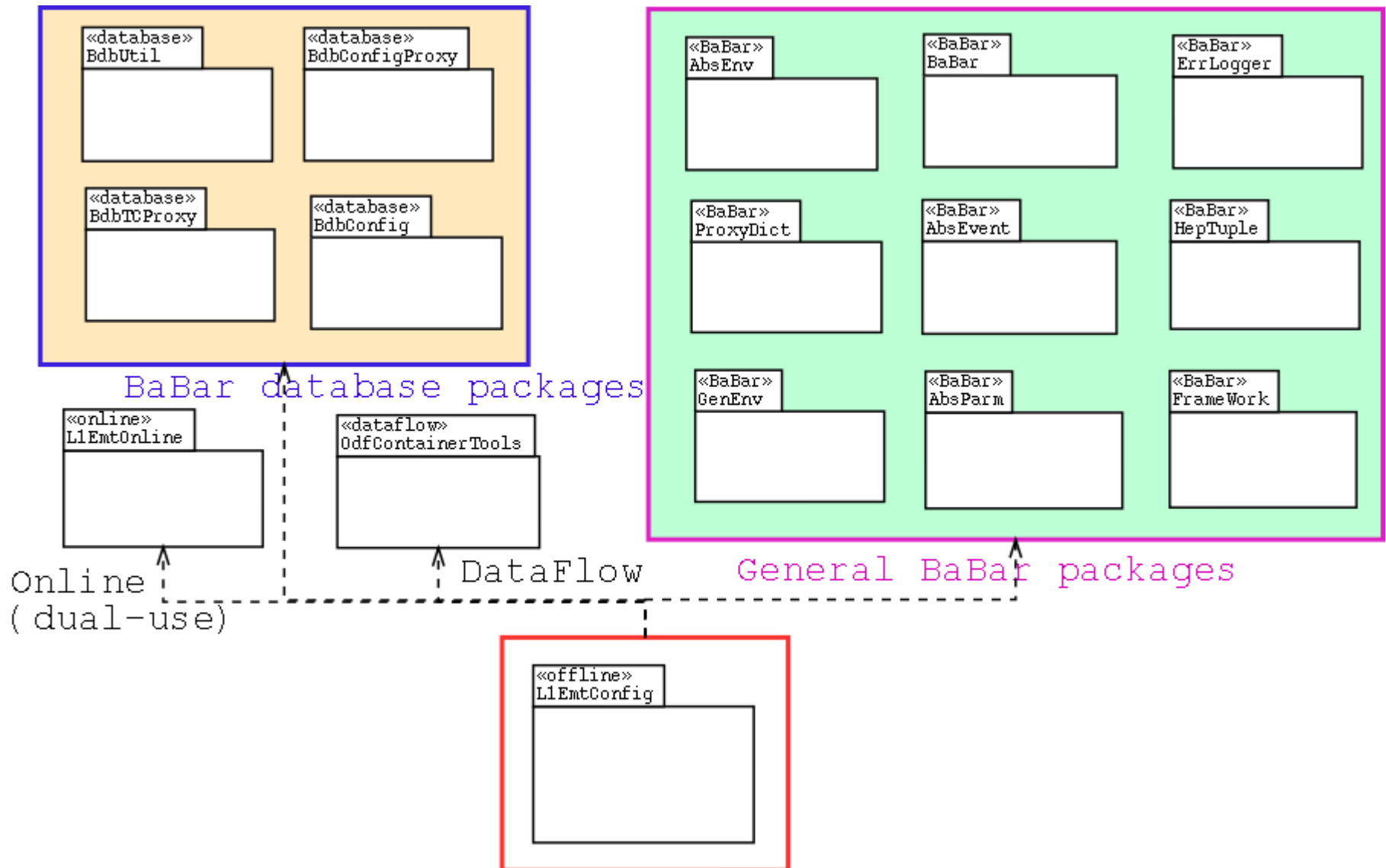
- Enable correct configuration of the Emt for datataking.
 - Generating configuraton xtc files from the database.
- Storage of the configuration data as persistent objects.
 - Provides exact record of the configuration used for every run.
 - Enables simulation to be made using exactly the same configuration as was used for data-taking.
 - Can get information into the Jas monitoring eg. overlay the tower mask on the occupancy plots.
 - Two main packages:- **L1EmtConfig** and **L1EmtConfigTools**.
 - **L1EmtConfig** contains all the classes used for the Emt configuration.
 - **L1EmtConfigTools** contains useful tools and utilities for working with the configuration.

L1EmtConfig

- Classes to produce persistent and transient objects containing our configuration data.
 - Data definition language (.ddl) files are very similar to normal C++ header files and ensure data is independent of platform type.
- Classes to generate configuration xtc files.
- Modules (classes derived from AppModule) used by L3 to get tower mask into configDict.

GNUmakefile	L1EmtConfigLoader.cc	L1EmtFcxTCProxy.cc
History	L1EmtConfigLoader.hh	L1EmtFcxTCProxy.hh
L1EmtAxRawDataP.cc	L1EmtConfigMonitor.cc	L1EmtFcxRawDataP.cc
L1EmtAxRawDataP.ddl	L1EmtConfigMonitor.hh	L1EmtFcxRawDataP.ddl
L1EmtAxRecord.cc	L1EmtConfigTCProxyLoader.cc	L1EmtFcxRecord.cc
L1EmtAxRecord.hh	L1EmtConfigTCProxyLoader.hh	L1EmtFcxRecord.hh
L1EmtAxTCProxy.cc	L1EmtFcxRawDataP.cc	L1EmtFcxTCProxy.cc
L1EmtAxTCProxy.hh	L1EmtFcxRawDataP.ddl	L1EmtFcxTCProxy.hh
L1EmtConfigInstantiate.cc	L1EmtFcxRecord.cc	bin_L1EmtConfig.mk
L1EmtConfigInstantiate.hh	L1EmtFcxRecord.hh	link_L1EmtConfig.mk

Dependencies of the package

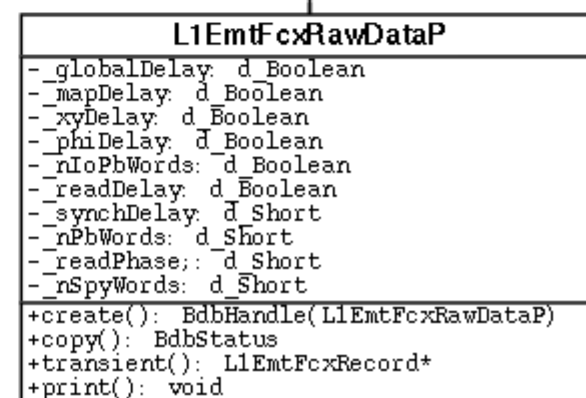
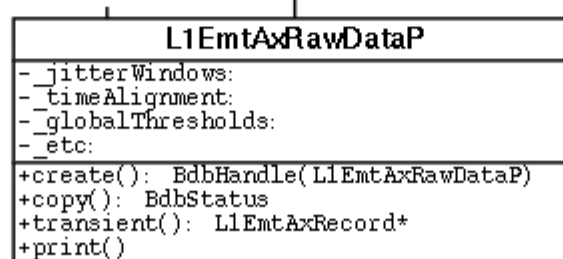
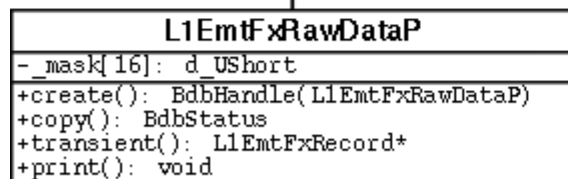
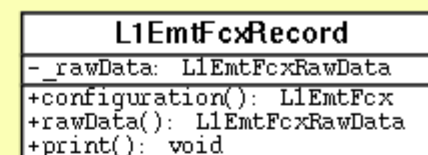
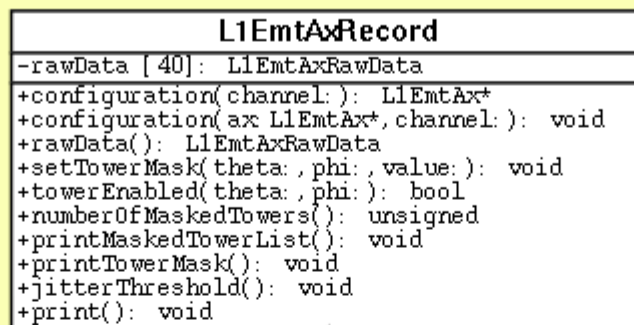
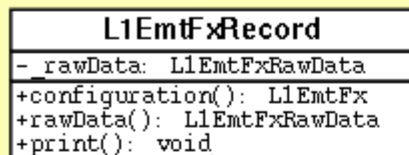


Configuration data classes

- Persistent capable classes
 - L1EmtAxRawDataP.ddl(cc)
 - L1EmtFxRawDataP.ddl(cc)
 - L1EmtFcxRawDataP.ddl(cc)
- Transient classes
 - L1EmtAxRecord.ddl(cc)
 - L1EmtFxRecord.ddl(cc)
 - L1EmtFcxRecord.ddl(cc)

- RawDataP classes store the configuration data and they inherit from BdbConfigObject.
- Transient classes are needed by the persistent classes to make a transient copy of themselves.

Transient data



Persistent data

The persistent objects

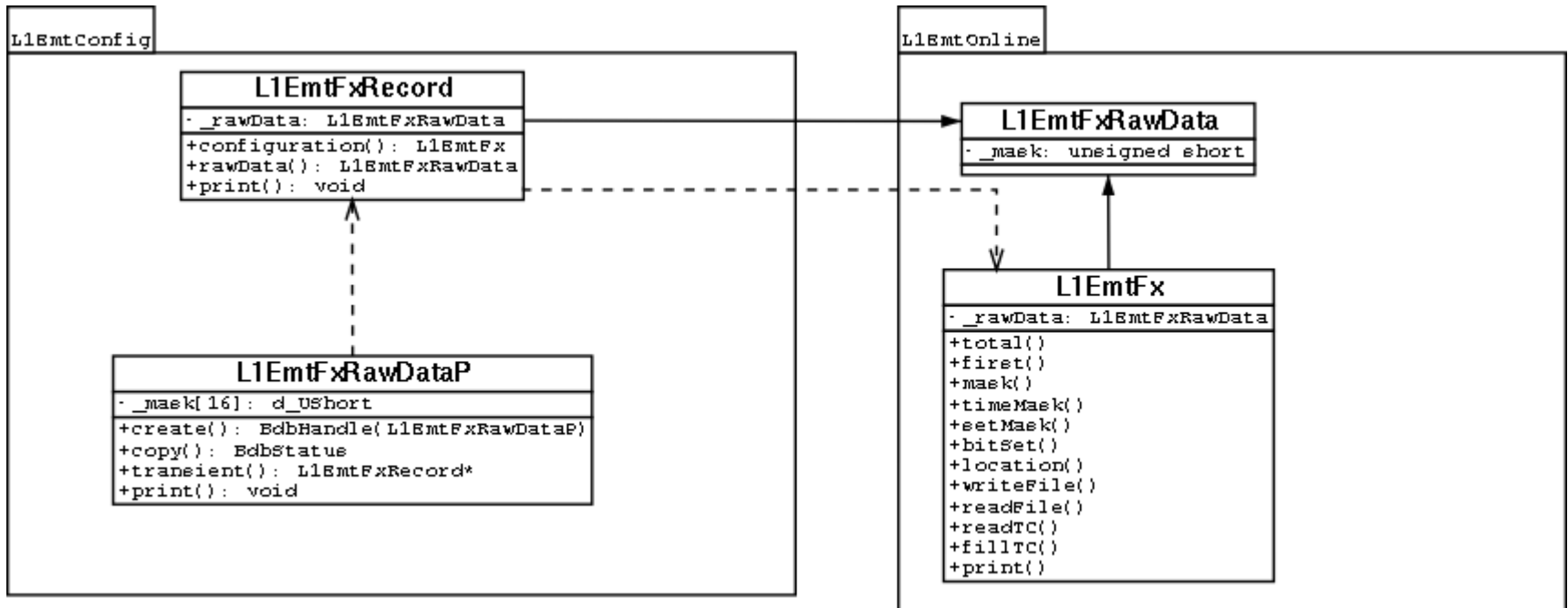
- The L1EmtxxxRawDataP class data:-
 - Contain a private copy of the configuration data.
- Class methods:-
 - create()
 - Factory method for the creation of the persistent objects.
 - copy()
 - Can make a copy of the persistent object.
 - transient()
 - Creates a transient copy of the persistent object.
 - Print()
 - Prints out details about the persistent object such as creation time and who it was created by.
 - Can overload this with some extra Emt stuff if we want.
- The name RawDataP is confusing.

The transient objects

- The L1EmtxxxRecord class data:-
 - Contain a private L1EmtxxxRawData object
- Class methods
 - configuration()
 - Returns a L1Emtxxx object
 - rawData()
 - Returns a L1EmtxxxRawData object.
 - Print()
 - Prints out whatever details we want.
 - Currently uses the configuration() method to create L1Emtxxx object and then uses its print() methods.

The L1EmtAxRecord object has been modified substantially to manipulate its member data to do things like mask towers, introduce phi dependent configurations. We may want to do some of these things in L1Emtxxx classes instead and keep the transient classes as tidy as possible.

An example:- The Fx Configuration



The L1EmtxxxTCProxy classes

- Three TC proxy classes for each of the Fx, Fcx, and Ax configuration classes.

L1EmtAxTCProxy.hh(cc) L1EmtFxTCProxy.hh(cc)
L1EmtFcxTCProxy.hh(cc)

- They inherit from the BdbTCProxy class.
- They can make an XTC file as required
 - fillXTC() method which is overloaded for our usage (see next page).

fillXTC() method

```
L1EmtAxTC* L1EmtAxTCProxy::fillXTC(const BdbHandle(L1EmtAxRawDataP)&
    pObject, const RdfConfigKey& rdfKey,
                                   odfArena& theArena)
{
    // create new XTC object
    L1EmtAxTC* axTC = new (theArena) L1EmtAxTC;
    L1EmtAxRecord* axRec = pObject->transient();
    for (unsigned phi(0);phi<40;phi++){
        L1EmtAx* ax = axRec->configuration(phi);
        ax->fillTC(axTC,phi);
        delete ax;
    }
    delete axRec;
    return axTC;
}
```

L1EmtConfigTCProxyLoader

- This registers all the necessary proxies for the TCs
 - The `proxies()` method.
- Must instantiate all the persistent objects that are going to be accessed by the application (ensures that virtual tables are loaded).
 - Don't know much about this -- read about it in the header of the base class ☺
 - The `vtables()` method.

The proxies() method

The proxies method takes a BdbTCProxyDict as its argument

```
Bool L1EmtConfigTCProxyLoader::proxies( BdbTCProxyDict* theDict )
```

```
{
```

Makes a cache object which can fetch configuration data

```
BdbConfigCache<L1EmtAxRawDataP>* axconfigCache =
```

```
    new BdbConfigTreeCache<L1EmtAxRawDataP>("Emt/AxConfig");
```

Creates a proxy method with can fill and XTC using on the cache data

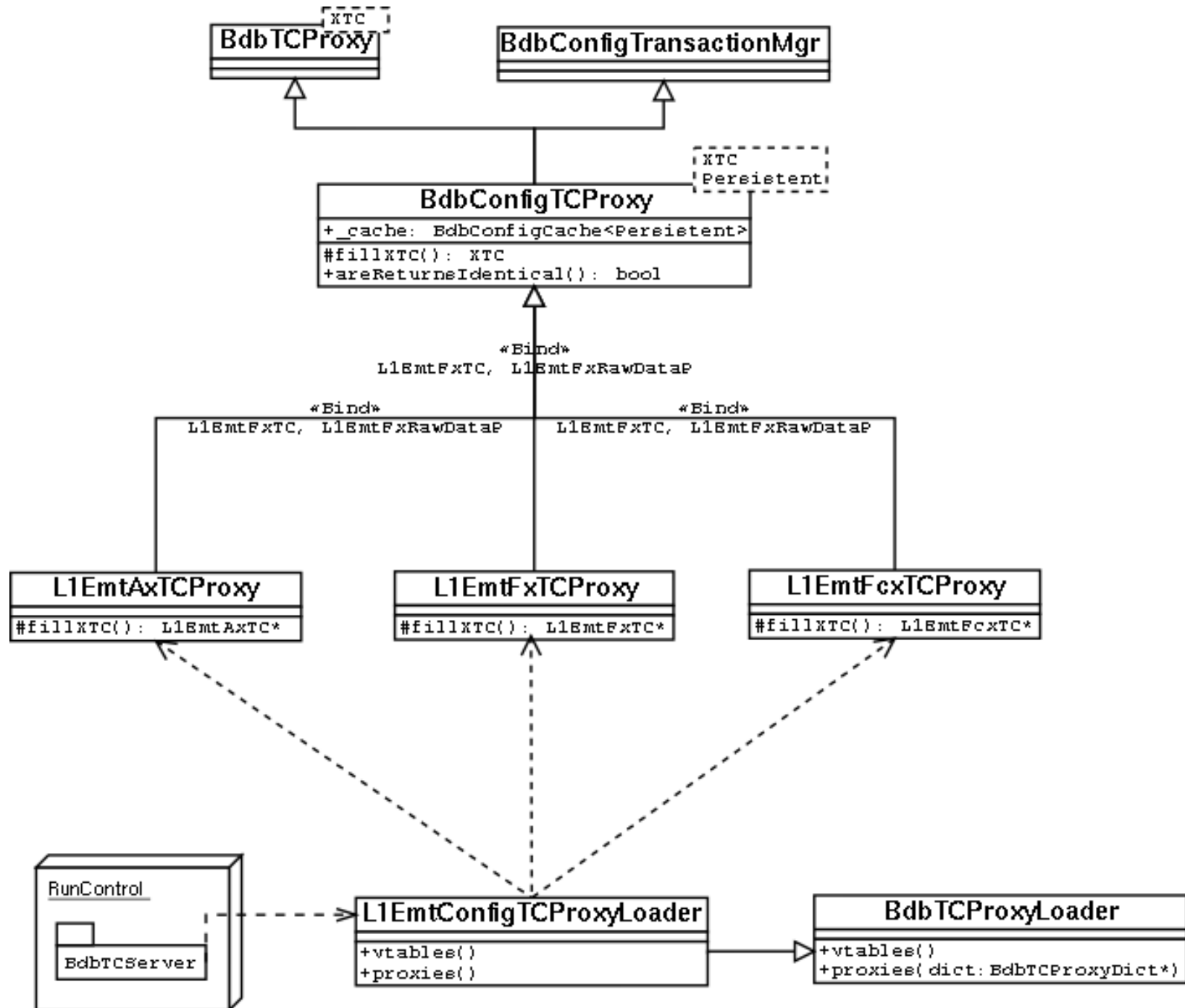
```
L1EmtAxTCProxy* emtAxTCProxy = new L1EmtAxTCProxy(
```

```
    axconfigCache );
```

Registers this proxy in the dictionary

```
bool axok = BdbTCDictFE<L1EmtAxTC>::
```

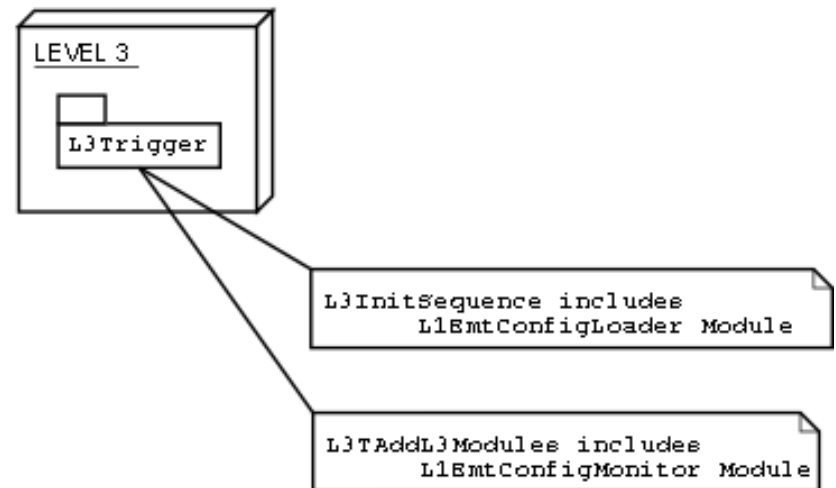
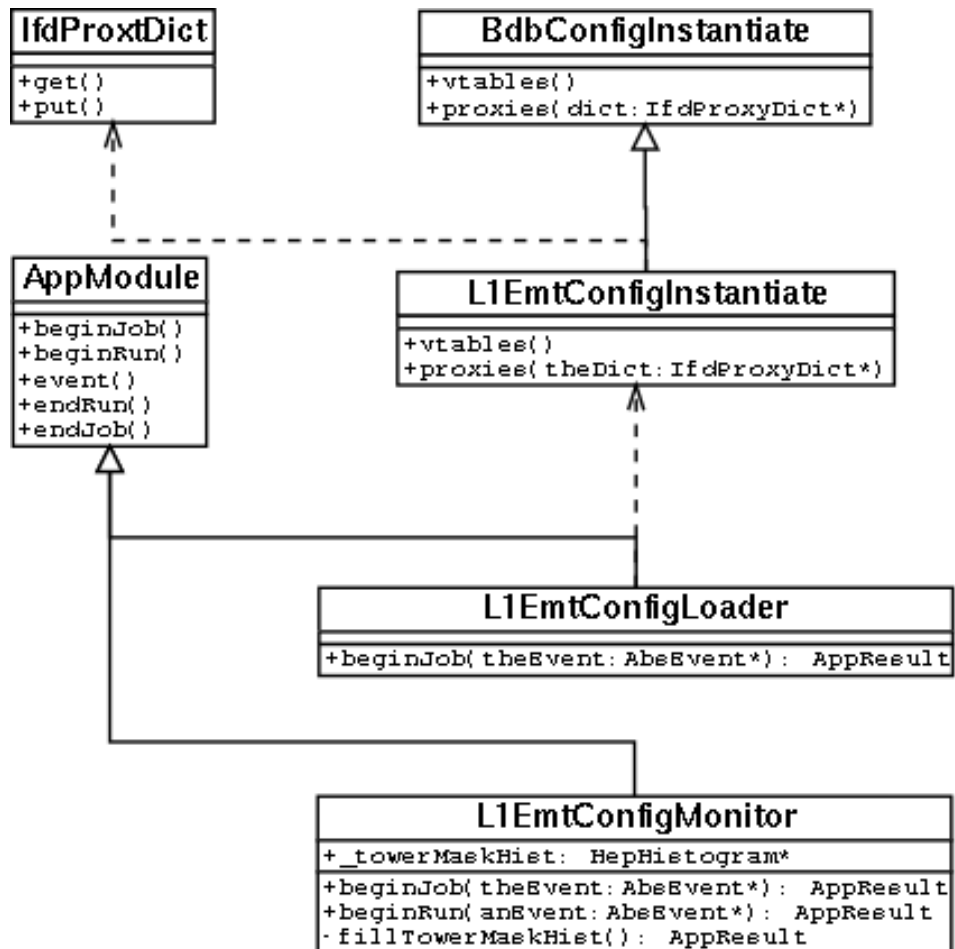
```
    put( theDict, emtAxTCProxy, lfdStrKey("Default") );
```



How monitoring gets the tower mask

- Class called **L1EmtConfigInstantiate**
 - Has a proxies() method similar to before. But this time instantiates proxies which can produce transient objects rather than TC files.
- Module **L1EmtConfigLoader**
 - Instantiates a L1EmtConfigInstantiate object
 - Uses its proxy method to register the proxy with the level 3 configuration dictionary (configDict) at beginRun
- Module **L1EmtConfigMonitor**
 - Books tower mask histogram
 - Fills tower mask histo at beginRun using the transient object it can fetch via configDict.
- Good example of how the config database can be used.
 - Perhaps the Emc or other subsystems could obtain a crystal mask to overlay in monitoring.

Modules/classes to get tower mask

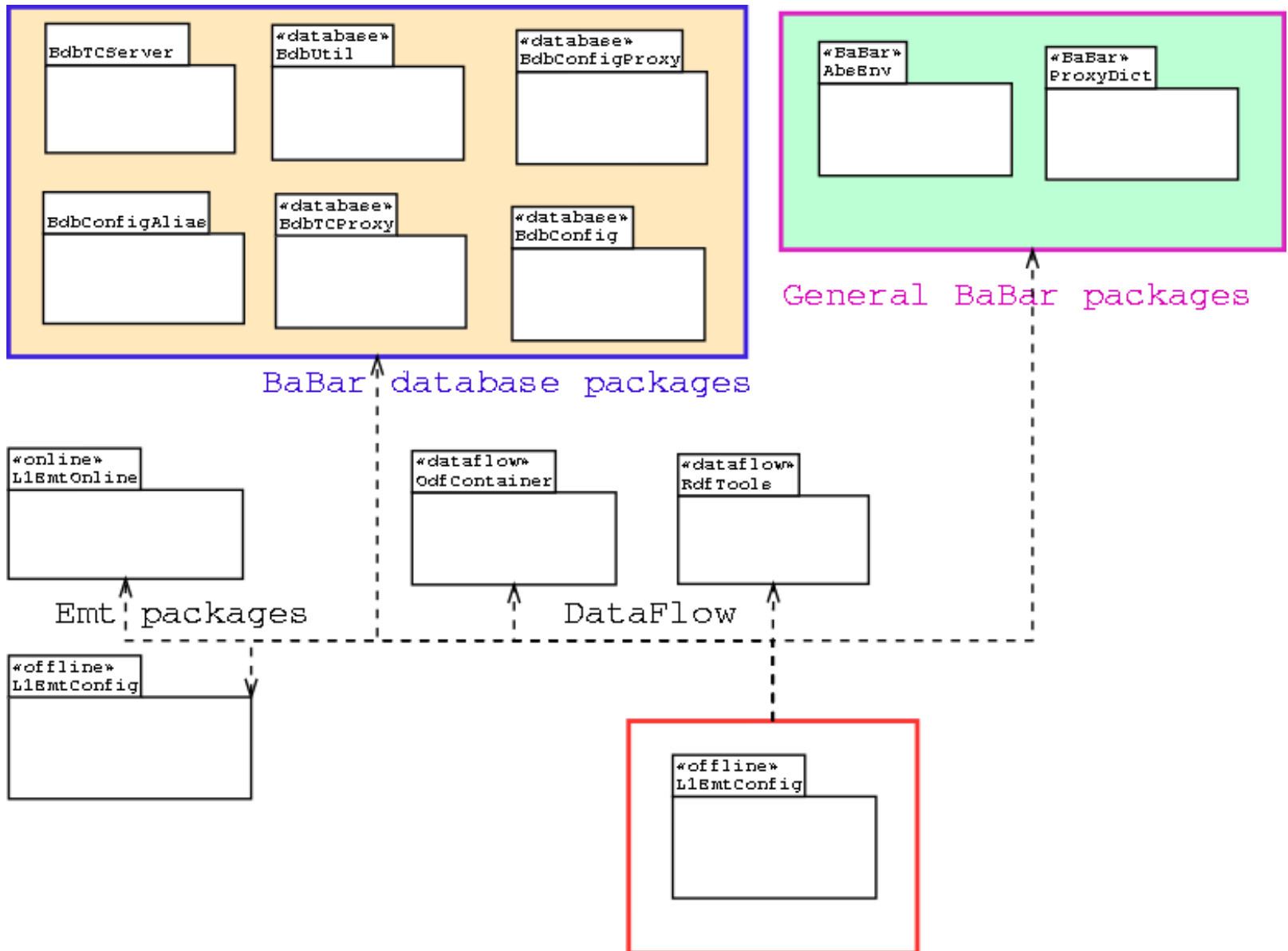


L1EmtConfig Tools

- Utilities to work with the configuration database and to also to decode configuration xtc files.
 - Can to take a configuration xtc and dump its contents.
 - Can create new persistent objects in the database.
 - Can extract a transient form of the configuration data, modify it, and then write the new data back into the database.
 - Can read the current configuration data out of the database.

GNUmakefile	L1EmtConfigListAliases.cc	L1EmtFxCfgReadConfigFromDb.cc
History	L1EmtDecodeAxConfigTC.cc	L1EmtGetAliasList.cc
L1EmtAxConfigWriteToDb.cc	L1EmtDecodeFcxConfigTC.cc	L1EmtReadAxTCFromDbToFile.cc
L1EmtAxConfigWriteToDbPhi.cc	L1EmtDecodeFxCfgTC.cc	README
L1EmtAxMakeTowerMaskHisto.cc	L1EmtFcxConfigWriteToDb.cc	bin_L1EmtConfigTools.mk
L1EmtFcxReadConfigFromDb.cc	L1EmtChangeTowerMask.cc	L1EmtFxCfgWriteToDb.cc
L1EmtAxReadTowerMaskFromDb.cc		

L1EmtConfigTools Dependencies



Decoding configuration TCs

- The programs to dump the configuration xtc files are
 - L1EmtDecodexxxConfigTC.cc
- They all operate using templated classes from OdfContainerTools
 - The xtc file is streamed into the program using the read() method in the class odfContainerTypedIO
 - This returns an odfXTCHandle object which is named xtcHandle
 - Memory responsibility is with the client here ie. us.
- Using the xtcHandle we create a L1EmtxxxTC object.
 - Use its print() method to dump its contents
 - Fills tower mask histo at beginRun using the transient object it can fetch via configDict.
- Then we delete the xtcHandle.

Writing persistent objects to the database

- Three programs to write new persistent objects to the database.
 - L1EmtxConfigWriteDb.cc
- The configuration parameters are hard coded in the src.
 - Errors can be introduced here due to the number of configuration parameters.
 - Care should be taken to create persistent objects with the correct configuration.
 - No methodology here to take account of phi dependent configurations such as the Ax configuration.
- Lastly these commands can take the secKey name as an option.
 - This is currently commented out. This avoids polluting the database with user mistakes.
 - At present it defaults to Ax, Fx, or FcxConfig with no spaces.

Procedure to write a persistent object

- Each program makes a L1EmtxxxRawData object which is then used to create a L1Emtxxx object.
- The configuration values are placed into this object (also printed to the screen).
- This is used to create a transient object.
 - L1EmtAxRecord transient(ax.rawData())
 - This constructor of L1EmtAxRecord simply duplicates the same configuration 40 times.
- Now we need to deal with the database....

The database transaction

First of all we create a BdbConfig object and start a transaction.

```
BdbConfig* theApp = BdbConfig::instance( )  
theApp->startUpdate( );
```

Make an object with the ability to write to the database (user in Emt config group).

```
BdbConfigWriter* theDatabase = new BdbConfigWriter( "Emt");  
theDatabase->open( );
```

Now we need to write a persistent object using the transient information.

```
BdbHandle(L1EmtFxRawDataP) theObject  
    = L1EmtFxRawDataP::create( transient, theDatabase, secKey );
```

If this is successful the details of the transaction are outputted to the screen.

The BdbConfigWriter object is then closed and deleted and the transaction to the database committed.

```
theDatabase->close( );  
delete theDatabase;  
theApp->commit( );
```

Phi dependent configurations

- **L1EmtAxConfigWriteDbPhi**
 - Takes the Ax configuration information out of the database.
 - Allows phi dependents configurations to be introduced by manipulating the transient object.
 - Writes a new persistent object into the database based on the transient object.
- To do this the program has a number of methods depending on whether you want to change jitterWindow, thresholds, FIR constants etc.
- We also have a more streamlined program called **L1EmtChangeTowerMask** which only (un)masks towers.

L1EmtAxConfigWriteDbPhi procedure

1. Accesses the Alias list to find the current PHYSICS top key.
 - Can also input this in decimal on the command line.
2. Starts a database transaction similar to before.
 - a read transaction this time.
3. Create a cache object that can fetch the config information and a proxy.
4. Registers the new proxy with ProxyDict
5. Using the Proxy dictionary as an interface we make a transient object using the top key obtained previously.
6. Make some changes using a menu driven system.
7. User can check correct configuration and abort if necessary.
8. Changes can be committed to the database and a new top key made.

Other utilities

- We have a series of programs called
 - L1EmtAxReadTowerMaskFromDb.cc
 - L1EmtFxReadConfigFromDb.cc
 - L1EmtFcxReadConfigFromDb.cc
- These are very useful for quickly the configuration for any top key.
- Useful to know which top keys are currently being used in the aliases.
 - L1EmtConfigListAliases.cc
 - This lists all the EMT and ORC aliases. This very useful and will also be handy for dealing with Alias Trees in the future.

Future improvements

- Some software rot has occurred.
 - L1EmtAxRecord has broken away from the design to be able to deal with phi dependent configurations.
 - Need to work out if this is the best way. It currently has four different constructors.
- Would be useful to have a L1EmtAxReadConfigFromDb that doesn't dump the configuraton 40 times, but instead dumps only the phi dependence and corresponding configuration.
- A few files could be removed.
 - L1EmtGetAliasList is no longer necessary.
 - L1EmtAxMakeTowerMaskHisto is defunct.
 - L1EmtReadAXTCFromDbToFile. Not sure if this is needed. It currently is in EXTRABINS and I don't think it builds.

Future improvements continued

- Another big improvement for the future would be to
 - Write tools to take the Fx and Fcx data from database to modify it and then write back in the configuration. Less mistakes would creep in this way.
 - Can we use the L1Emtxxx classes to do the the RawData manipulation rather than complexifying the Record classes like we have done for AxRecord.
 - Perhaps rename L1EmtxxxRawDataP to L1EmtxxxRecordP which is more intuitive.
- Generally everything is okay, but the code could be tightened up, and some nice new features introduced.
- There could be other things we might want to write if the Alias Trees get deployed in the future.

Summary

- We have covered the Emt configuration.
- The trigger is the only subsystem to currently use the persistent form of the configuration database. It would be nice if the other systems can do this as well.
- We should help as much as possible dealing with anything necessary for the alias tree change.
- You will see some of what we have covered here mentioned later where it is used for monitoring and simulation.