# Introduction to ReactorFsim

Matthew Worcester

*University of Chicago*

(Dated: March 9, 2005)

This introduction provides a brief description of ReactorFsim, the Braidwood Neutrino Experiment fast simulation package. It gives an introduction to the code and some simple steps and tips for getting started.

**Contents**

## I. INTRODUCTION

ReactorFsim is the fast simulation package for the Braidwood Neutrino Experiment. It contains an event generator (ReactorEvent) that can create several types of events, including reactor neutrinos, Californium 252 source(s), and muons. It also contains a spherical detector simulation (ReactorDetector) through which the particles generated in the event may be propagated. Each event can be run through multiple detector simulations in each job to test various detector parameters (*e.g.* detector radii or Gadolinium concentration) with identical events.

ReactorFsim produces both PAW [1] and ROOT [2] output files for physics analysis. The code requires the CERN program libraries (CERNLIB) [3] to be installed locally in order to create the PAW output and use the CERNLIB routines declared in ReactorFortran. The code also requires ROOT to be installed locally to produce the ROOT tree output.

Section II describes the steps required to run the ReactorFsim simulation. Section III briefly describes the most important files in the source code.

## II. GETTING STARTED

The first steps to use the ReactorFsim simulation are to obtain the source code and data files from the Braidwood CVS repository, customize the Makefile if necessary, and set the proper environment variables. All of these steps are required to build the ReactorFsim executable. Once the executable is built, it can be run with user inputs defined by an event setup file, as described below.

### A. Retrieving ReactorFsim from CVS

ReactorFsim is available from the Braidwood CVS [4] repository. The URL for the Braidwood CVS repository is: http://braidwood.uchicago.edu/private/software/cvs.html. The site is password protected. Passwords can be obtained from Ed Blucher [5] or Mike Shaevitz [6].

To retrieve the ReactorFsim package, the user must set the CVSROOT environment variable to the Braidwood CVS repository, log in to the repository, and check out the package. Approximately 14 MB of free disk space are required. The user must also have a

| Command |
|---|
| setenv CVSROOT :pserver:[username]@cp4.uchicago.edu:/theta13/cvs |
| → set the CVSROOT environment variable |
| → the user must supply the username |
| cvs login |
| → log in to the CVS repository |
| cvs checkout -r [version] ReactorFsim |
| → check out ReactorFsim |
| → the user must supply the version |

TABLE I: CVS commands to retrive the ReactorFsim package.

username and password to the repository. A username and password can be obtained from Matt Worcester [7]. The basic commands to check out the ReactorFsim package are given in Table I. More detailed instructions on CVS commands are available at the Braidwood CVS repository URL given above, including an example CVS session.

The ReactorFsim package will be checked out into a directory called ReactorFsim containing all of the files and subdirectories necessary to build and run the ReactorFsim executable. The ReactorFsim directory will be created by the check out process if it does not already exist. More information about ReactorFsim, including a list of available versions, is given on the online Braidwood code browser: http://braidwood.uchicago.edu/private/software/codebrowser.html, which is accessible by the same password as the Braidwood CVS site.

### B. Building the Executable

Once the ReactorFsim package is downloaded the user will need to build the executable. ReactorFsim has been tested on Red Hat Linux and Solaris UNIX systems, and some expert help is available for those platforms. See Table II for the systems and compilers tested to date. No development has been done with other architectures.

A Makefile is provided in the top level of the package. Users must first customize the Makefile to point to their local CERNLIB directories. The lines you must change are near

| Platform | Compiler |
|---|---|
| Red Hat Linux 7.3 | gcc 2.96 |
| Solaris | CC |

TABLE II: Platforms and compilers that have been used to develop ReactorFsim.

the top of the Makefile:

$$\mathrm{CERN\_ROOT} = /\mathrm{cernlib}/\mathrm{cern}/2001 \tag{1}$$

$$\mathrm{LIB} = /\mathrm{cernlib}/\mathrm{cern}/2001/\mathrm{lib} \tag{2}$$

The user must also specify their local ROOT directories via environment variables. The necessary ROOT environment variables are given in Table III. They are also summarized in the README file provided in the top level of the ReactorFsim package. Be careful not to overwrite any existing directories in your PATH or LD_LIBRARY_PATH environment variables when adding the ROOT directories. ROOT production versions 4.00.06 and more recent are known to work with ReactorFsim. Recent ROOT production versions can be obtained from the ROOT website [2].

One the CERNLIB and ROOT directories are specified, the command "make all" issued in the top level of the ReactorFsim package will compile the source code, link the object files, and build the executable. A successful build will produce an executable called ReactorFsim in the top level of the ReactorFsim package. Approximately 2 MB of free disk space are required to create the object files and build the executable. The command "make clean" issued in the top level of the ReactorFsim package will remove the executable and all object and source files created during the build.

| Variable | Value |
|---|---|
| ROOTSYS | ROOT installation directory |
| LD_LIBRARY_PATH | $(ROOTSYS)/lib |
| PATH | $(ROOTSYS)/bin |

TABLE III: Necessary ROOT environment variables.

| Input | Default | Definition |
|---|---|---|
| NEVENT | 100 | number of events to generate |
| OUTPUTFILE | test.ntuple | PAW output file |
| AROOTFILE | test.root | ROOT output file |
| RADII | 260 260 350 350 | three detector radii and the maximum event radius in cm: R0 R1 R2 Rmaxgen |
| DEPTH | 450 | detector depth in mwe |
| FASTNEUTRON | off | turn fast neutrons on/off |
| KIND | reactor | event kind, options are: reactor, cf252, pmtrad, muon, veto, all |
| PMTRAD | off | turn on/off radioactive decays in PMTs (must be "on" for KIND = pmtrad) |
| BPMTOUTPUTDEBUG | off | turn on/off PMTOutputArray debugging |
| MERGETIME | 2 | set PMTOutputArray's time resolution in ns |
| TRIGGER | off | turn on/off trigger simulation |

TABLE IV: User-defined inputs specified in the event setup file..

### C. Running the Executable

Most user defined inputs to the executable are given in a setup file in the ReactorFsim source directory: src/event_setup.txt. The inputs and their default values are listed in Table IV. These inputs may be changed over multiple runs without recompiling the executable. Prior to running the executable, the user should set the inputs as desired.

In the top level ReactorFsim directory the command to run the executable is "./ReactorFsim". To run the executable with the default parameters requires approximately 3 MB of free disk space for the PAW and ROOT output files. For large runs it is suggested that the user not keep the output files in their home directory. A complete file path may be specified in the event setup file, *e.g.* OUTPUTFILE = /fullpath/test.ntuple.

## III. IMPORTANT FILES

The short file descriptions in this section are intended to help users and developers understand the basic structure of the code and to guide searches for specific functionality. They are not comprehensive.

### A. ReactorFsim.cpp

ReactorFsim.cpp contains the main() code. It sets up the PAW and ROOT outputs. It defines each detector simulation by creating a member of the ReactorDetector class. Some detector parameters such as Gadolinium concentration that are not controlled by the event setup file may be changed from their default values by setting them in ReactorFsim.cpp. It contains the main event loop that generates each event by creating a member of the ReactorEvent class, runs the event through the detector simulations, and fills the PAW and ROOT outputs.

### B. ReactorEvent.cpp

ReactorEvent.cpp is the event generator. It generates different particles for the different types of events:

- reactor neutrino inverse-beta decay interactions producing one positron and one neutron per event; events are weighted by the neutrino flux
- $^{252}$Cf decays producing prompt gammas and multiple neutrons per event based on measured multiplicities
- muon energy loss with one muon per event, including production of one electron and one neutron per event via production and subsequent beta-decay of heavy isotopes $^{9}$Li and $^{8}$He; events are weighted by isotope production cross-section
- veto events in which particles produced in the separate Braidwood veto simulation are read in from a separate ascii file

The generated particle information is stored in energy/momentum and time/position arrays for later extraction.

### C. ReactorDetector.cpp

ReactorDetector.cpp simulates the detector. It creates the detector simulations in the constructor and propagates the generated events though the detectors in the LightsOut function. The LightsOut detector simulation consists of three functions:

- GenerateSecondaries reads the particle information arrays from ReactorEvent, uses those quantities to propagate the generated particles through the detector simulation, and refills those arrays with the new energy/momentum and time/position information.

- GenerateGammas models the energy deposited in the detector from the information in the post-secondary energy/momentum and time/position arrays and fills "gamma" arrays with the energy, time, position, and parent particle of the deposited energy.

- GenerateResponse simulates the response of the detector to the energy created in GenerateGammas from the "gamma" arrays, including a preliminary energy reconstruction using the generated position information of the energy deposits.

After modeling the detector response, LightsOut also contains several functions to do position (CalculateQPosition and ImproveQPosition) and energy (CalculateEnergy) reconstruction. This energy reconstruction uses the reconstructed position information.

---

[1] PAW, Physics Analysis Workstation, URL `http://wwwasd.web.cern.ch/wwwasd/paw`.

[2] R. Brun and F. Rademakers, the ROOT System Home Page, URL `http://root.cern.ch`.

[3] CERN Program Library, URL `http://wwwasd.web.cern.ch/wwwasd/cernlib/`.

[4] Concurrent Versions System, URL `https://www.cvshome.org`.

[5] blucher@hep.uchicago.edu

[6] shaevitz@nevis.columbia.edu

[7] mworcest@hep.uchicago.edu