CodeGen Design Notes and Status
CDF Note 5641

Dennis Box, Jack Cranshaw, Paolo Calafiura, Rick St. Denis, Jim Kowalkowski

August 14, 2001

Contents

1 Introduction

A C++ Code generator has been implemented to simplify database access for calibration consumers and analysis code for Run II. Users specifiy and generate database tables and C++ accessor code from simple java classes. An overview of this systems design goals, structure, features, and planned enhancements is given.

2 Architecture Overview

2.1 Simplifying Assumptions

- Read and write access patterns are known ahead of time [1] and can be embedded in code templates
- A useful 1 to 1 mapping can be made from C++ objects to database tables, names of tables and generated access objects are the same except for capitalization.

2.2 Data Flow

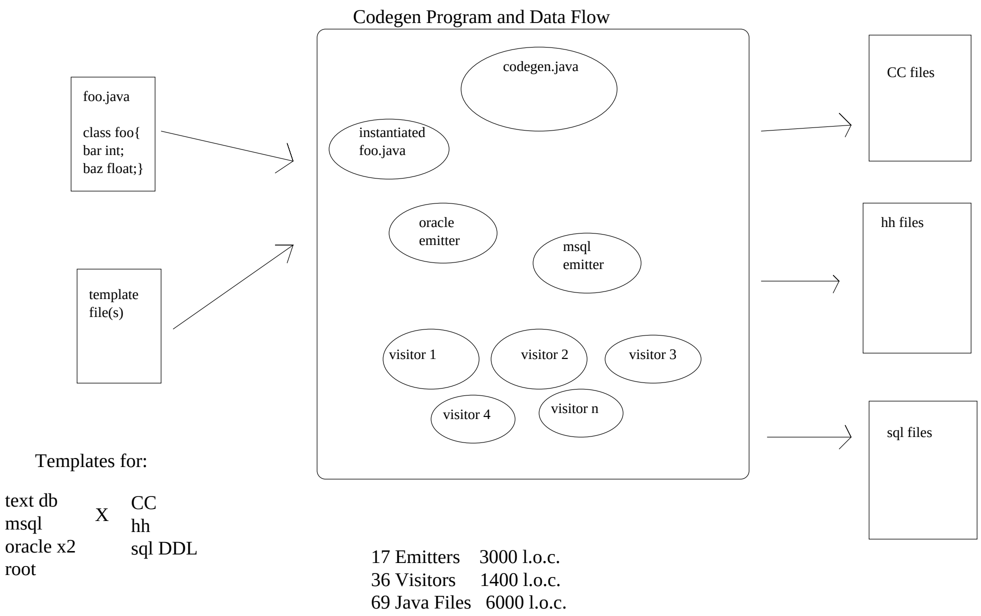

Figure 1: Program Structure and Data Flow (l.o.c.: lines of code)

Data flow starts with a simple java description file (see foo.java in figure 1) and 'template files' which resemble C++ or sql files with place holders for table and variable names. The codegen.java program instantiates an instance of foo.java inside a tree-like structure called a dictionary (figure 2), which program objects implementing the visitor pattern[5] inspect using the java reflection mechanism[6]. Emitter objects use the parsed structure of the description file to generate C++ and SQL for database access.

2.3 Components

A famililarity with the CDF software repository and build system [3] is assumed in this document. It is often helpful to use the CDF Code Browser [2] to cross reference makefiles and java files while learning this systems idiosyncracies .

- AACodeGen Package — This package contains the java files which do the actual code generation . It implements the large central box of figure 1 with emitter and visitor ovals in it.

  This package was modeled on the visitor-acceptor design pattern [5]. Visitor-acceptor patterns are normally applied when a network or tree of nodes, the acceptors, need to have operations performed on the data they contain, which is done by the visitors. Iterator classes are often used to process all the nodes and apply the visitors.

  Using this model, the java classes are broadly separated into 4 categories:

  - The Acceptors – Subclasses of Attrib.java, Component.java, DictEntry.java, and ROattrib.java are the 'nodes' in the dictionary and contain the actual representations of the data that are to be generated into C++ classes and SQL tables.
  - The Visitors – There are 35 of these, they all have names like xxxxVisitor.java. The operations they perform on the nodes is typically determining the type (int, float, string etc) of the data members, and prepending or appending code fragments as appropriate to the database. The best documentation for what these actually do is, unfortunately the code itself. A 'gmake jdoc' command will re-generate the documentation web pages which can then be read directly, most of visitors are commented as to what they actually do.
  - The Iterators – There are 15 of these, they are called emitters in this package. These emitters perform several important tasks in addition to assisting the Visitors in iterating through the Acceptor nodes. They also analyze 'template' files, searching for tokens to be replaced with output from the visitor-on-node operation.
  - The Utilities – miscellaneous file handling and string helper classes.

- CalibDB Package — The template files are in subdirectories of CalibDB/templates. There are currently subdirectories for msql, otl(oracle template library), text, and root databases. This corresponds to the box on the bottom left hand side of figure 1.

- XXXXDBTables Package(s) — XXXXDBTables/tables directory, where 'xxxx' represents various components. A quick look at the CDF code browser [2] reveals a COTCalibDBTables directory for the Central Outer Tracker, a CTCCalibDBTables directory, and many more. These packages correspond to the box on the top left hand side of figure 1, which show a single foo component being generated.

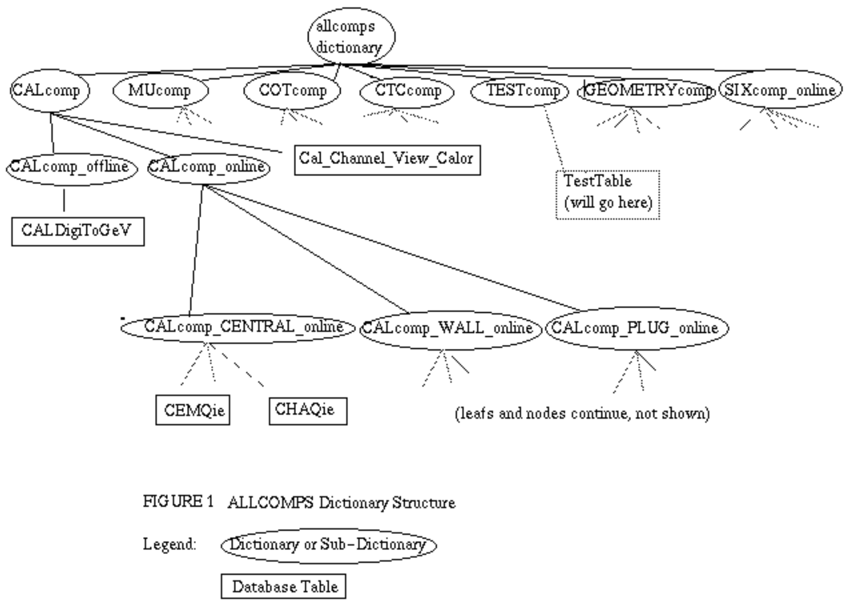

Figure 2: Dictionary Structure

The code that is generated goes into a temporary directory off of the SRT_LOCAL/tmp{machine-architecture} directory. For instance, if the developer were using KCC 4.0 on a Linux machine and developing in the fooBar directory, the generated code would go in a subdirectory of fooBar/tmp/Linux2-KCC_4_0/ This is explained further in the usage document.[4]

3 Read-Only Access With Code Gen Lite

If the java acceptor nodes are made to inherit from the 'ROattrib' class instead of 'Attrib', the resulting C++ code is can perform very flexible SELECT's, but no INSERT's on its database table. The SELECT statement is modified by means of an ROKey, which takes a WHERE clause as its only argument, and appends it to the default select statment for the table. . See the DBViews package, or the DBUtils/Examples/readView subdirectory for working examples.

4 Adding a Feature to the Code Generator

Adding a feature usually involves the following steps:

1. Determine what modified C++ or SQL code should look like

2. Modify templates
3. Modify/add visitors
4. gmake codegen, inspect C++
5. IF (not happy with result) GOTO 1
6. compile C++
7. test compiled program
8. IF (not happy with result) GOTO 1

5 Development History

- Originally implemented by Paolo Calafiura in 1998. Used IDL as class description language, which was parsed with FIDLE package and output to C++ using perl as a filter.
- Re-written by Paolo Calafiura in 1999. Used simple java classes to describe desired output, and reflection, visitor design pattern and 'emitters' to generate C++ and SQL for text and Msql databases.
- Features added by Dennis Box and Jim Kowalkowski to generate access to Oracle. Integrated into CDF offline nightly build system. Original architecture was sometimes bypassed, resulting in both visitor-dispatched and if-then-else dispatched code emitters. Recent features include:
  - Replaced Oracle OCI (Oracle Call Interface) access layer and templates with OTL (Oracle Template Library) interface
  - changed static linking to shared object linking of Oracle libraries
  - added support for multiple schema/database/server access
  - float/int/byte array support
  - modifications to java description files to support database loading simulations
  - read-only (lite) access access simulation.
- Features Desired but not yet Added:
  - address code bloat issue - generated template code produces large binaries with nearly unmanagable number of linker symbols
  - rewrite to reduce number of visitors and emitters
  - get rid of msql interrface, replace with odbc to support generic database

6 Notes on Components of AACodeGen

6.1 Package emitter.visitor.countFieldVisitor

Interface Summary	
countFieldVisitor	Interface definition for the field visitors - These objects operate nodes using java.lang.reflect.Field methods - Implements the emitString method.

Class Summary	
CountLengthFieldVisitor	Superclass for lengthFieldVisitors - These visitors know how to find the length of strings and arrays embedded in the Attrib nodes.
enumC2TypeVisitor	this field visitor makes the insert_sql[] statement in the OTL template - used by otlEmitter
enumCTypeVisitor	this field visitor makes the insert_sql[] statement - used by otlExportCCEmitter
enumFieldVisitor	a field visitor that enumerates on its fields returning something like ":0 :1 etc " - used by otlEmitter
readrowFieldVisitor	a field visitor returning something like "[0],"%d",_channel - used by textCCEmitter

6.2 Package emitter.visitor.EntryFieldVisitor

Interface Summary	
EntryFieldVisitor	interface for the field visitors - has visit() methods for both nodes in a tree and fields in the nodes.

Class Summary	
AttribListVisitor	a component visitor which creates a list of the nodes and fields it finds - used in codegen, codegen_lite and codegen_export.

6.3 Package emitter.visitor.FieldVisitor

Interface Summary	
FieldVisitor	Interface definition for the field visitors.

Class Summary	
compareFieldVisitor	generates the compare() method in the hh file - used by hhEmitter
cTypeVisitor	a field visitor returning a c language type like “float” or int for each field - used by otlcolsFieldVisitor
ddlFieldVisitor	a field visitor returning something like “CHANNEL int” - used by msqlEmitter and textEmitter
ddlOracleFieldVisitor	subclass of ddlFieldVisitor to handle ’features’ of oracle ddl - used by otlEmitter
declareFieldVisitor	a field visitor returning something like “float _channel” - used by ExportDBhhEmitter and SimplehhEmitter amd hhEmitter and otlExportCCEmitter
declareFieldVisitorConst	a field visitor returning something like “float _channel” Environment: developed for the javagen code generator project at LBL
dumpFieldVisitor	a field visitor returning something like “ “channel = “ << t._channel” - used by hhEmitter to make the ostream dump() method in the hh file
equalFieldVisitor	a field visitor returning something like “channel = copy.channel” for each field in node - used by hhEmitter and SimplehhEmitter and ExportDBhhEmitter
FieldLengthVisitor	field visitor that implements getLength() method on fields - used by enumCTypeVisitor and enumC2TypeVisitor , subclassed by ddlFieldVisitor and otlInputNameFieldVisitor and setRawFieldVisitor and setMemberFieldVisitor and

	otlcolsFieldVisitor and otlNameFieldVisitor
getterFieldVisitor	a field visitor returning something like "long channel() const { return _channel;}" used to make accessor methods in hh classes - used by hhEmitter and SimplehhEmitter and ExportDBhhEmitter
hhConstructorFieldVisitor	a field visitor returning something like " int channel" - used by hhEmitter to make the Struct Raw in the generated hh file
ioNameFieldVisitor	a field visitor returning something like "channel" unless its an array of bytes, ints or floats in which case it wraps the type with conv_util:: functions to convert it to a hex string - used by hhEmitter
ioNameFieldVisitor2	a field visitor returning something like "channel" - used by hhEmitter
lessThanFieldVisitor	visitor returns something like " foo() < x._foo " - used by hhEmitter
msqlFormatFieldVisitor	a field visitor returning something like "%%d" - used by msqlExportCCEmitter
msqlNameFormatFieldVisitor	a field visitor returning something like "CREATION_TIME = %d" - used by msqlExportCCEmitter
nameFieldVisitor	a field visitor returning something like "channel" - subclassed by hhConstructorFieldVisitor , ioNameFieldVisitor , and ioNameFieldVisitor2 - used by otlcolsFieldVisitor , ExportDBhhEmitter , SimplehhEmitter , SimpleotlCCEmitter , msqlCCEmitter , msqlEmitter , msqlExportCCEmitter , otlEmitter , otlExportCCEmitter , textCCEmitter , and textEmitter

otlcolsFieldVisitor	this visitor makes public data members in the OTL.hh.template - used by otlEmitter.java
otlInputNameFieldVisitor	generates the otl input stream, handling special cases like converting vectors of ints floats and bytes to hex streams - used by otlCCEmitter
otlNameFieldVisitor	generates the otl output stream, handling special cases like converting vectors of ints floats and bytes to hex streams - used by otlCCEmitter
otlNameFormatFieldVisitor	a field visitor returning something like "CREATION_TIME = %d" -used by otlExportCCEmitter
printfFieldVisitor	a field visitor returning something like "name().c_str" or "channel" - used by textCCEmitter , otlExportCCEmitter , msqlExportCCEmitter , msqlCCEmitter
setMemberFieldVisitor	a field visitor returning something like "channel = copy.channel" - NOT USED BY ANYTHING? DELETE ?
setRawFieldVisitor	a field visitor returning something like "channel = copy.channel" in the struct Raw - handles byte float int arrays to stream io - NOT USED BY ANYTHING? DELETE ?
setterFieldVisitor	a field visitor returning something like "void setCOLUMN_NAME (COLUMN_TYPE COLUMN_NAME) { COLUMN_NAME_ = COLUMN_NAME; }" - used by hhEmitter
stringFieldVisitor	a field visitor returning a fixed string per field - NOT USED BY ANYTHING? DELETE ?
textFormatFieldVisitor	a field visitor returning something like "%%d" -used by textEmitter

valuedConstructorFieldVisitor	a field visitor returning something like "channel(achannel)" - used by ExportDBhhEmitter , SimplehhEmitter , hhEmitter

6.4 Package emitter

Class Summary	
Attrib	Describes an attribute (row) of the calibration data tables.
ccEmitter	Abstract base class for the c++ code emitters.
CCType	Static class mapping java to CC type conversion.
Component	Abstract class describing a node of the calibration tree.
DBString	a string type for databases and generated C++
DictEntry	Abstract base class for Attrib and Component classes.
emitter	Abstract base class for the code emitters.
ExportDBhhEmitter	Replaces tokens like 'Export_blah' in templates to form DBExport .hh files.
gWriter	encapsulates file access
hhccEmitter	emits the tokens for the attribute c++ header Environment: developed for the javagen code generator project at LBL
hhEmitter	emits and replaces in template file the tokens for the hh file
msqlCCEmitter	emits the tokens for the c++ embedded msql implementation
msqlEmitter	emits the msql statements for the CalibDB interface
msqlExportCCEmitter	Create and the replace the token strings in template to generate C++ code.
MSQLType	static class defining java to MSQL type conversion
nameEmitter	emits only the table name tokens.
otlCCEmitter	Replaces tokens in OTL.cc.template to generate Foo.OTL.cc .
otlEmitter	Replaces tokens in file pcalib.osql.template to generate DDL that creates Oracle tables.
otlExportCCEmitter	Used by codegen_export.java to replace tokens in a template file.
OTLType	Static class mapping java to OTL type conversion.
ROattrib	A subclass of Attrib with

	the sole purpose of alerting codegen.java to use ROtemplates instead of ordinary templates for code generation
SimplehhEmitter	Replaces tokens in hh.template to create file Foo.hh.
SimpleotlCCEmitter	Used by codegen_lite.java to replace tokens in XMonDBTables and RunConfigDBTables versions of otl.cc.template.
textCCEmitter	emits the tokens for the c++ embedded text implementation
textEmitter	emits the text statements for the CalibDB interface
TEXTType	static class defining java to TEXT type conversion

7 Conclusions

The Code Generator has proven useful to physicists who need to persist calibration data without learning the intricacies of multiple database interfaces. This usefulness has come at the cost of a system that is somewhat difficult to add features to and produces libraries that are larger than they need be. Despite this useful features have been added and the generated code will be used in Run II calibration and analysis.

References

[1] "A Calibration Database Schema for CDF II," Jack Cranshaw, Mark Lancaster, Jim Kowalkowski, Paolo Calafiura, Marjorie Shapiro, Dennis Box, Robert Harris. CDF Note 5088, 8/12/99 revised 2/12/01

[2] "CVS source tree for AACodeGen package, ", The CDF Collaboration. http://cdfcodebrowser.fnal.gov/CdfCode/

[3] "A UNIX based Software Management System," Robert Harris. CDF Note 3891,9/25/96 revised 6/24/99

[4] "CodeGen User Guide," Dennis Box, Paolo Calafiura, Jim Kowalkowski. CDF Note 5461 ,9/25/96 revised 6/24/99

[5] "Design Patterns Elements of Reusable Object Oriented Software," Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Addison-Wesley Publishing Co. 1995 ISBN 0-201-63361-2 pp. 331-344

[6] "Java In A Nutshell, 3rd Edition" David Flanagan. O'Reilly Publishing Co. 1999 ISBN 1-56592-487-8 Chapters 4 and 14