# The Physical Design of the CDF Simulation and Reconstruction Software #A245

presented by Liz Sexton-Kennedy, Fermilab

* Logical vs. Physical Design

* CDF's Physical Design

* Rules and Guidelines

* Achievements

CHEP 2000 Padova, Italy
February 10, 2000

Liz Sexton-Kennedy
10-Feb-2000

# Logical vs. Physical Design

## Logical design deals with language constructs

* Classes  * Inheritance  * Types of Data Members  * their Visibility

## Physical design addresses the issues involving

* Header and Source  Files * File Placement   * Directories
* Libraries  * Compile and Link Time Dependencies Between Them

## Classes can be physically coupled in varying degrees

### ➡Full Coupling

* the Header file of one class must include the header file of another

### ➡Name-Only Coupling

* a forward declaration is sufficient in the header file
* a forward declaration is sufficient in the source file

## Different logical design choices can lead to different levels of physical coupling.

## As systems grow larger attention to physical design becomes critical.

Liz Sexton-Kennedy
10-Feb-2000

# CDF's Physical Design

The code system is large, with about 1.3 million lines of code organized into 148 packages.

The packages are layered into a hierarchical tree of dependencies and can be grouped into categories

It is critical that our developers be aware of these categories so that they know how to structure their code and avoid generating cyclic dependencies

Binary Packages - intended to produce a binary like Production, 7

Module Packages - can't depend on each other, det. or obj., 24

Algorithm Packages - rigidly ordered dependencies, 33

Data Object Packages - EDM and geometry objects, 17

Interface Packages - CDF compatible interfaces to externals, 12

Infrastructure Packages - framework, EDM, and database 29

Standard Utilities and Services Packages - generator, Zoom/CLHEP, ROOT, tcl, ect. 26

# Rules and Guidelines

The two most important rules for keeping the physical design clean are :

* Packages may not cyclically depend on each other
* Use name-only dependencies wherever possible

The following rules are strictly followed:

* Keep Class data members private
* Avoid global data, and whenever possible avoid class static data
* Avoid using preprocessor macros in header files
* Use predictable include guards around the contents of header file
* Do not use "using" declarations in header files
* Enums, typedefs, constants should be defined within class scope
* Only classes and inline functions should be defined in header files
* Each header and source file pair should define one class
* If you fully depend on a class use it's header file, not local def'n
* Follow our naming conventions for files and language components
* Do not use C-style casts
* Don't use condition comp. clauses that change the size of an object
* Use of third party software must be approved
* Class documentation should appear in the header and must include the author's name

The LXR code browser had be an invaluable tool in finding classes and fixing problems after they occur.

# Achievements

CDF has been able to avoid cyclic package dependencies. All packages fit into a fixed hierarch that we can capture in a make file fragment.

Since they are hierarchical, packages need only specify their immediate dependencies.

We are able to link all CDF offline applications statically in one pass. The ordering is fixed, making it much easier for end users to link their own jobs.

None of the above was true in Run I.

There are no classes that, when changed, will trigger massive recompilation of the whole system. There are some base classes which can affect a whole category, such as AppModule or StorableObject, but these are the exceptions.