LBNE Software and Computing
Update for CAPTAIN Software Workshop

Brett Viren

Physics Department

BROOKHAVEN
NATIONAL LABORATORY

November 1, 2013

Contents

LBNE S&C Organization

Survey of Existing LBNE Software

Software Installation System

What I'm up to right now in LBNE:

- Developing the LBNE software installation system.
- Steering development of a formal LBNE S&C requirements document.
- Helping the LBNE Physics Tools and S&C coordinators.
- Checking out the CAPTAIN software effort!
- Trying hard to not be given an official title....

Software and Computing, Project and Collaboration

Project-funded and organized part (somewhat limited scope):

- Run-control/slow-control
  - Leverage IceCube experience
- DAQ
  - Leaning towards basing it on artdaq
  - Will get workout soon with the 35ton prototype detector
- Sundry engineering, administrative and "project" software.

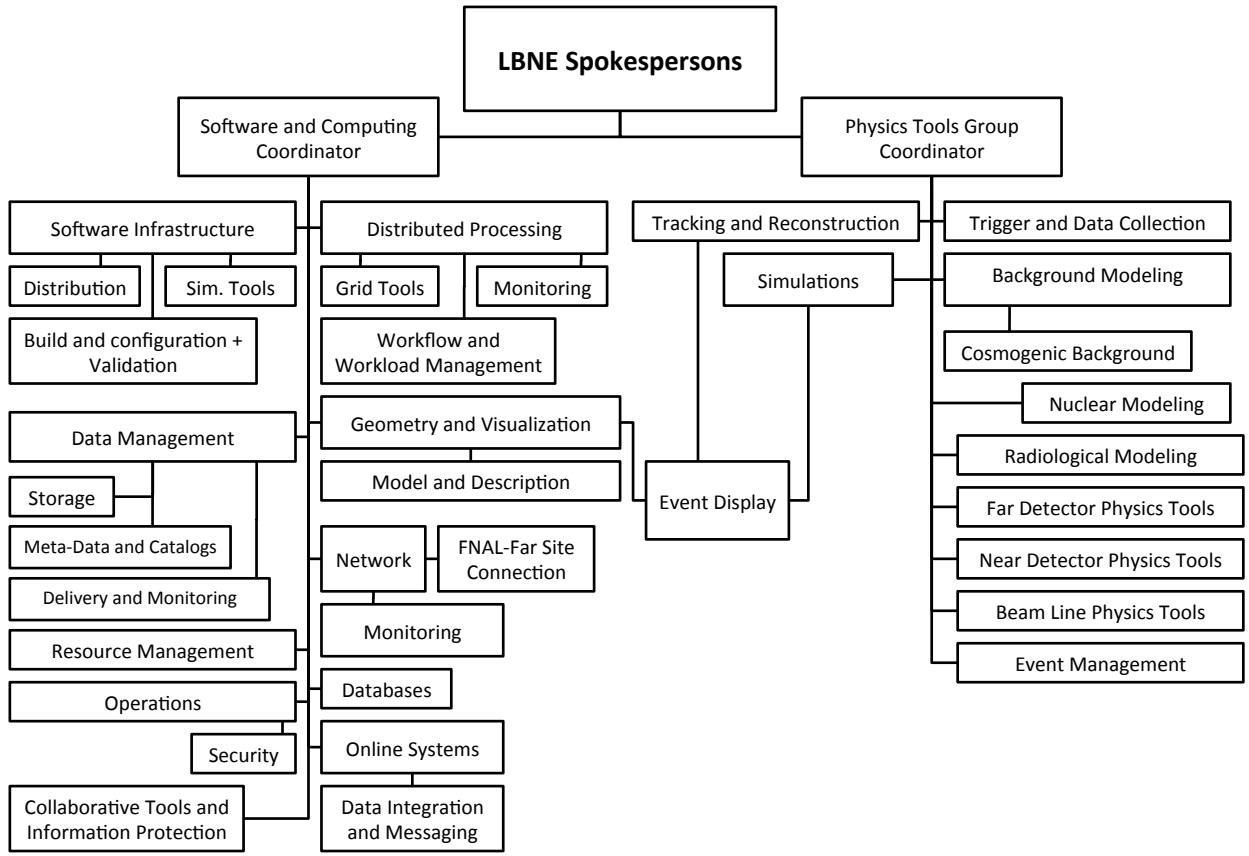
Base-funded, collaboration organized (everything else)

- Simulations, data reduction and reconstruction
- Data storage and distribution, databases
- Production processing, grid computing
- Installation, validation
- Collaborative tools (FNAL Redmine, mailing lists)

Focus on the collaboration software and computing $\longrightarrow$

Current S&C Org. Chart (WIP)

- The structure is still being defined and refined.
- Very few names have been identified so far.
- Doesn't show S&C Technical Advisory Committee in formation.

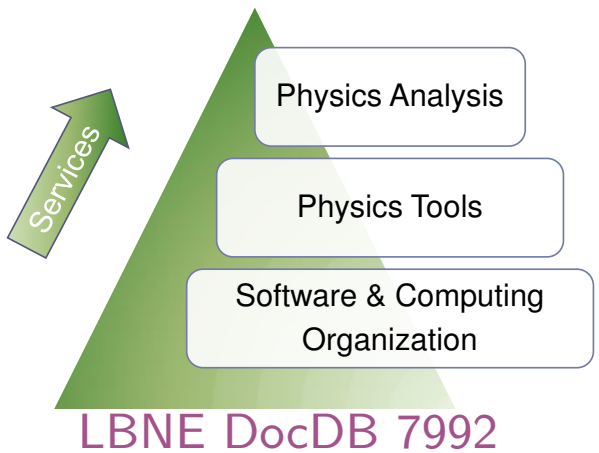


LBNE DocDB 7818

The Three Layer Cake of Support

Layered support structure

1. Produce scientific results
2. Production and verification of analysis tools
3. Interface with Project and coordinate collaboration S&C efforts



These three coordinator roles are spelled out in more detail in "LBNE Collaboration Organization Document", DocDB 6547

S&C Requirements Document

A recent effort to develop a somewhat formal "requirements document"

Purpose
- Knit together the fledgling S&C group.
- Organize things for ourselves as a collaboration.
- Inform DOE.

Scope S&C (not Physics Tools) side of things: Data, Databases, Software, Grid, Production Processing, Geometry, Visualization, Networking,...

Level Intended to list high-level needs, not dictate solutions.

Live document at:

Releases DocDB 8035

Working area `http://www.phy.bnl.gov/~bviren/lbne/sc/requirements/`

LBNE S&C Organization

Survey of Existing LBNE Software

Software Installation System

Beam Simulation

G4LBNE

- Stand-alone Geant4 executable
- Largely derived from on G4NUMI (MINOS)
- Target, horns, decay pipe, absorber.
- Mix of hard-coded C++ and GDML geometry.
- Redmine/CVS
  - expect to transition to Redmine/git

Fast MC

Fast MC = flux files + GENIE + GLOBES → ntuples + plots

- Ties together:
  - Flux files
  - GENIE
  - GLOBES
- Redmine/SVN
  - May move to Redmine/git

Far Detector Simulation and Reconstruction

LArSoft:

- Multi-experiment LAr-related code base
- Based on art framework
- Redmine/SVN
  - Currently an SRT-based project
  - Moving to Redmine/git and CMake

35ton prototype detector studies also being done with LArSoft

Near Detector Software

- An initial plan for "fast" and "full" simulations exists
- Effort needs to ramp up here.

LBNE S&C Organization

Survey of Existing LBNE Software

Software Installation System

LBNE Software Installation Problem Space

- Rather large code base: ∼40 externals, ∼5 LBNE sub-suites
  - build the build tools: CMake, GCC
  - usual large HEP packages (Geant4, ROOT, GENIE)
  - Python, Boost, gccxml, both PostgreSQL and MySQL, ...
  - A large framework: art + larsoft (+artdaq for online)
  - LBNE code: FD/ND/Beam/Fast simulation, reco

    → not all groups need (want) all software
- Multinational, distributed collaboration, expertise varies
- Multiple platforms: at least Sci. Linux, Deb/Ubu, Mac OS X
- Require, roughly in order of decreasing importance:
  1. Control versions independently from other experiments.
  2. Facilitate any collaborator building all relevant software from pristine sources on all supported platforms.
  3. Provide pre-built binaries.

But, above all: get the software into the hands of those that can contribute.

Problem Zero

Much of the existing software developed in a mono-culture (@fnal.gov)

- Building and using LBNE software across the entire collaboration is "challenging".
  - Largely has not been possible to this point.
- Building LArSoft/art is particularly daunting.
- The breadth of LBNE projects presents another challenge.

$\Rightarrow$ need a "simple" and "user friendly" way to install all the software.

Enter **worch**

The **worch** gestalt:

- Insert a layer between user and a package's native build mechanism.
- Aggregate required packages into a coherent suite.
- Distill all information required to build all packages in a suite into a central, human-oriented configuration file (or file set).
- Specify desired directory layout for temporary files and final install.
- Factor common installation patterns into reusable parts.
- Generate the commands to execute individual installation steps.
- Feed steps and their dependencies to an execution engine (**waf**).
- Highly automated: initiate a build and go do something else productive for several hours.
- Leave behind useful information to debug when things go wrong.

Layers can go on top of **worch** to manage suite-wide release issues (eg, via worch bundles) and continuous-integration systems.

High level **worch** Design

worch = **w**af + **orch**estration

The "orchestration" consists of:

configuration declaring all "knowledge" needed to install a software suite.

- interpreter of configuration producing per-package parameters and overall suite definition.
- features batteries-included set of methods driven by these parameters to perform *steps* of an installation.

Example use: ORKA's Geant4/ROOT application

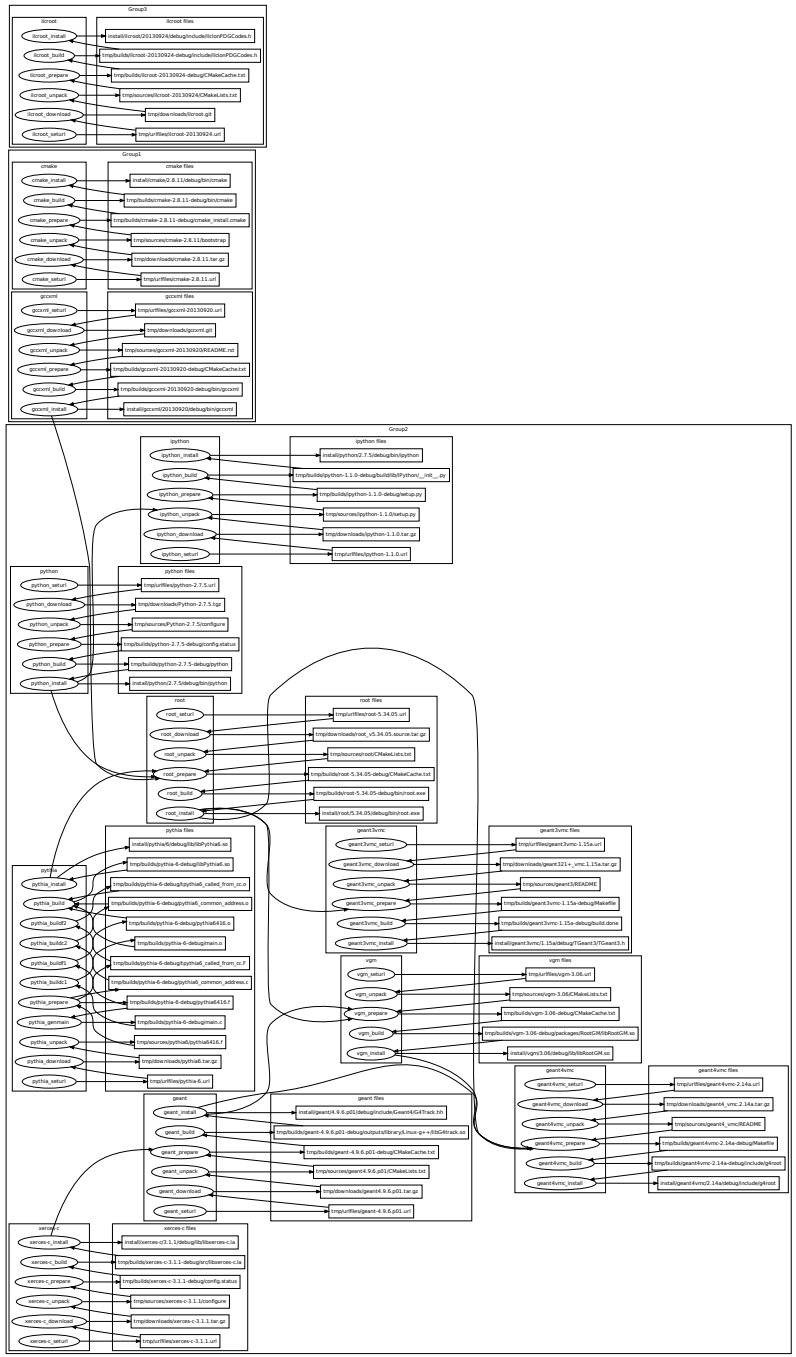
```
$ git clone https://github.com/brettviren/worch.git
$ cd worch
$ ./waf --prefix=/path/to/install \
        --orch-config=examples/orka/suite-ilcroot.cfg \
          configure build
...
'build' finished successfully (31m22.537s)
```

Builds some 10 externals + the main "ORKA-ILCRoot" app. $\longrightarrow$

Example: ORKA

1. cmake
2. gccxml
3. geant
4. geant3vmc
5. geant4vmc
6. ilcroot
7. pythia
8. python
9. root
10. vgm
11. xerces-c

Relatively small project. Figure shows build steps and their dependencies.



worch Configuration Language

- Syntax based on Python `ConfigParser` (aka "INI" files)
  - Lightly-structured (**not** YAML/XML'ish)
  - Purely declarative.
  - Titled sections holding set of key/value pairs.
  - No data types, everything is a string.
- Interpretation extends this to allow:
  - Variable references: `install_dir = {package}-{version}`
  - Hierarchical data structures via certain key's value interpreted to reference another section by its title.
  - Simple "`#include`" mechanism.
- **worch** adds additional interpretation:
  - Feature-specific required variables (with sane defaults)
  - Relative path resolution
  - Expression of inter-step dependencies and environment settings.

Interpretation causes all variables set in a section high in the hierarchy to be available, and possibly overridden, in lower-level sections.

worch Package "features"

A **worch** package feature is:

- a name in a package's `features` configuration item
- a reusable chunk of code which:
  - factors out common, reusable installation steps
  - interprets configuration parameters into **waf** rules

Example features:

tarball	ensures tar/zip file is downloaded and unpacked
vcs	as above but from git/cvs/svn/hg repos and based on a tag
patch	apply a patch file (which may be downloaded)
prepare	prepare source for building, (also "autoconf" and "cmake")
makemake	basically `make; make install` steps
etc	some specialized features for environment management systems (see next), tricky packages or ones lacking native build systems (pythia6), pure-Python packages.

Environment Management

End-user environment management with **worch**

- No special treatment, support as just another package "feature"
- Produces whatever files required by a particular env. mgt. system based on existing **worch** configuration
- So far, batteries-included for two systems.

Modules
- Free Software: `http://modules.sf.net/`
- Creates the (TCL) `modulefile` directly.

UPS
- FNAL-centric, `http://www.fnal.gov/docs/products/ups/`
- creates UPS "database" files w/out requiring UPS itself
  → but **worch** can/does build UPS
- In-build support for UPS required for building art

What is this **waf** thing???

waf is at `http://code.google.com/p/waf/`

- The name does not mean nor stand for anything
- Python (v2.3-v3.1) tool for building build systems
  - (**worch** currently tested with 2.6/2.7/3)
- Low-level `make`/`autoconf`/`cmake` replacement or a meta-layer on top.
- Configured in Python via "`wscript`" files (think `Makefile`)
- Provides a task scheduling and dependency engine.
- Has batteries-included for low level build patterns (C/C++ compilations, library linkage, LaTeX, etc)
- Packaged as a single, self-contained, downloaded Python executable
  - → can simply include the **waf** executable bundle in with the source (note: **worch** has an additional bundling method: **waf** + **worch** + configs)

Aside: Synergy with ATLAS

ATLAS moving to **waf** for build and configuration management

- Replace CMT/LCGCMT with **hwaf** and **waf**

  hwaf user command line (Go) executable for setting up environment, initiating user analysis packages, building

  waf replacing CMT for building experiment packages

  worch replacing LCGCMT builders for externals

- Work being done by Sebastien Binet of LAL/IN2P3
  - frequent contributor to **worch** development

Once the basic LBNE installation system is working I will look into **hwaf** more deeply.

Software Installation Status and Near Future

- **worch** is largely feature complete
  - Builds some (non-art) LBNE sub-projects
  - Used by ORKA for their ROOT/G4-based suite
  - ATLAS's use for building externals
- Building art has been challenging
  - easily taken as much effort as all other externals combined
  - art's native CMake build requires UPS to find dependencies
    - would prefer it to use the standard `-DYourExternalDir=/blah`
  - this means UPS must be built and made available by **worch**
    → initially broke part of **worch**'s model
  - had to learn many details of UPS (manual can be misleading)
  - dictates installation layout matches FNAL conventions

  → UPS problems solved, now applying to build art itself
- LArSoft's move from SVN/SRT to git/CMake in progress, will use the latter
- Probably want some novel end-user package management.
  - I will look at **hwaf** more once we get to this point