WAGASCI SOFTWARE
INSTALLATION and USER GUIDE

Author: Pintaudi Giorgio

Physics Department, Yokohama National University
240-8501 Yokohama-shi Hodogaya-ku Tokiwadai 79-5
Minamino Laboratory
Email: giorgio-pintaudi-kx@ynu.jp
Phone: (+81) 070-4122-3907

October 20, 2018

Contents

Chapter 1

NEUT 5.4.0

NEUT is a neutrino interaction simulation program library that has been developed for the studies of the atmospheric neutrino and the accelerator neutrinos. In this guide it is described how to compile NEUT on a Linux machine. **NEUT 5.4.0** is the latest NEUT version at the time of writing (October 20, 2018). The author of this guide is not the NEUT author, but it is a member of the T2K collaboration. NEUT is not an open-source software and its sources **can be accessed only by T2K collaborators** in possession of a username and password for the T2K intranet.

A fresh Linux installation is assumed, so, before compiling NEUT, I am showing how to install or compile all NEUT dependencies. If you already have some of the dependencies installed, you can safely skip the relative paragraphs. Regarding the style of this guide, I wanted to stay on the safe side so I have explained every step in detail. This means that the guide could appear boring and over-repetitive to a reader well-versed in Linux software compilation and programming and I preventively apologize for that.

1.1 List of NEUT dependencies

To compile NEUT, three main dependencies are needed:

- **CERNLIB 2005**
  CERNLIB 2005 is a quite old version of `cernlib`, its latest release being dated 2012. Unsurprisingly it is quite tricky to compile. This is why I have included in the present guide a complete and detailed walk-through to the compilation of CERNLIB 2005. There are several patches that need to be applied to both CERNLIB and NEUT source codes. Some of them were written by me, some others by Hayato-san.

- **ROOT 5**
  NEUT is compatible **only with ROOT 5** and not ROOT 6 yet. There are plans to upgrade NEUT so to make it compatible with ROOT 6, but I don't know if there is any progress in that direction nor if there is any ETA. This plan was announced in the last T2K meeting in May. The latest and most updated version of ROOT 5 is ROOT 5.34.00-patches (it is based on ROOT 5.34.39 and the latest commit on the that GitHub branch is March 2018) and I have chosen that for my personal installation.

- **Fortran compilers** (`gfortran` and `g77`)
  The old fortran compiler `g77` is not maintained anymore and, for example, it is not present in the Ubuntu repositories since the Ubuntu 8.04 Hardy Heron (my first Ubuntu, what nostalgia). Anyway, it is still possible to install it in a somewhat nonstandard way as it will be shown in the following.

1.2 NEUT compatibility

NEUT officially supports only CentOS 7, but this doesn't mean that it cannot be run without issues on other Linux OSes. I will introduce here 3 OSes that I have successfully compiled NEUT onto: CentOS 7, Ubuntu 17.10 and Ubuntu 18.04. I will show how to compile NEUT only on Ubuntu 18.04 64 bit because that is the most delicate case and because Ubuntu 18.04 is the latest Ubuntu LST (Long Term Support) release.

1.2.1 CentOS 7 - TO DO

In May 2018 Hayato Yoshinari-san, who I think is also the main developer of NEUT, organized a small course about NEUT. Since the course was in Japanese and I am not fluent yet, I could understand very little. Anyway, in that occasion Hayato-san provided us with a CentOS 7 virtual machine containing all the needed dependencies to compile and run NEUT. The virtual machine can be downloaded from here. I am not sure that this link is of public dominion, but as far as it is circulated inside the T2K collaboration there shouldn't be any problem. The CentOS virtual machine can be run through Oracle VM VirtualBox. The CentOS virtual machine comes without any desktop manager. But it is possible to install it if needed (I have installed GNOME). The user-name is `neut` and the password is `neut5.4.0`. Here some relevant details about it:

OS	CentOS 7
kernel	3.10.0-693.21.1.el7.x86_64
gcc/g++/gfortran	4.8.5 20150623 (Red Hat 4.8.5-16)
ROOT	5.28.00h+
CERNLIB	2005 (dated 2016/12/12)
NEUT	5.4.0

1.2.2 Ubuntu 18.04

With the aim of writing the present guide, I wanted to test the NEUT compilation on a fresh install. I have chosen Ubuntu 18.04 because it is the latest Ubuntu LTS version, so in the following I will assume an Ubuntu 18.04 64 bit fresh install. Every other Linux distribution should be compatible after minimal modification to some of the shell commands, particularly regarding the package manager and the package names. Notice that I haven't used the default Ubuntu 18.04 compiler (version 7.3.0) but I have installed the old 4.8 version of `gcc`, `g++` and `gfortran` as shown in subsection 1.3.2.

OS	Ubuntu 18.04
kernel	4.15.0-23-generic
gcc/g++/gfortran	(Ubuntu 4.8.5-4ubuntu8) 4.8.5
ROOT	5.34.00-patches (patched version of ROOT 5.34.39)
CERNLIB	2005 (dated 2016/12/12)
NEUT	5.4.0

1.3 Compile NEUT and its dependencies

1.3.1 Preliminaries

1. Open a terminal or a shell. Ideally you should never close this shell from now until last step. If you need to close it nothing bad will happen but you might need to set up CERNLIB and ROOT environments again and/or come back to the right folder depending at which step you want to resume compilation.

2. First, we need to install some packages from the Ubuntu repositories. Some of them may be optional but I didn't have the patience to cherry-pick them. I have included them all just to be on the safe side. Do notice that it may take some time and space (∼500MB).

```
sudo apt-get update && sudo apt-get upgrade
sudo apt-get install build-essential git dpkg-dev cmake xutils-dev \
g++ gcc gfortran binutils libx11-dev libxpm-dev libxft-dev libxext-dev \
libssl-dev libpcre3-dev libglu1-mesa-dev libglew-dev \
libmysqlclient-dev libfftw3-dev libcfitsio-dev libgraphviz-dev \
libavahi-compat-libdnssd-dev libldap2-dev python-dev libxml2-dev \
libkrb5-dev libgsl-dev libqt4-dev libmotif-dev libmotif-common \
libblas-dev liblapack-dev csh tcsh gcc-4.8 g++-4.8 gfortran-4.8
```

1.3.2 Compilers

If you try to compile NEUT with compilers newer that 4.8, everything will still compile fine but you will get the following error on NEUT execution time:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
LOCB/LOCF: address 0x7f68e28cd740 exceeds the 32 bit address space
or is not in the data segments
This may result in program crash or incorrect results
```

```
Therefore we will stop here
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This error is ultimately connected to the `FFKEY` call in fortran. `FFKEY` allows something which is read in to be an integer or float at a later time in the run. This should not be allowed (and isn't) by modern standards, but is still in CERNLIB and NEUT.

To switch between compilers, create and make executable the following shell script

```
#!/bin/sh

if [ -z "$1" ]; then
   echo "usage: $0 version" 1>&2
   exit 1
fi

if [ ! -f "/usr/bin/gcc-$1" ] || [ ! -f "/usr/bin/g++-$1" ] || [ ! -f "/usr/bin/gfortran-$1" ];
    then
   echo "no such version gcc/g++/gfortran installed" 1>&2
   exit 1
fi

sudo update-alternatives --set gcc "/usr/bin/gcc-$1"
sudo update-alternatives --set g++ "/usr/bin/g++-$1"
sudo update-alternatives --set gfortran "/usr/bin/gfortran-$1"
```

The script accepts as the only and compulsory parameter the version of the compiler to switch into. It is `4.8` for 4.8 compilers and `7` for the 7.x compilers.

Be sure to run the script and switch to the 4.8 compiler before starting the compilation of ROOT, CERNLIB and NEUT.

1.3.3 ROOT 5

As already pointed out in the introduction, at the time of writing, **NEUT 5.4.0 is only compatible with ROOT 5**. Please refer to section 1.1 for further details about this "issue". Here I will explain how to compile ROOT 5.34.00-patches starting from the source code since it is the safest way to insure that everything is functioning correctly: if there is any incompatibility it is usually reported at compilation time, while, if you install ROOT through the pre-compiled binaries, any incompatibility will be uncovered only at run time, when the errors can be more obscure and misleading to interpret.

I recommend the use of the "three folders" procedure to install ROOT. One folder is for the source code, one for the compiling/building and one for the final installation. The sources folder is managed entirely using git and it contains a copy of the ROOT git repository. You can checkout the branch that you need to compile at any time. Another folder contains the temporary building files (such as the object files). It is effectively used only during compilation and can be removed afterwards. It is quite heavy (about 3GB) so I usually delete it as soon as I have installed ROOT. The last folder is the installation folder where a certain version of ROOT is installed. When running ROOT, in fact only this folder is needed.

I am showing now how to install ROOT. The following instructions are freely taken from the following pages:

- Building root
- ROOT v5-34-00-patch release-notes
- Build prerequisites

3. Open a terminal if not already opened.

4. Then install some fonts required by ROOT

   ```
   sudo apt install xfstt xfsprogs t1-xfree86-nonfree ttf-xfree86-nonfree \
   ttf-xfree86-nonfree-syriac xfonts-75dpi xfonts-100dpi
   ```

   You can optionally install the following packets to have a more thorough ROOT installation. They are by no means mandatory. I honestly have no idea what many of these packages do. I just hate dealing with missing dependencies when I run software.

```
sudo apt install libgif-dev libtiff-dev libjpeg-dev liblz4-dev \
liblzma-dev libgl2ps-dev libpostgresql-ocaml-dev libsqlite3-dev \
libpythia8-dev davix-dev srm-ifce-dev libtbb-dev python-numpy
```

5. Create two directories that will contain the ROOT source code and the compiled code. You can change the paths as you wish, but then you have to remember to change all the references and links in the following accordingly. You may also need to modify the patches since they include these paths.

   ```
   mkdir -p $HOME/Code/ROOT/{sources,5-34-00-patches,5-34-00-patches-build}
   cd $HOME/Code/ROOT
   ```

6. Clone the ROOT source code into the sources directory using git and then:

   ```
   git clone http://github.com/root-project/root.git sources
   cd sources
   git checkout -b v5-34-00-patches origin/v5-34-00-patches
   cd ../5-34-00-patches-build
   ```

7. Execute the cmake command on the shell.

   ```
   cmake -Dbuiltin_xrootd=ON -DCMAKE_INSTALL_PREFIX=$HOME/Code/ROOT/5-34-00-patches ../sources
   ```

   CMake will detect your development environment, perform a series of test and generate the files required for building ROOT.

8. After CMake has finished running, start the build from the build directory:

   ```
   cmake --build . --target install
   ```

   On UNIX systems (with make or ninja) you can speedup the build with

   ```
   cmake --build . --target install -- -jN
   ```

   where N is the number of available cores.

9. Now you can safely remove the build directory if you need that space.

   ```
   cd
   rm -Rf $HOME/Code/ROOT/5-34-00-patches-build
   ```

10. Every time that you run ROOT, you need to setup the environment. There are at least three ways to do that, more ore less automated.

    - One is to run the command

      ```
      source $HOME/Code/ROOT/5-34-00-patches/bin/thisroot.sh
      ```

      Every time it is needed.

    - Another way is to define an alias in the `.bash_aliases` file or in the `.bashrc` file. For instance I have set

      ```
      alias root5='source $HOME/Code/ROOT/5-34-00-patches/bin/thisroot.sh'
      alias root6='source $HOME/Code/ROOT/6-12-06/bin/thisroot.sh'
      ```

      To quickly switch between different ROOT versions.

    - Alternatively one can set up the correct root environment at boot by inserting the following lines in the `.profile` file

      ```
      # set the needed environment variables, PATH and LD_LIBRARY_PATH for root
      if [ -f "${HOME}/Code/ROOT/5-34-00-patches/bin/thisroot.sh" ] ; then
      ```

```
    source ${HOME}/Code/ROOT/5-34-00-patches/bin/thisroot.sh
    fi
```

Then every shell will have that version of root as default.

11. Before using root, remember to reboot (to reload the fonts). Then you can ROOT with the command `root`.

1.3.4 CERNLIB 2005

The following procedure was freely taken from the three reference web pages:

- NEUT installation
- CERNLIB installation
- g77 installation

12. Create a new folder for CERNLIB and move to it. In the following I am assuming that you won't leave that folder until the CERNLIB compilation is complete.

```
mkdir $HOME/Code/CERNLIB
cd $HOME/Code/CERNLIB
```

13. CERNLIB uses a different naming for make. I suppose this is due to small differences in naming conventions between UNIX and Linux (g-nu).

```
sudo ln -s /usr/bin/make /usr/bin/gmake
```

14. The following commands add the hardy repositories to the `sources.list` file and install `g77`. This step is needed to download and install the fortran `g77` compiler. It is advisable to comment or remove these lines after the `g77` installation.

```
sudo tee -a /etc/apt/sources.list << END

# Old hardy repository needed to install g77
# It is recommended to comment or remove following lines after g77 installation

deb [trusted=yes] http://old-releases.ubuntu.com/ubuntu/ hardy universe
deb-src [trusted=yes] http://old-releases.ubuntu.com/ubuntu/ hardy universe
deb [trusted=yes] http://old-releases.ubuntu.com/ubuntu/ hardy-updates universe
deb-src [trusted=yes] http://old-releases.ubuntu.com/ubuntu/ hardy-updates universe
END
sudo apt-get update && sudo apt-get install g77
```

15. Now it is time to solve a couple of bugs of the `g77` compiler. This is quite an old and now largely obsolete compiler. For this reason many adjustments are needed to get it working. You may need to modify the code below if your system libraries path or your gcc compiler version are different. For more information and troubleshooting please consider visiting this page.

    The following command will create a symbolic link of a library that `g77` would not able to find otherwise.

```
sudo ln -s /usr/lib/gcc/x86_64-linux-gnu/7/libgcc_s.so /usr/lib/x86_64-linux-gnu/
```

16. Then let us set up the `LIBRARY_PATH` environment variable. This must be done every time you want to use the `g77` compiler.

```
export LIBRARY_PATH="/usr/lib/x86_64-linux-gnu:\$LIBRARY_PATH"
```

If you want to avoid issuing the previous command every time, you can do the following:

```
     tee -a $HOME/.profile << END

     # set LIBRARY_PATH so it includes x86_64-linux-gnu if it exists
     if [ -d "/usr/lib/x86_64-linux-gnu" ] ; then
     LIBRARY_PATH="/usr/lib/x86_64-linux-gnu:\$LIBRARY_PATH"
     export LIBRARY_PATH
     fi
     END
```

17. Download all the CERNLIB 2005 source code archives:

```
wget http://www-zeuthen.desy.de/linear_collider/cernlib/new/cernlib-2005-all-new.tgz
```

```
wget http://www-zeuthen.desy.de/linear_collider/cernlib/new/cernlib.2005.corr.2014.04.17.tgz
```

```
wget http://www-zeuthen.desy.de/linear_collider/cernlib/new/cernlib.2005.install.2014.04.17.tgz
```

18. Extract the downloaded files:

```
     tar -zxvf cernlib-2005-all-new.tgz
     cp cernlib.2005.corr.2014.04.17.tgz cernlib.2005.corr.tgz
     tar -zxvf cernlib.2005.install.2014.04.17.tgz
```

19. Download my patched version of the old `patchy 4` source code and substitute it with the non patched one. I am not sure if the `patchy` suite is really necessary for NEUT but it is better to have and not need it than need it and not have.

```
rm patchy4.tar.gz
wget https://www.dropbox.com/s/n7n1wiivn1noets/patchy4.tar.gz
```

20. Download and check the patch:

```
     wget https://www.dropbox.com/s/loa53mklvotzsvj/jojo-CERNLIB.patch
     cp Install_cernlib Install_cernlib.backup
     patch -p0 --dry-run < jojo-CERNLIB.patch
```

21. If there isn't any error, patch the installation script

```
     patch -p0 < jojo-CERNLIB.patch
```

22. Now you can compile CERNLIB.

```
     ./Install_cernlib
```

23. Check that the compilation process went well by looking for errors in the log files in `2005/build/log`

```
     grep ./2005/build/log -R -i -e "error [0-9]" | grep -v ZDROP | grep -v ZBOOKN
```

24. Finally set up the CERNLIB environment for the next section.

```
     source cernlib_env
```

1.4 NEUT 5.4.0

25. If you are continuing from the same terminal as the previous section, you don't need this step. Otherwise open the terminal and issue the following commands to setup the NEUT compilation environment:

```
source $HOME/Code/ROOT/5-34-00-patches/bin/thisroot.sh
source $HOME/Code/CERNLIB/cernlib_env
```

26. Create the NEUT folder and move to it.

```
mkdir $HOME/Code/NEUT
cd $HOME/Code/NEUT
export NEUT_ROOT=$HOME/Code/NEUT
```

27. Download the NEUT source code from this link. To download the file it is necessary to insert one's own T2K credentials. Then move the downloaded archive in the NEUT folder. During the download don't close the terminal. Go back to the terminal and issue:

```
cd $HOME/Code/NEUT
mv neut_5.4.0_20180120.tar.gz neut_5.4.0_20180120.tar
tar -xvf neut_5.4.0_20180120.tar
mv neut_5.4.0/src ./src
rmdir neut_5.4.0
```

28. Download and check the patches

```
cd src/neutsmpl
wget https://www.dropbox.com/s/dxyjk5fzw0vuv85/jojo-NEUT-1.patch
wget https://www.dropbox.com/s/ew07kn9duzvhqtt/jojo-NEUT-2.patch
patch -p0 --dry-run < jojo-NEUT-1.patch
patch -p0 --dry-run < jojo-NEUT-2.patch
```

29. If there aren't errors, apply the patches:

```
patch -p0 < jojo-NEUT-1.patch
patch -p0 < jojo-NEUT-2.patch
```

30. Finally compile NEUT. All should be fine now.

```
./Makeneutsmpl.csh
```

Chapter 2

ROOT 6

ROOT is a modular scientific software framework. It provides all the functionalities needed to deal with big data processing, statistical analysis, visualization and storage. It is mainly written in C++ but integrated with other languages such as Python and R[1].

This chapter is very short because much of the content that should go here is already covered in section 1.3.3. The only real difference is that here we are compiling the latest version of ROOT with the latest compilers (at the time of writing: October 20, 2018).

I am assuming here that you have never installed ROOT before. That is why we are cloning the git repository. If you already have another installation of ROOT and you just want to upgrade or reinstall, please make the proper modification to these steps. I am assuming that you are capable of that.

This chapter is tailored towards Ubuntu 18.04 but it should be the same for any other Linux distribution with some minor modifications.

2.1 Preliminaries

First install all the packages listed in section 2. Again, some of them may not be strictly needed but cherry-picking which package is needed and which not could be very time consuming. If you have very strict space limitations and you are somewhat forced to install the least amount of software, I really feel sorry for you but don't expect me to willingly help you...

Then make you sure you are using the last version of the compilers by issuing

```
gcc --version
```

The output should be

```
gcc (Ubuntu 7.3.0-16ubuntu3) 7.3.0
[...]
```

2.2 ROOT Installation

Then follow the step 4 of section 1.3.3. Instead of step 5, type

```
mkdir -p $HOME/Code/ROOT/{sources,6-14-02,6-14-02-build}
cd $HOME/Code/ROOT
```

Instead of step 6, type

```
git clone http://github.com/root-project/root.git sources
cd sources
git checkout -b v6-14-02 v6-14-02
cd ../6-14-02-build
```

Instead of step 7, type

[1]Text taken from ROOT homepage

```
     cmake -Dbuiltin_xrootd=ON
-DCMAKE_INSTALL_PREFIX=$HOME/Code/ROOT/6-14-02 ../sources
```

The following steps are the very same of section 1.3.3 where in place of `5-34-00-patches` you just have to type `6-14-02`.

Chapter 3

DAQ software

The main software for data acquisition of the WAGASCI experiment is called Anpan (Acquisition Networked Program for Accelerated Neutrinos). It is based on the Calicoes software that was used for the Silicium Tungsten Electromagnetic Calorimeter (SiW-ECAL) prototype of the future International Linear Collider, developed by Frédéric Magniette and Miguel Rubio-Roy. Both Anpan and Calicoes are based on the Pyrame framework of which they two similar implementations.

Until Summer 2018, we were using the Calicoes software without almost any modification as the data acquisition software for the WAGASCI prototype (i.e. Proton Module + WAGASCI water-in detector + Ingrid module).

Between the end of the WAGASCI prototype commissioning run in Summer 2018 and the beginning of the WAGASCI Experiment Physics run in April 2019, I started my Ph.D. at YNU and I decided to further develop and improve the WAGASCI DAQ system. So I took the decision (quite one-sidedly I have to admit) to change the name of the acquisition software and release it under the GPLv3 license.

It is not clear which license is Calicoes released according to. Anyway Pyrame (which is written by the same developers) is released under GPLv3 so I assumed that Calicoes is released under the same license. This entitles me (Pintaudi Giorgio) to release again a modified version of the software under different name but giving due credit to the original authors (Frédéric Magniette and Miguel Rubio-Roy).

I have chosen that name because wagashi (after which the WAGASCI experiment is named) in Japanese means “traditional sweets” and anpan (a soft bun filled with red soy beans paste) is a kind of Japanese sweet snack (even if not properly an strictly traditional sweet). Anyway, I love anpan and I eat one of them almost everyday so we can say that the development of this software was fueled mainly by anpan. From that the name choice.

This chapter of the documentation isn’t a thorough guide of all the code that makes up anpan because much of that is already explained in the Pyrame documentation website and Calicoes documentation website. Anyways those websites are mainly aimed at developers who are well-versed in scientific programming and are not afraid of reading through much of the source code. Instead, this guide is meant to be a user manual for the physicist who just wants to use the software (and maybe only write some simple scripts).

One must always keep in mind that most of the times scientific software is not written by a big team of programmers with a steady job and a steady income, but by a few researchers or worse grad students with little (or no) income whatsoever, unsteady future prospects and a very limited budget. Because of the lack of money and man-power, usually the first thing that get cut or reduced is documentation (because it is not strictly necessary for the software to run). This, as long as the original developers are also the users of the software, has no tangible consequences. But as new physicists start to use the software an evil-loop onsets where software is used just as-is and works-just-because-it-works. Luckily the Calicoes and Pyrame software is adequately documented and with this guide I would like to close the gap between users and software even further.

3.1 Pyrame

I won’t go into detail describing the Pyrame framework since much of what I have to say is already covered in the Pyrame documentation website. So before continuing to the following I would suggest the reader to take a quick look there and get a rough idea of what is happening inside of Pyrame. This section is merely intended as an appendix and a review of the content shown there. The only real difference with the online documentation is that, instead of trying to explain how the software works, I will take the point of view of the user and I will only focus on how to use Pyrame to get some work done.

First, what the user need to understand is that Pyrame is a distributed software. This means that each part or function of Pyrame can be located on a different machine. Each of those parts are called "modules" and each module communicates with the others through the exchange of TCP/IP packets. If you don't know what the TCP/IP protocol is you can consult the Wikipedia pages: TCP and IP, that should be enough.

Basically each machine able to get an IP address can open a network port through which a certain application (or in the case of Pyrame a certain module) can communicate and interact with the modules "outside". For modules residing on the same machine the `localhost=127.0.0.1` IP address must be used. The purpose of the localhost IP address is simply to send an outgoing packet back in. So for example if we want to reach the Configuration Module (CMOD) that is running on the port `CMOD_PORT=9002` on the local machine we need the only to use the couple `localhost:CMOD_PORT` or explicitly `127.0.0.1:9002`. Or if we want to communicate with the Variables Module (VARMOD) that is running on the port `VARMOD_PORT=9001` on another machine in the same sub-net with IP `192.168.10.100` we would address it as `192.168.10.100:9001` and so on. The list of all the default ports is located in `/opt/pyrame/ports.txt`

There are 7 fundamental modules that must be always running during data acquisition for Pyrame to function correctly:

- The Variables module: `VARMOD_PORT=9001`
- The Configuration module: `CMOD_PORT=9002`
- The Mount Datadir Module: `MOUNTD_PORT=9005`
- The Acquisition-Chain Module: `ACQ_PORT=9010`
- The Run Control Module: `RC_PORT=9014`
- The Storage Module: `STORAGE_PORT=9020`

There may be other modules running depending on the specific configuration. The needed modules are usually started automatically when a configuration file is loaded so the user doesn't need to worry about that. To learn more about the configuration file go to section TO-DO.

Starting from Section 3.1.2 I will give a short description of each of these modules, but before that I would like to talk about the "State Machine".

3.1.1 State Machine

In Pyrame, the detector is always thought to be in one of four states. They are:

1. UNDEFINED: The detector state has not been probed yet. This is the state of the detector before Pyrame has started or after it has ended.
2. READY: The detector itself is in the same UNDEFINED state as before but the software has been initialized and the detector configuration has been loaded into memory.
3. CONFIGURED or RECONFIGURED: The configuration has been applied to the detector. This means that the firmware of the various boards has been configured, the power supplies turned on, the voltage set up and so on. The detector is ready to take data.
4. ACQUIRING The detector is acquiring data.

The principle at the base of the Pyrame workflow is that the user must go through all the steps from 1 to 4 in order to take data. The software will take charge of managing all the synchronization issues. All the transitions between machine state can be accomplished with the aid of shell scripts called from a terminal: `load_config_file.sh`, `initialize.sh`, `configure.sh`, etc.... It is also possible to use the CouchDB web-GUI to manage the transitions.

3.1.2 The Configuration Module

The configuration module (called also CMOD) is in charge of many things. First, it is dedicated to configuration storage and export. Every Pyrame module can register in the cmod and store its configuration parameters. The configuration parameters and the modules themselves are organized as a tree-like structure representing the actual structure of the detector. TO-DO how modules can edit the configuration?

Usually the user just load a configuration file in the cmod using the shell script `load_config_file.sh filename.xml`. The cmod contains at any time a copy of the configuration of all the modules and it can generate xml files containing all the parameters to be used with calxml.

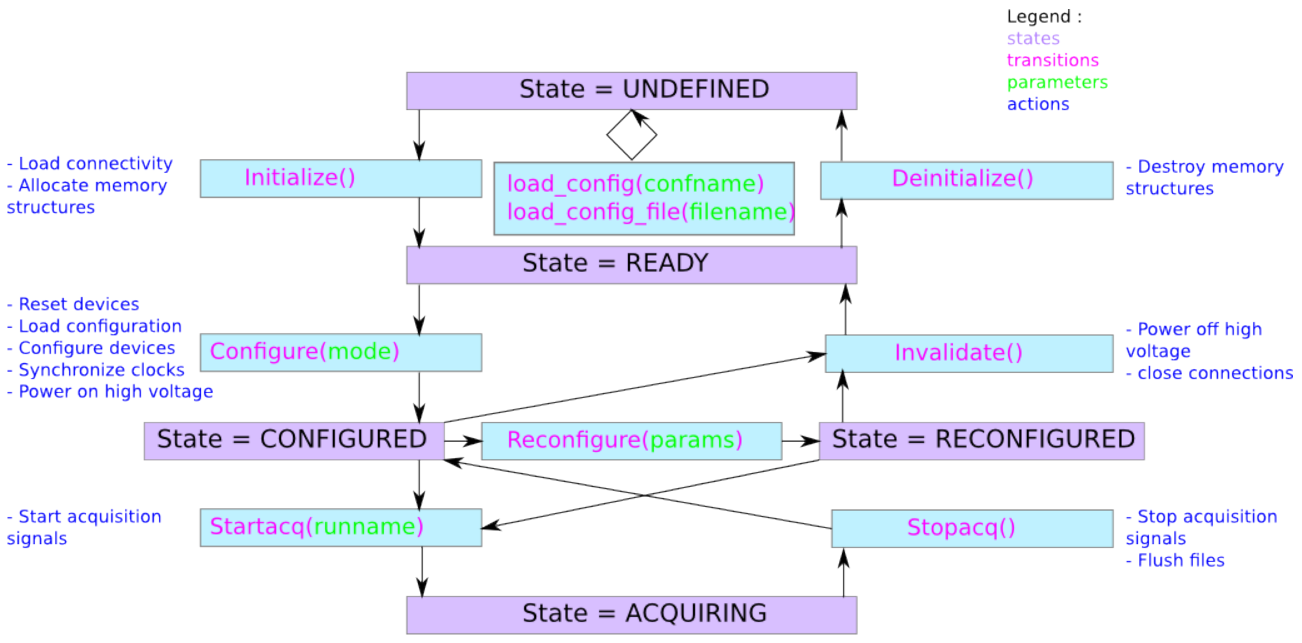

Figure 3.1: Pyrame State Machine

Another role that cmod plays is to manage the transitions between different states through the `transition_cmod` function. Again, usually this transition is managed by shell scripts or directly by the GUI so cmod is quite transparent to the user.

3.1.3 The Variables Module

The varmod is a centralized service that allows all the modules to share variables for collecting statistics or sharing global information. You can store a value associated with a name (a variable) and then get it back. You can also make some basic operations on the variables. For more information refer to the Pyrame documentation.

3.1.4 The Acquisition-Chain Module

In my opinion, understanding completely the Pyrame Acquisition-Chain is the most daunting task, if one really wants to know how the Pyrame software works internally. Since I am far from an expert in that field, I will only gather here what I have understood so far.

From the perspective of the user, reading the Pyrame documentation, the comments in the configuration file and the comments in the `cmd_acqpc.py` file should be enough to get things going in almost every real-world scenario.

For the developers and more experienced users let's start by taking a look at the graph 3.2: I will only focus on the WAGASCI case. All the configuration regarding the acquisition chain is done under the `pcacq_X` field in the configuration file. If you take a pre-existing configuration file written by myself you will find many descriptive comments in that section that hopefully will guide you through a painless configuration.

Media acquisition

The "media acquisition" phase is controlled by the acquisition plugins. What these plugins do is to "open" an acquisition media and get the data from there. The acquisition media can basically be of two types: the Ethernet port (with TCP or UDP protocols) or a raw file that was previously acquired (for testing or debugging purposes). For a more in-depth description of every plugin refer to this page and its sub-pages. After the data has been acquired, it is not ready to be analyzed yet because it contains the IP/TCP caps and it is in binary form.

You can take a look at the source code and related comments in the `anpan/cmd_acqpc/cmd_acqpc.py` file and in particular in the `init_fin_acqpc` function where every acquisition plugin is listed and explained.

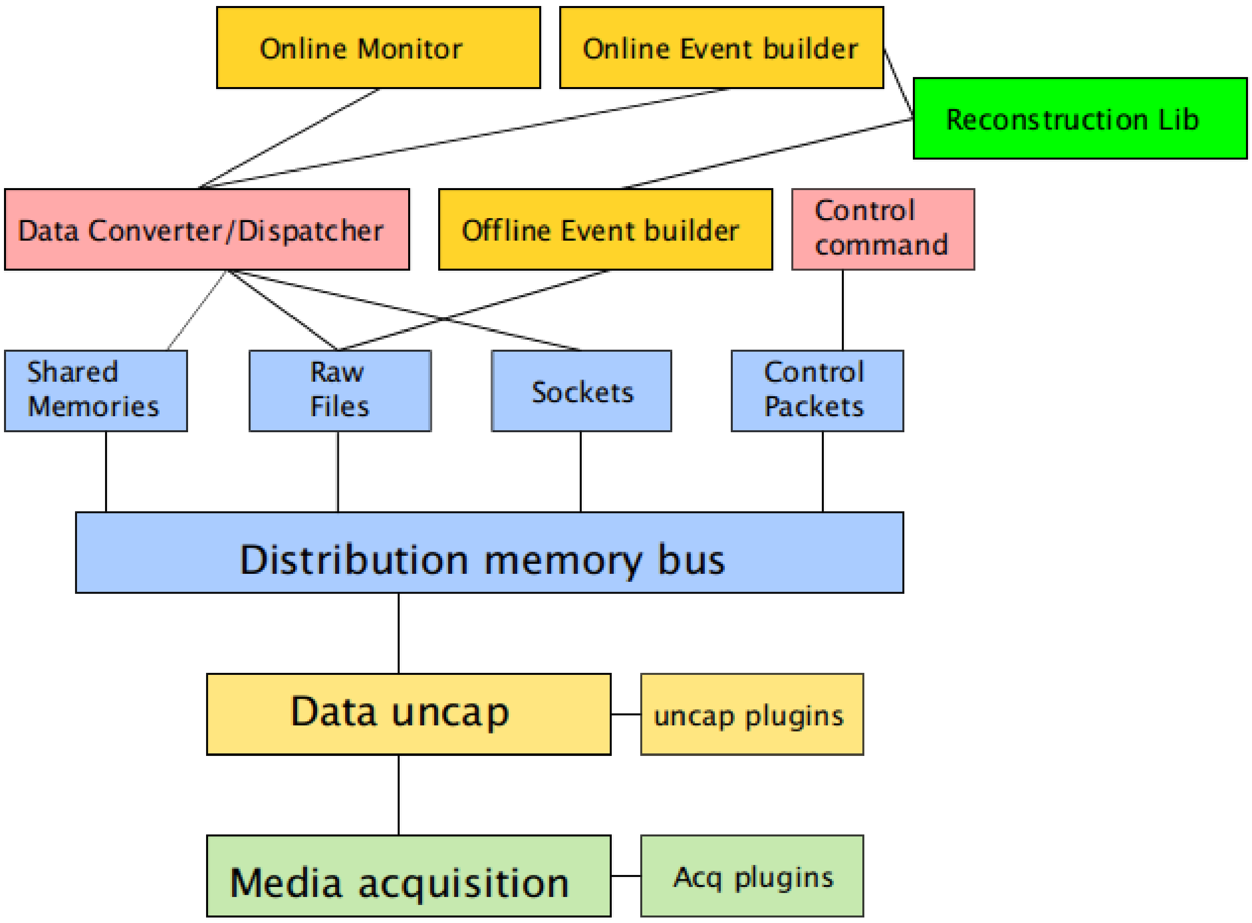

Figure 3.2: Pyrame State Machine

Data uncap

In this face, the acquired data is uncapped. This means that, for example in the TCP case, the IP/TCP headers are stripped, the data integrity checked (lost or corrupted packages) and the data content filtered (data packets or control packets). Honestly I don't know much more than this about these plugins.

Data Converter/Dispatcher

Here the decoder library is called. For example in the case of SPIROC2D the library path is `spiroc2d_decoder.so`. This library takes the uncapped raw data coming from the acquisition source (for example the GDCC) and convert it into a stream of blocks of events. These blocks are then fed to the Online Monitor and Online Event Builder programs. The decoder library needs to know the channel mapping (the geometry) of the detector to associate to every channel its position in space.

However, maybe because of some bugs or lack of time, the SPIROC2D converter for the WAGASCI prototype commissioning run was disabled. In the `cmd_dif.py` file the line needed to start the conversion was commented out and function's result was always set to positive, thus completely disabling Online Monitor capabilities of Pyrame.

I am going to enable it again and debug it (TO-DO)

Online Monitor

The WAGASCI online monitor code is contained in the `anpan/guis/om_wagasci` folder. It is a program external to Pyrame and WAGASCI that needs to be started by hand. It however interacts with Pyrame under the hood. It receives the Blocks and Events created by the "Data Converter/Dispatcher" face. More info about the More info can be found in section TO-DO.

3.1.5 The Run Control Module

TO-DO

3.1.6 The Storage Module

Storage is a Pyrame module that manages data created by other modules and its inclusion on the RunDB. For further info about the RunDB module refer to the Pyrame documentation.

Storage allows to use a mount point and a path to transparently store data elsewhere in the network. It also automates common tasks such as saving the calxml configuration and a log with optional varmod variables (e.g.: stats) at the same place as data files.

As far as I know there are only two kinds of storage that the Storage Module can handle. One is the standard file on a file system and another one is a NFS share on the network.

In the case of a standard file this is the format for the mp variable:

```
<param name="storage_brg_mp">file:///</param>
```

In the case of a NFS share this is the format for the mp variable:

```
<param name="storage_brg_mp">nfs://server/nfs</param>
```

3.1.7 Run and interact with Pyrame

There are several ways to interact with Pyrame and its modules. When the AnpanInstaller script installs Pyrame, it makes sure that Pyrame is automatically started at every reboot. On system with `systemctl`[1] Pyrame is started as a system daemon and its execution is controller with the `systemctl` or `service` shell commands. To enable Pyrame at startup use

```
sudo systemctl enable pyrame
```

To disable it

```
sudo systemctl disable pyrame
```

[1]Both Ubuntu 18.04 and CentOS 7 use `systemctl`

To start a single time Pyrame

```
sudo systemctl start pyrame
```

To stop it

```
sudo systemctl stop pyrame
```

Moreover there are several ways to interact with Pyrame modules. If a module is already running the simplest way to interact with it is to open a terminal and use the command:

```
chkpyr2.py hostname port function parameters
```

where `hostname` is the IP address of the module, for example `localhost`, `port` is the port number of the module, `function` is the particular function that you want the module to perform and `parameters` are the parameters to be passed to that function. All the public functions of the Pyrame and Anpan/Calicoes modules are listed on the online documentations.

You can also interact with Pyrame modules from external python scripts ... TO-DO

3.1.8 Pyrame Configuration

TO-DO

3.1.9 Blocks and Events

In Pyrame the raw data is organized in events and blocks of events. As far as I understood, events are grouped into blocks that are sent between Pyrame modules.

Event

On a raw level, an event is just a single hit TO-CHECK. Each hit is roughly described by the time, space and charge released. Other parameters that may refer to an event are described below.

An event is described by a struct like this:

```
struct event {
  char **time;          // List of string containing hit times
  int time_size;        // Number of hit times
  char **space;         // List of string containing hit points
  int space_size;       // Number of hit points
  char **data;          // List of string containing various
                        // event properties.
  int data_size;        // Number of elements in data
  char used;            // ??
  struct event *next;   // Pointer to the next event
  struct block *block;  // Pointer to the block
};
```

Up to now the `data` list contains the following elements:

- “spill”: spill number.
- “spill_flag”: the spill flag is set when the event happens inside of the beam spill time-frame. This means that the particle detected is most probably the result of a neutrino interaction from the neutrino beam inside or outside the detector.
- “roc”: ROC chip number.
- “rocchan”: ROC chip channel number.
- “bcid”: BCID (Bunch Crossing IDentification). Despite the misleading name, the BCID indicates just the number of rise time of the slow since the *startacq* signal. More on this on TO-DO.
- “sca”: Switched Capacitor Array number.
- “plane”: Plane number.

- "channel": Channel number.
- "en": Energy (charge). This field correspond to the ADC count.
- TDC time: "time"
- "hit": The hit flag is set if the event contains a hit. The hit may be due to a particle passage inside of the detector or simply by dark or electric noise.
- "gain": The gain flag is set when we want to measure the gain. In that case we compare the pedestal and first hit position to calculate the gain.

Block

A block is described by a struct like this:

```
struct block {
  int id;                // Block identifier
  char finalized;        // ??
  int nb_events;         // Number of events in the block
  char **props;          // List of strings that can contain
                         // various block properties as the
                         // spill number, spill flag, etc...
  int props_size;        // Number of props strings
  char *bstr;            // ??
  struct evtstruct *es;  // This struct is used to navigate
                         // through the various props strings
  struct event *events;  // List of events
  struct block *next;    // Pointer to the next block
};
```

Up to now the props list contains the following elements:

- spill number: "spill"
- spill flag: "spill_flag"

The "spill_flag" is "1" during the spill time frame and "0" otherwise. This flag is useful to discriminate between cosmic muons and particles coming from accelerated neutrino interactions. For further info about the WAGASCI spill timings refer to section TO-DO.

3.2 Anpan

- WAGASCI_PORT=12000
- ACQPC_PORT=12001
- GDCC_PORT=12003
- CCC_PORT=12004
- DIF_PORT=12005
- ASU_PORT=12006