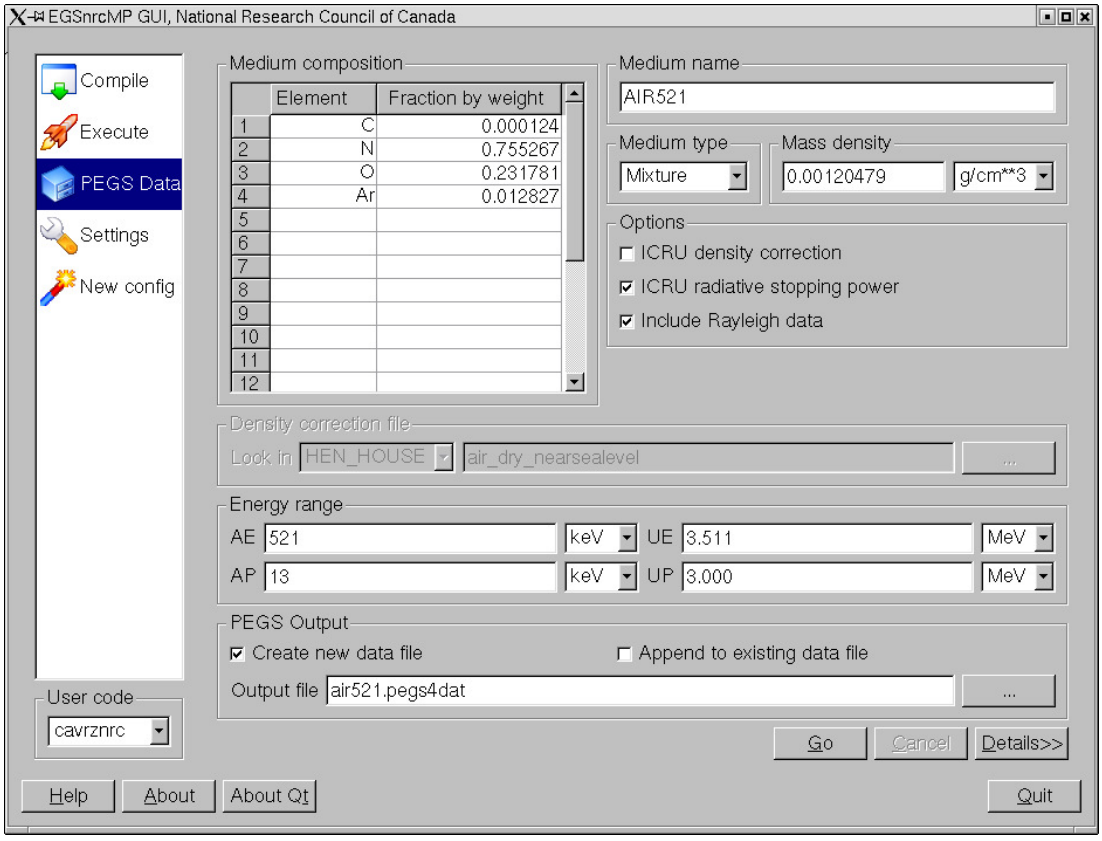
EGSnrcMP: the multi-platform environment for EGSnrc

I. Kawrakow, E. Mainegra-Hing and D. W. O. Rogers
Ionizing Radiation Standards
National Research Council of Canada
Ottawa, K1A OR6

September 2006

NRCC Report PIRS-877



Available on-line via:
http://irs.inms.nrc.ca/software/egsnrc/documentation.html

EGSnrc Code System Licence

Abstract

This manual describes the NRC EGSnrcMP multi-platform environment for running the EGSnrc Monte Carlo code system for simulating the transport of photons and electrons. In previous distributions of EGSnrc, the system was run in the csh scripts and structure that had been developed to work in a Unix environment with EGS4. The EGSnrc code itself and the scripts in particular meant the system would not run on Windows platforms, and the scripts had become very hard to maintain since they had grown to handle many more situations than originally designed for. The EGSnrcMP environment is a ground-up reworking of the way EGSnrc is maintained and run. The code itself has been reworked slightly to allow the same code to work on all operating systems (with some very minor exceptions which are handled by alternate coding of some macros), and the scripts reworked to be far more flexible and powerful. In addition, graphical user interfaces to run the codes and installation have been developed based on the Qt GUI toolkit library.

Contents

List of Figures

Nomenclature

The nomenclature with the EGSnrcMP environment is somewhat confusing to start, and hence a brief set of definitions is provided. The $ identifies an environment variable and the forward slashes, /, are separators on Unix/Linux. Although on Windows the notation is different (%var% and \), for simplicity throughout the text we will use the first notation.

`$HEN_HOUSE` is the directory on which the EGSnrc system is maintained. It can be anywhere on the file system except that no spaces are permitted in the path (and so, Windows users can not install the system on *e.g.* `C:\Documents and Settings\user\HEN_HOUSE`). On Unix/Linux systems the `HEN_HOUSE` variable is set for the user in the `egsnrc_cshrc_additions` or `egsnrc_bashrc_additions` files. On Windows systems it must be defined in the user's environment (this can be done by the installation wizard).

`$EGS_HOME` is the main directory for an individual user's user-codes. The environment variable `EGS_HOME` must be defined to point to this directory. On Windows systems this can be automatically done by the installation program. On Unix and Linux systems, users must define this variable in their shell initialization file (typically `.cshrc` for C-shell and derivatives or `.bashrc` or `.profile` for Bourne shell and derivatives).

`$EGS_CONFIG` is the absolute path to the file describing the config which is to be used. It must be defined by the environment. On Windows systems it can be automatically defined by the installation program, On Linux and Unix system users must define it in their shell initialization file.

Config is a set of variable definitions for various EGSnrc directories, compiler name and compilation options, etc. These definitions are stored in a config file pointed to by the environment variable `EGS_CONFIG`, which is created during the installation/configuration process. The config file is included by all make files when building an EGSnrcMP user-code and is also used by some of the Unix shell scripts.

`$my_machine` is the name used to refer to a config. It is used to distinguish different Fortran files and executables related to different configs and also the different `bin` and `lib` areas which contain files specific to a given config on both `$HEN_HOUSE` and `$EGS_HOME`. This variable is defined in the `$EGS_CONFIG` file and selected by the user during installation/configuration. This variable, *i.e.*, `$my_machine`, is also called `CONFIG_NAME` at various places in the documentation.

`egs_gui` is a GUI for compiling and executing a user-code, for running the PEGS4 code and for adjusting settings and creating a new config. For Linux and Unix users the alias `egsgui` is defined, which executes `egs_gui` in the background and discards all diagnostic messages printed to standard output.

`egs_inprz` is a GUI for working with input files for the RZ user-codes, compiling and executing an RZ user-code and for adjusting settings and creating a new config. For Linux and Unix users the alias `egsinprz` is defined, which executes `egs_inprz` in the background and discards all diagnostic messages printed to standard output.

`egs_install` is a GUI for installing/configuring the EGSnrcMP system. A reduced version of this GUI is used by `egs_gui` and `egs_inprz` for adjusting settings and creating a new config.

`$PATH` is the environment variable which tells the system where to find executable codes and it must include the `$my_machine` subdirectory of `$HEN_HOUSE/bin` and `$EGS_HOME/bin`.

1 Introduction

1.1 Intent of this report

This report describes a major revision of the EGSnrc system's structure, but not any changes in the physics of the system. The underlying rationale was to develop a system that could work on the Microsoft and Mac OSX platforms as well as Unix/Linux platforms while minimizing the differences in the source code required. At the same time we needed to rework the Unix scripts to be more flexible. The previous system was based on assumptions about compilers that were native to most systems, whereas today there are a variety of compilers available for many systems. To accomplish these goals required some minor changes in the coding of the EGSnrc system itself and a minor modification of the initialization calls required in user-codes. These changes are documented here, as are the new GUIs and installation scripts.

1.2 Overview of the changes

There have been several improvements in the EGSnrc system, which make the installation, maintenance and use of the system much simpler and, at the same time, portable to other operating systems:

- EGSnrc data file units and user-defined I/O units are opened explicitly through a call to subroutine `egs_init`, which must be step 0 in every EGSnrc user-code. Symbolic links are not needed anymore and the previously used `.environment` file is replaced by the `.io` file.
- The EGSnrc system and the user-codes are now built using `make`, a standard tool on Linux and Unix systems. Windows users can use the GNU version of make.
- Config files (`*.conf`) are used to describe a combination of operating system and Fortran compiler. This implies that one can use the same EGSnrc installation from different operating systems and/or use different compilers by just switching the current config.
- Command-line arguments can be passed to a user-code to specify things like the input file, the `.pegs4dat` file, the name for output files, the `$HEN_HOUSE` and `$EGS_HOME` directories, flags for a batch run, *etc.*. One can list all available options by using the -h or –help command line option.
- Utilities are available for managing I/O units
- For Unix/Linux systems, a more efficient and general parallel job submission and processing approach has been implemented.
- An interface for accessing EGSnrc data structures and calling EGSnrc subroutines from a C program is now available, *i.e.*, user-codes can be written in C or C++.

- Besides the `gz, bz2` and `tar` archives, new archives are created using the zlib library for compressing files using an in-house format (`*.egszip` files). This way, archives can be uncompressed by the EGSnrcMP installation program on any operating system without relying on external uncompressing utilities that might not be available.

- Three graphical user interfaces are available for working with the EGSnrcMP environment: `egs_install`, `egs_inprz` and `egs_gui`. The first one is a graphical installation utility that can be used for creating or updating the EGSnrcMP environment and further configs (operating system-compiler combinations). The second one is for working with the NRC RZ user-codes suite (see Reports PIRS-702[1] and PIRS-801[2]). The third one can be used to work (compile, run) with any EGSnrc user-code and produce PEGS4 data sets. The two latter GUI's can also invoke a reduced version of `egs_install`, called `egs_configure` for creating new configs.

- The spin data base is now in binary format, *i.e.*, it is 5 times smaller than the text version and can be read much faster. ICRU density effect correction files for calculating electron stopping powers for elements and compounds are distributed in single files and split during installation.

- On Windows, GNU `make` is included in the installation in case it is not available on the user's system. If no Fortran compiler is available, users are asked by the installation GUI whether it should install the GNU C, C++ and Fortran compilers from the EGSnrc distribution site.

- All NRC user-codes distributed with the system have been cleaned up to use system independent wrapper functions and subroutines for date and time, hostname, etc.

- The main system and the NRC user-codes distributed with the system have been adapted so that no global static compilation switch is required.

1.3 Brief description and comparison to the previous system

In the previous system, the user used a series of scripts to perform various tasks. The scripts `egs_compile` and `egs_run` were used to compile and run user-codes. At the compilation stage, the `user_code.configuration` file was used to specify all the different source code modules to be used to create the user_code, *e.g.*, `dosrznrc.mortran, srcrz.mortran, ranlux.macros, ranlux.mortran` *etc.*. At run time, the file `user_code.environment` was used to associate various Fortran I/O units to explicit file names.

While the previous system was flexible in many ways, there were several serious drawbacks. Firstly, the `egs_compile` and `egs_run` scripts hardwired various assumptions about compilers and operating systems which meant, *e.g.*, that if you were using the GNU g77 FORTRAN compiler on a Solaris system, the scripts didn't work and had to be changed. These scripts didn't work at all under Microsoft Windows and the method used to associate I/O units with specific files was not portable to the Windows environment.

To avoid the above problems, the EGSnrcMP environment has been developed. The increased flexibility has come at the cost of increased complexity in the implementation, but

the vast majority of this is hidden from the user and the new system should be easy to use once a few new concepts are understood.

An important concept is that of the `$EGS_CONFIG` environment variable. This variable points to a file with all the relevant information about the environment being used such as the operating system being run under, the compiler to be used and its switches. The user is able to set up several environments within one installation (*e.g.*, using different compilers or different hardware or different operating systems) and then alternate between them by resetting the `$EGS_CONFIG` variable. The `$HEN_HOUSE/bin` and `$HEN_HOUSE/lib` areas (see figure 4, page 30) then have separate subdirectories corresponding to each `$EGS_CONFIG` environment. This is discussed in more detail in the next section.

To overcome the problems of defining I/O units, two new subroutines have been added to the EGSnrc system, `egs_init` and `egs_finish` which handle the I/O units. This involves two new features. One is the ability of the user-codes to read command line inputs (*e.g.*, `-p tutor_data` specifies that the `pegs4` dataset `tutor_data.pegs4dat` is to be used) and also the use of the file `user_code.io` which allows arbitrary Fortran I/O units to be associated with arbitrary files (analogous to the old `.environment` files). The new routine means that the EGSnrc system now explicitly opens and closes files using their specific names, thereby making the code portable to the Windows system. The use of these new capabilities are described below in section 10, page 33.

The final major component on the new environment is the use of `make` files in the compilation process rather than scripts. There is a version of GNU `make` available for the Windows OS so that this makes the EGSnrcMP environment portable across different operating systems. In a similar vein, the reworking of the I/O routines means that one can run most user-codes directly without using an `egs_run` script.

2 How to install EGSnrc using the EGSnrcMP environment

2.1 Unix/Linux platforms

2.1.1 Libraries or compilers to be separately obtained

It is essential to have a working Fortran compiler for installing and using the EGSnrc system. If no Fortran compiler is available, the installation process aborts. The GNU suite of compilers is available for free and can be downloaded at `http://www.gnu.org`

A working `make` utility is needed for building the different system components and the user-codes. Make utilities from different Unix flavours tend to differ. The main system has been tested and found to work with `make` on AIX, OSF1, IRIX, SunOS, HP-UX and Linux. The Makefiles for the C interface will only work with the GNU version of `make`.

The new EGSnrc system uses the Qt library version 3.1 or higher for the graphical user interfaces. Most Linux distributions include the Qt library these days, since the popular Desktop Environment KDE is based on this library. However, if the Qt library is not available in the user's system, one can download it for free at `http://www.trolltech.com`. For more information about installing Qt, see section 2.4, page 15.

To take advantage of the new EGSnrc C interface, a working C or C++ compiler is required. Again, the GNU suite of compilers can be installed.

2.1.2 Traditional Linux/Unix Installation

The traditional method for installing EGSnrc is to run an installation script after obtaining several tar files, either from the web site (`http://irs.inms.nrc.ca/software/egsnrc/download.html`) or from a CD (only handed out to those taking a course).

This is still a viable method. The files that must be downloaded are:

```
V4_EGSnrc.tar[ |.gz|.bz2]
V4_EGSgui.tar[ |.gz|.bz2]
V4_manuals.tar[ |.gz|.bz2]
V4_user_codes.tar[ |.gz|.bz2]
V4_spinms.tar[ |.gz|.bz2]
```

where `[ |.gz|.bz2]` indicates that you need one of the file formats, the shortest one for which you have the code installed for uncompressing the format (*i.e.*, `gunzip` for `.gz` and `bunzip2` for `.bz2`). If you have none of them, just get the uncompressed tar files.

The only absolutely essential file is `V4_EGSnrc.tar`. The file `V4_spinms.tar` has changed format from previous versions (binary file is much shorter and is read much faster but the physics is unchanged), so it needs to be downloaded even if you have the previous version. The other three tar files are also highly recommended.

All of these files need to be placed on a single subdirectory since the install script assumes

this. But the location of the EGSnrc system is independent of where the tar files are kept.

In addition to the above files, one needs the script `install_egs`. It is easiest if it is copied to the same area as the above files, but, in principle, it can be stored anywhere that the user has write permission (and in particular it must be copied from the course CD).

Prior to starting the installation, one must ensure that the environment variable `QTDIR` is defined to point to the Qt library installation directory if you want to install the `GUIs` (see section 2.1.1).

The traditional installation process is followed by issuing the command `./install_egs` (after ensuring that the file is executable). Note that one MUST NOT redirect the output from the install script since this script is interactive, unlike in past releases. On the other hand, the script does create several output files. The files `install.status` and `install.log` are both on the directory from which `install_egs` is run and they inform the user what has happened in `install_egs`.

By far the most complex part of the installation is handled by the script `configure` which is called from within `install_egs`. The `configure` script establishes the parameters for a given config file related to the compiler used and the options chosen (see section 9.1). This script leaves the file `config.log` on `$HEN_HOUSE/scripts` (which is where the script `configure` resides). Note that the user can run this script separately to create another config file and then alternate between configs by changing the environment variable `EGS_CONFIG`. This allows, *e.g.*, easy switching between compilers.

Another component of the installation is handled by another script, which may be called from within `install_egs`, *viz.*, `finalize_egs_foruser`. This script leaves behind a log file called `finalize_$USER.log` on `$HEN_HOUSE/scripts` where this script is also found. The purpose of this script, which can be run independently, is to set up an individual user's area. For a single user, this area is usually set up on the same account as the `HEN_HOUSE`. However, for an installation with many users, the `HEN_HOUSE` may be on one account and each user's area on their independent accounts, in which case `finalize_egs_foruser` needs to be run for each user's account (but must be run by someone with write permission on the users account and the `$HEN_HOUSE`)..

The final component of the installation that may be called from within the `install_egs` script is the generation of the GUIs. This can also be done independently by going to the GUI area `$HEN_HOUSE/gui` and executing a `make` command after the environment variable `$EGS_CONFIG` has been defined.

The installation scripts require many inputs, usually with good defaults suggested. In most cases, if the response is unclear, typing `help` will generate a message with further information.

Once the `install_egs` script has been run, there are three statements that must be added to your `.cshrc`, `.bashrc` or `.profile` file (depending on which shell you run in interactively). These are near the end of the output from `install_egs` and involve `setenv EGS_HOME` (or `export EGS_HOME` for `.bashrc`), `setenv EGS_CONFIG` and
`source (appropriate_directory)_egsnrc_cshrc_additions` or
`source (appropriate_directory)_egsnrc_bashrc_additions`.
Once these additions have been made to your `.cshrc` or `.bashrc` file, you must source the

file to bring these definitions into effect. Note that you may want to redefine `EGS_CONFIG` if you want to use an alternative compiler or another configuration of some sort.

The `egsnrc_cshrc(bashrc)_additions` file also sources the file `$HOME/.egsnrc_cshrc_additions` or `$HOME/.egsnrc_bashrc_additions`. This file allows the user to tailor their personal environment although this could be done directly in your `.cshrc` or equivalent file directly.

The final component of the installation is setting up the information needed for doing runs in batch mode on the user's system. This is discussed below in section 6, page 25.

2.1.3 Other installation alternatives for Linux

2.1.3.i `egs_install` GUI

An alternative way of installing the system on a Linux system is to use the graphical user interface called `egs_install` (vs the script `install_egs` described above). This GUI uses the `Qt` library and the binary on the distribution site (size $\sim$ 9Mb) has been created with the GNU C++ compiler version 2.95.3 statically linked to the Qt 3.2.1 library on a SuSE 7.3 Linux system. We have successfully tested this binary of the installation GUI on the following Linux systems: SuSE 8.2, Red Hat 9.0 and Mandrake 9.1. There is no guarantee that this binary will run properly on older systems (and it is guaranteed that it will not run on systems that are so old that they use the `a.out` format for executables.).

This GUI works on the same set of `tar` files as described above in section 2.1.2 except that it uses the `*.egszip` format of all the files (this format is like a `.gz` file, but the GUI has built-in routines for reading it). The GUI looks for these files on the local area and if they are not found, on the user's request the GUI will download them from the EGSnrc distribution site. The user can select the specific system components that should be downloaded/uncompressed. This allows the flexibility of partial future updates.

The installation GUI is based on the Qt system but is statically linked so the user does not need to install the Qt libraries to use the install GUI. However, as part of the installation process, the user is asked to make the GUIs for running the codes `egs_gui` and `egs_inprz` and to do this, one needs the Qt libraries installed (see section 2.4).

2.1.3.ii `egs_install_self` GUI

This GUI is identical to the previous one except that the file includes all the necessary files, *i.e.*, it is a self-extracting binary. As a consequence the file is much larger ($\sim$ 23 Mb).

2.1.4 The `test_egsnrcmp_distribution` script

The script `test_egsnrcmp_distribution` is found on `$HEN_HOUSE/scripts`. It exercises the distribution and can be used to create a file with the output from various test runs by issuing:

```
test_egsnrcmp_distribution >& test_distribution_machine.out &
```

These can be compared to the output from the same script run on various machines and found on the area `$HEN_HOUSE/scripts/test_distribution_outputs`.

2.2 Windows platforms

2.2.1 Libraries or compilers to be separately obtained

It is essential to have a working Fortran compiler for installing and using the EGSnrc system. On the EGSnrc distribution site, the archive `V4_MinGW.egszip` contains a Windows version of the GNU suite of compilers (C, C++ and Fortran) and other related tools. For more information on the GNU suite of compilers for Windows the user is referred to `http://www.mingw.org`

During installation, if no Fortran compiler is found, the user is asked whether the GNU compiler should be installed by the setup program. If no Fortran compiler is available, the installation process must be cancelled by the user since it will not be possible to build the EGSnrc system and the user-codes.

Another key element in the new EGSnrc system is the GNU `make` utility. Since it might not be available on a Windows system, we have also included a free distribution of GNU `make`, which is copied to the system's binary directory.

The new EGSnrc system uses the Qt library for the graphical user interfaces. With the Windows installation, version 3.2 of this library is included and copied to the EGSnrc binary directory of the current config. Since the GUI's are distributed in executable form, there is no need for compiling them at installation time. All the sources for the GUI's are distributed under the GPL with EGSnrc and can therefore be modified to meet a user's needs. However, the user must possess a licensed copy of the Qt library for Windows and a C++ compiler (MS, Borland and recently GNU) in order to be able to re-compile.

2.2.2 Downloading the necessary files

To install the EGSnrc system, the different components (system files, user-codes, spin data base, documentation, *etc.*) and the installation GUI (`egs_install.exe`) can be directly downloaded from the EGSnrc distribution site. The distribution files have the `*.egszip` extension and are created using the zlib data-compression library (see `http://www.gzip.org/zlib/`). They can be placed together anywhere as the installation program will prompt for their location in case they are not on the same location (default).

There are two alternatives to the installation method outlined above: the self-extracting executable file `egs_install_self.exe` with all system components included (size $\sim$ 30 Mb) or the smaller executable file `egs_install.exe` (size $\sim$ 4Mb). The latter approach will have the installation program check on its current location for system components (`*.egszip` files) and if none is found, it will ask whether it should download them from the EGSnrc distribution site (`http://irs.inms.nrc.ca/software/egsnrc/download.html`) or else it will inquire for the location of the distribution files.

If no compiler is found, the user is prompted whether the installation GUI should install the GNU Fortran, C and C++ compilers and the GNU `make` utility on `$HEN_HOUSE\gnu`. Once the GNU compilers are installed, the PATH environment variable for the current user is updated with the binary directory for the GNU tools `$HEN_HOUSE\gnu\bin`.

2.2.3 Installation

The installation wizard guides the user through the whole process requiring information related to files location, compilers to use, what kind of installation the user wants, *etc.* Firstly, on the environment page, the installation utility asks the user for the location of the EGSnrc system (environment variable `HEN_HOUSE`) and the user area (environment variable `EGS_HOME`, where the user keeps his/her user codes). On the next screen, the system configuration page, the user is required to enter among other things, a name for the configuration file if there was no previous configuration or else, a combo box displays the current configuration file together with a list of available configurations. The environment variable `EGS_CONFIG` points to the current configuration file, which is the control centre of the system and contains many specifications of the particular compiler-operating system combination.

These environment variables need to be set prior to using the EGSnrc system. This can be done by the user after finishing the installation or the installation program can take care of it if the user selects this option on the GUI (`set HEN_HOUSE`, `set EGS_HOME` and `set EGS_CONFIG` check boxes).

Once all the information has been entered, one can start the installation process, which first uncompresses the files into the selected system location (`HEN_HOUSE`) and then performs a series of tests for the selected Fortran and C compilers. The results of these tests are used to create the system/compiler dependent files `machine.mortran` and `machine.macros`. The installation utility then starts the actual building of the EGSnrc system, by creating the Mortran3 string processor `mortran3.exe` and the PEGS4 data pre-processing tool `pegs4.exe`. After the EGSnrc system is installed, if the user requested it, a user area is created with all the NRC user codes and a user pegs4 data directory. Finally, user codes previously selected by the user on the wizard, are compiled and the path to the location of the executable files on the system and user areas is added to the environment PATH variable, if it was not already appended.

2.3 Installation on Mac OSX

Most of the installation instructions about Linux/Unix systems (see section 2.1) apply to Mac OSX. Both, the script and graphical installation wizard, worked on our test systems (fairly old hardware running Mac OSX 10.2 and Mac OSX 10.3). There are few restrictions and differences:

- The self-extracting installation method does not work on Mac OSX.
- If you want to use the command line interface and your default shell is `bash`, you have to put the environment variable initializations in the file `.profile` in your home directory instead of `.bashrc`.
- When starting the GUIs from a shell, you have to use `open egs_gui` or `open egs_inprz`. Because of this difference, some of the aliases defined in the `egsnrc_*_additions` files will not work for Mac OSX.

- If you want to be able to start the GUIs directly (*i.e.* not from a shell), you have to define the `EGS_CONFIG`, `EGS_HOME` and `HEN_HOUSE` environment variables in a file named `environment.plist` in the directory `$HOME/.MacOS`. This is a XML file that may or may not already exist. If the file does not exist, create it and put the following in it:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
       <key>EGS_CONFIG</key>
       <string>the_absolute_path_to_your_config_file</string>
       <key>EGS_HOME</key>
       <string>the_absolute_path_to_your_EGS_HOME</string>
       <key>HEN_HOUSE</key>
       <string>the_absolute_path_to_HEN_HOUSE</string>
</dict>
</plist>
```

  Note that the directory definitions (`EGS_HOME` and `HEN_HOUSE`) MUST end with a slash. If the file already exists, just insert somewhere the 3 definitions.

- It appears that Mac OSX executables are not backward compatible (the installation wizard compiled on the 10.3 system did not run on the 10.2 system). On the web we are providing the installation wizard compiled on 10.2. It runs on 10.2 and 10.3 but there is no guarantee that it will work on earlier Mac OSX versions.

Please note that we typically do not have access to a Mac OSX system and therefore can not provide support for this platform.

2.4 A word about the Qt library

The GUIs were developed using Qt, a multi-platform, C++ Graphical User Interfaces toolkit that enables building efficient, portable and maintainable GUI applications. Qt is a fully object-oriented, easily extensible C++ application framework that enables rapid building of state-of-the-art GUI applications. For more information please see http://www.trolltech.com.

We started developing the GUIs with Qt version 3.0. Considering the new developments and improvements in the Qt library since then, we strongly recommend that the user installs the latest available version of this library to avoid possible conflicts with older versions.

On Linux we have built and run the EGSnrc GUIs successfully for version 3.1.1 and up for SuSE 7.3 and 8.2, Red Hat 9.1 and Mandrake 9.0. Since KDE, the most widespread `Graphical Desktop Environment` for **Linux**, uses the Qt C++ cross platform GUI toolkit for KDE development, the Qt library is shipped in most Linux distributions. When installing

a different Qt library to the one that comes with the Linux distribution on the user's machine, one should **never** perform the installation on top of the existing Qt library, *i.e.*, one should select a different location. This is to avoid rendering the KDE Desktop environment unusable since it has been compiled with the older Qt library.

To install the Qt library for Unix/Linux and Mac OSX:

Download the compressed archive `qt-x11-free-3.2.x.tar.gz` (Unix/Linux) or `qt-mac-free-3.x.x.sit` from the Trolltech Web site http://www.trolltech.com by going to the Downloads section and clicking on the **Qt/X11 Free** or the **Qt/Mac Free** link.

Uncompress the archive and copy the Qt directory to a desired location.

Read the `INSTALL` ASCII file in the Qt directory for environment settings and after setting the necessary environment variables, and from the Qt directory run

```
./configure -thread -no-xft -fast
```

Note the `./` in front of `configure` to make sure the correct one is invoked. The switch `-thread` is needed since some of the GUIs use multi-threaded processes and the switch `-fast` is optional to speed up the Qt building process by not building the numerous examples and tutorials. The switch `-no-xft` is needed on some systems to prevent problems related to fonts not used by the GUI anyway.

Once `configure` has finished, run `make`, again from the Qt directory, to build the library. Note that this can take a few hours on GHz range machines.

On Windows, we provide the Qt library in the form of a dynamic link library (dll), which is used by the graphical user interfaces `egs_gui.exe` and `egs_inprz.exe` which are distributed in binary form. If for some reason the user intends to re-compile these GUIs, he/she will have to obtain a licensed version of the Qt library.

3 How to define the environment

In the above we have already mentioned what needs to be done to define the EGSnrcMP environment but here we will summarize.

On the Windows systems, the installation GUI handles all the definitions needed (`$EGS_HOME, $EGS_CONFIG` etc).

On Linux/Unix systems one must include some definitions in the shell initialization files `.cshrc, .bashrc` or `.profile` depending on which shell you use for interactive runs.

```
.cshrc
setenv EGS_HOME path_to_directory_holding_users_codes
setenv EGS_CONFIG path_to_file_holding_conf_file
source path_to_hen_house/scripts/egsnrc_cshrc_additions
```

Note that `EGS_HOME` is typically `$HOME/egsnrc/` and `EGS_CONFIG` is typically `path_to_hen_house/specs/name.conf`.

This latter file defines `$PATH` to include `$HEN_HOUSE/bin/$my_machine` and `$EGS_HOME/bin/$my_machine`.

```
.bashrc
export EGS_HOME=path_to_directory_holding_users_codes
export EGS_CONFIG=path_to_file_holding_conf_file
.  path_to_hen_house/scripts/egsnrc_bashrc_additions
```

This latter file defines `$PATH` to include `$HEN_HOUSE/bin/$my_machine` and `$EGS_HOME/bin/$my_machine`.

Note that the definition of `EGS_HOME` requires the slash at the end.

One must also set up the information needed for doing runs in batch mode on the user's system. This is discussed below in section 6, page 25.

4 How to create an executable user-code

In the original EGSnrc environment and the EGS4 Unix environment, compilation of user-codes, both the Mortran and Fortran steps, was handled by C-shell scripts and made use of `user-code.configuration` files which told the scripts what source code files to include in a given user code.

In the EGSnrc system, this approach has been replaced by the Makefile approach which can be used on both Unix/Linux systems and on Windows systems (by way of the GNU `make` utility). This approach is much more flexible and means one system can be maintained for the two major operating environments. At the same time, some scripts have been written which allow the old compilation commands and syntax to be used, just to make the transition easier.

As discussed above (section 2.1.2, page 10 and section 2.2.3, page 14) an important part of the process is to have various environment variables set (*viz.*, `EGS_HOME, EGS_CONFIG` and `HEN_HOUSE`) and on Unix/Linux systems to include a variety of other variables by sourcing either the `egsnrc_cshrc_additions` or `egsnrc_bashrc_additions` files from `.cshrc` or `.bashrc` files (depending on which shell you work in interactively).

In the following we start by describing the use of `Makefiles` to compile EGSnrc user-codes and then describe the use of the GUIs and the scripts.

4.1 Compilation using Makefiles

Makefiles are a system for compilation whereby dependencies are formally defined and when a compilation is asked for, the Makefile only does the compilation for those components which have changed. In this context, compiling includes the Mortran pre-processor step, the Fortran compilation step and the linking step. These steps combine to create an executable file from Mortran source files.

4.1.1 Default use of `make`

To compile a user-code, go to the directory containing the `user-code.mortran` source (usually `$EGS_HOME/user-code`) and issue the command `make`. This default use of `make` will create the user-code executable with optimization turned on.

When one issues the command `make` on the area `$EGS_HOME/user-code`, this first uses the file `$EGS_HOME/user-code/Makefile` (see appendix B, page 53) which in turn includes the files `$EGS_CONFIG`, `user-code.make` and `$HEN_HOUSE/makefiles/standard_makefile`. Note that `$EGS_CONFIG` also includes the files `$HEN_HOUSE/specs/all_common.spec` and `$HEN_HOUSE/specs/unix.spec` or `$HEN_HOUSE/specs/windows.spec`. Fortunately the user does not need to touch any of these files except `user-code.make`.

4.1.1.i `user-code.make`

The file `user-code.make` is where the user tells the system what pieces of source code to

include in the user-code. An example is given in Appendix B.2 (see page 55). By default the user-code includes nine files, as defined by the following statement in `$HEN_HOUSE/specs/all_common.spec`:

```
SOURCES =  $(EGS_SOURCEDIR)egsnrc.macros \
           $(MACHINE_MACROS) \
           $(RANDOM).macros \
           $(USER_CODE).mortran \
           $(RANDOM).mortran \
           $(EGS_UTILS)nrcaux.mortran \
           $(MACHINE_MORTRAN) \
           $(EGS_SOURCEDIR)egs_utilities.mortran \
           $(EGS_SOURCEDIR)egsnrc.mortran
```

Note that each of the continuation lines above begins with a tab. These defaults can be overridden by including a modified version of the above statement in `user-code.make`, but otherwise this statement is not needed in `user-code.make`. This `.make` file can often be very simple.

4.1.2 Other targets with `make`

Makefiles define 'targets' to be made and these can include a variety of options. The following targets are defined in the EGSnrc Makefiles file:

```
make              Build the user-code executable with optimization
make opt          turned on.

make noopt        Build the user-code executable with optimization
                  turned off.

make debug        Build a user-code executable for debugging

make fortran      Mortran step only.

make clean        Remove the fortran file, mortjob file, mortlst file
                  and the various executables.
```

4.1.3 Running make for different configurations

As mentioned before (section 2), EGSnrc config files define important system specifications and the environment variable `EGS_CONFIG` points to the current config file. If a user wants to build a user-code for a config other than the current one, all he/she has to do is invoke `make` and pass a new value for the `EGS_CONFIG` environment variable to it, *i.e.*

```
make EGS_CONFIG=new_config_file
```

replacing temporarily the value of `EGS_CONFIG`. This way any variable values used in the

Makefiles can be replaced. For more information on EGSnrc configs and config files the user is refered to section 9.

The general command for running `make` is:

```
make [opt|nooopt|debug|fortran|clean] [EGS_CONFIG=new_config_file]
```

4.2 Compilation using `egs_gui`

`egs_gui` is a graphical user interface (GUI) for doing various tasks related to EGSnrc. When using `egs_gui`, one does not need to go to the user-code directory since one is given an explicit choice between all of the user-codes that are available. On Unix/Linux the alias `egsgui` runs the GUI in the background and filters out the diagnostic messages whereas direct use of `egs_gui` runs in the foreground and prints a variety of diagnostic messages to the terminal.

If you want to change the config file prior to doing the compilation, click on the settings icon in the GUI and select one of the available config files.

Alternatively, if you have write permission on the `$HEN_HOUSE`, you can use the 'New config' screen in the GUI to create a new config file.

For the specific case of the RZ user-codes, the user can also build these codes using the `egs_inprz` GUI in a similar fashion as with `egs_gui`. On Unix/Linux the alias `egsinprz` runs the GUI in the background.

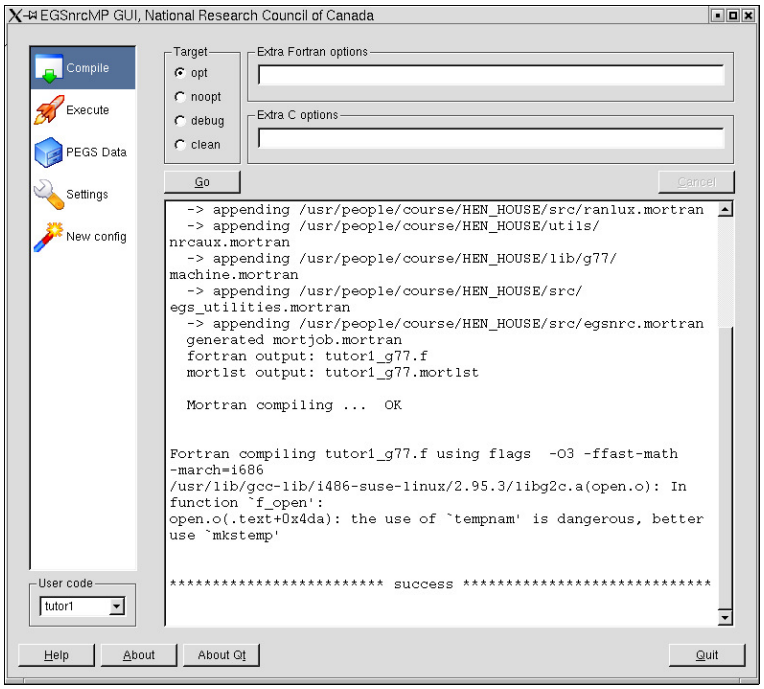


Figure 1: Screen shot of the `egs_gui` after compiling tutor1.

4.3 Compilation using `mf user-code` (Unix/Linux specific)

For compatibility with the old usage, the script `$HEN_HOUSE/scripts/compile_user_code` has been written. It has several possible arguments.

```
compile_user_code [mf|m] user_code [a] [opt|noopt|debug] [config=XXX]
```

The script compiles user_code. All other options are optional and in fact the parameter 'a' is not used (present for compatibility)
`mf` => Mortran and Fortran compile and then link
`m` => Mortran compile and create the Fortran file
`opt` => use optimization
`noopt` => use no optimization
`debug` => create executable ready for a debug run
`config=XXX` => use the config file specified as a full pathname instead of the default which is `$EGS_CONFIG`.

4.4 Location of executables

After creation of the executable file, it is placed on the area specified by
`$HEN_HOUSE/bin/$my_machine` if it is a system wide file such as `mortran3` or `pegs4` or on `$EGS_HOME/bin/$my_machine` if it is an individual's user-code.

5 Running a standard EGSnrc user-code interactively

Once a standard user-code has been built, one can execute it using command line options or through one of the two available GUI's, *viz.*, `egs_gui` or `egs_inprz`. The `egs_inprz` GUI creates/modifies input files, compiles and runs the NRC RZ user-code suite. In contrast the `egs_gui` GUI compiles and executes **any** EGSnrc user-code and creates PEGS4 data files but does not create input files. On Unix/Linux systems one can also use the scripts `run_user_code` and `run_user_code_batch` to execute the user-code interactively and to submit EGSnrc jobs to a queuing system respectively (see section 6 page 25).

5.1 Executing a user-code using `egs_gui` and `egs_inprz`

The `egs_gui` can be used under Windows or Unix/Linux to execute an already compiled user-code. The use is mostly self-explanatory but a few points are worth noting.

- The user can select where to look for the PEGS4 data set. By default the GUI looks on either the user's PEGS4 area (`$EGS_HOME/pegs4/data`) or the system PEGS4 area (`$HEN_HOUSE/pegs4/data`). If data files are obtained from other locations, these are remembered by the GUI
- In the lower left corner the user selects which user-code to execute. The available list consists of all subdirectory names in the `$EGS_HOME` area.
- The input file name does not need the extension (which must be `.egsinp`).
- **On Windows** the batch option is currently not available

As mentioned above, if the user wants to run the RZ user-codes with the ability of modifying/creating input files, `egs_inprz` can be used instead of `egs_gui`. Pressing on the execute button of the GUI brings up a dialog with the execution options which are identical to those in `egs_gui`. For more information on how to work with `egs_inprz`, the user can start the GUI and press on the Help button to display its manual in html format.

On Unix/Linux systems the following aliases are defined in the shell addition files `$HEN_HOUSE/scripts/egsnrc_[bash|csh]rc_additions`:

`egsgui` is aliased to `$HEN_HOUSE/bin/$my_machine/egs_gui` and runs it in background and ignores the usually unneccesary diagnotic messages from the script.

`egsinprz` is aliased to `$HEN_HOUSE/bin/$my_machine/egs_inprz`

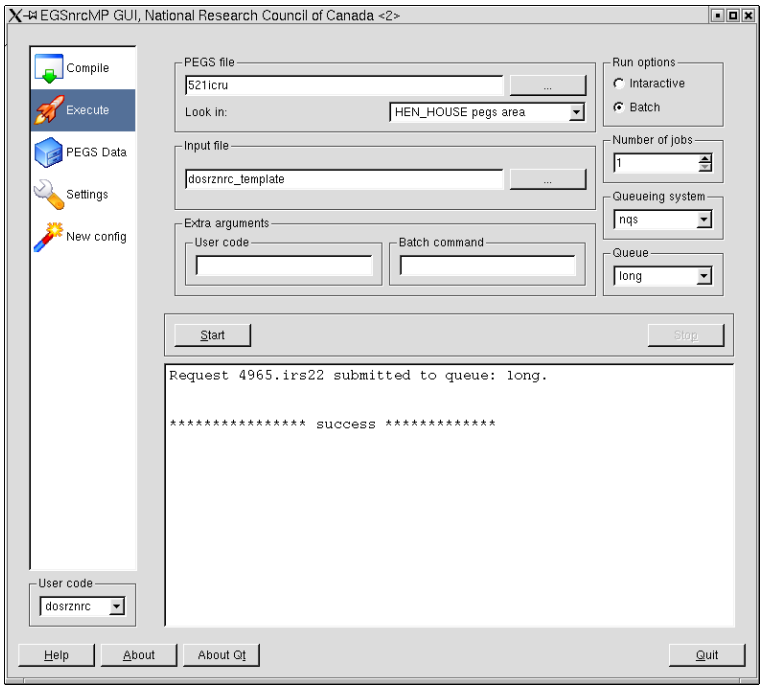


Figure 2: Screen shot of the `egs_gui` after running a DOSRZnrc job in batch mode.

5.2 Executing an NRC user-code using command-line arguments

User codes can be executed directly on the command line in a Unix or DOS shell using

```
user_code -p pegs_file [-i ifile] [other args]
```

where `user_code` is the name of the user code (*e.g.* `tutor1, cavrznrc`, etc) and `pegs_file` is the name of the PEGS4 data file. The `-i ifile` argument specifies that input comes from the file named `ifile.egsinp` on the user code area, this argument is only necessary if the user code requires an input file. `other args` are other optional arguments (see below). For instance, to execute the `tutor1` tutorial code, which needs the PEGS data file `tutor_data.pegs4dat` and no input, use

```
tutor1 -p tutor_data
```

To execute the cavrznrc user code using the `cavrznrc_template.egsinp` file distributed on the `cavrznrc` area, use

```
cavrznrc -p 521icru -i cavrznrc_template
```

Here is a list of all standard arguments understood by EGSnrc user codes that employ the `egs_init` subroutine (see section 6.4 on page 26 concerning the input for batch runs):

`-i or --input ifile`	`Specifies that the input file is ifile.egsinp`
`-o or --output ofile`	`Output data will be written to ofile.egslog, ofile.egslst, etc., instead of ifile.egslog, ifile.egslst, etc.`
`-p or --pegs-file file_name`	`Use the pegs4 data file file_name.pegs4dat. The system will look for it in the HEN_HOUSE and the users pegs4 data areas`
`-H or --hen-house dir`	`Change the HEN_HOUSE to be dir instead of the directory specified in the machine.macros file.`
`-e or --egs-home dir`	`Change EGS_HOME to be dir instead of the directory specified by the EGS_HOME environment variable`
`-b or --batch`	`Specify this is a batch run. See next section.`
`-P or --parallel nparallel`	`Run nparallel jobs simultaneously`
`-j or --job njob`	`This is job # njob`
`-h or --help`	`Print this help message and exit`

Note that arguments can be given in arbitrary order.

5.3 Executing a user-code using the scripts (Unix/Linux specific)

For compatibility with the old usage, the script `$HEN_HOUSE/scripts/run_user_code` has been written and the command `ex` has been aliased to this script. See the next section for the corresponding `exb` command. The general command for running `ex` is

```
ex user_code input_file pegs_file [noopt]
```

the first three parameters **must** be entered in that order to mantain backward compatibility with the old system. The input file and the PEGS4 data file can be entered with or without extension.
`noopt` is optional and tells the script to use the executable compiled with no optimizations.

6 Running a standard EGSnrc user-code in batch mode

6.1 Informing EGSnrcMP about your batch queuing system

Before running an EGSnrc user-code in batch mode, you must inform the EGSnrcMP environment about what queuing system you are using. At present batch operations are not available for Windows.

The default batch submission system assumed is the standard Unix job submission tool `at`. In the directory `$HEN_HOUSE/scripts`, three files are provided which contain specific definitions for the `at`, `NQS` and `PBS` batch submission systems. These files are named:

`batch_options.batch_system`

where `batch_system` stands for `at`, `nqs` or `pbs`, the three systems that happen to be available at NRC. If the user wants to make NQS, PBS or any other system the default job submission system, he/she can define the environment variable `EGS_BATCH_SYSTEM` to be `nqs, pbs` or the name of the other queuing system for which they have created a `batch_options.batch_system` file modelled on the exisiting files.

`batch_options.at`
`batch_options.nqs`
`batch_options.pbs`

To use the names of the queues at your local installation, you must edit the names found near the bottom of the appropriate `batch_options.batch_system` file on `$HEN_HOUSE/scripts`.

6.2 Using `exb` or `run_user_code_batch`

The script `$HEN_HOUSE/scripts/run_user_code_batch` (alias: `exb`) is for submitting EGSnrc jobs to a queuing system taking advantage of the new implementation for parallel processing in EGSnrc (see section 12, page 40). The general command for using `exb` is

```
exb user_code input_file pegs_file [noopt] [queue_type] [options]
```

The first three parameters **must** be entered in that order to maintain backward compatibility with the previous EGSnrc/EGS4 systems. The input file and the PEGS4 data file can be entered with or without extension. The rest of the parameters can be in an arbitrary order. The [options] can be any combination of the following

`batch=batch_system`	=> batch submission system
`eh=new_egs_home`	=> Overrides environment variable `EGS_HOME`
`hh=new_hen_house`	=> Overrides environment variable `HEN_HOUSE`
`config=new_egs_config`	=> Overrides environment variable `EGS_CONFIG`
`p=number_of_jobs`	=> Number of parallel jobs to submit to the queue

`noopt` (optional) tells the script to use the executable compiled with no optimizations.
`queue_type` gives the name of the batch queues used at your local installation. By default, `queue_type` is set to `long`. This is discussed further in section 6.1 above.

6.3 Using batch mode with the GUIs

On Unix/Linux if the batch mode is selected in the GUIs, then the queuing system and queue inputs become active. The default values are those in use at NRC. The GUI recognizes which queuing systems are available by looking on `$HEN_HOUSE/scripts` for files in the form `batch_options.batch_system` where `batch_system` is the name of the remote job submission system.

6.4 NRC user-code command-line arguments for batch mode

In section 5.2 on page 23, the command-line inputs to the standard NRC user-codes are described. There is one further parameter available for jobs which are being run in batch, *viz.*:

```
-b or --batch          Specify this is a batch run.  The difference between
                       a batch run and an interactive run is that in
                       batch mode unit 6 is connected to a file,
                       whereas in interactive mode unit 6 output goes
                       to the standard output. The file name in a batch
                       run is determined as follows:
                       * it is set to ofile.egslog, if ofile was
                         specified with the -o option,
                       * it is set to ifile.egslog, if there was no
                         -o option used but an input file was
                         specified with -i
                       * it is set to test.egslog, if neither -i nor
                         -o was used
```

Note that the user does not use this feature directly since the GUIs and the `exb` script take care of using it properly.

7 How to create a new data set using PEGS4

7.1 Using `egs_gui`

By far the easiest way to create a new pegs data set is using the `egs_gui` GUI which allows you to input the standard variables needed and to search the density effect corrections easily. Figure 3 presents a screen shot of the GUI.

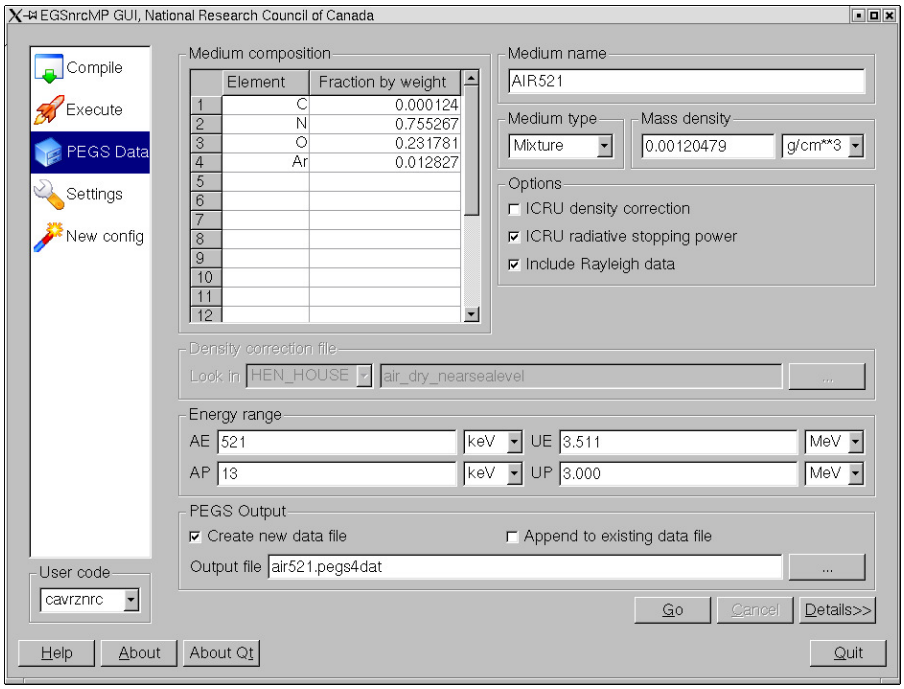


Figure 3: Screen shot of the `egs_gui` set to calculate a data set for air using the ICRU Report 37 density effect.

The GUI does not keep the input file but executes the code directly. There is also no output listing but the traditional output information is available by hitting the 'Details' button on the GUI after execution.

The PEGS4 data set created is put on `$EGS_HOME/pegs4/data` using the filename specified by the user, with an extension `.pegs4dat` and thus it is necessary that `$EGS_HOME/pegs4/data` and especially `$EGS_HOME` are defined. If one wants to create a dataset on the `$HEN_HOUSE/pegs4/data` area, this can be done by going to the Settings screen and entering the absolute path for `$HEN_HOUSE` as the value for `$EGS_HOME`, assuming of course that the user has write permission on `$HEN_HOUSE`. Note that the user is given the option of creating a stand-alone data file or of appending the new data to an already existing data set. The code also appends information about each data set created to the file `$EGS_HOME/pegs4/data/pegs4.log`. Here is an example of the `pegs4.log` file after 2 data sets have been created.

```
>more pegs4.log
------------------------------------------------------------------------
medium = ARGON512
data generated on: Thu Nov 13 14:46:22 2003
type=ELEM rho= 1.7800E-03 ne= 1 iunrst=0 epstfl=0 iaprim=1
Rayleigh data not included
cross section data from: /usr/people/course/HEN_HOUSE/pegs4/pgs4pepr.dat
form factor data from: /usr/people/course/HEN_HOUSE/pegs4/pgs4form.dat
ae=   5.12000E-01 ap=   1.00000E-03 ue=   6.00000E-01 up=   8.90000E-02
data written to: /usr/people/group1/egsnrc/pegs4/data/argon.pegs4dat
------------------------------------------------------------------------
medium = air521
data generated on: Thu Nov 13 15:04:37 2003
type=MIXT rho= 1.2048E-03 ne= 4 iunrst=0 epstfl=1 iaprim=1
Rayleigh data included
density correction file: /home/course/HEN_HOUSE/pegs4/density_corrections/
                                    compounds/air_dry_nearsealevel.density
cross section data from: /home/course/HEN_HOUSE/pegs4/pgs4pepr.dat
form factor data from: /home/course/HEN_HOUSE/pegs4/pgs4form.dat
ae=   5.21000E-01 ap=   1.00000E-03 ue=   3.51100E+00 up=   3.00000E+00
data written to: /home/group1/egsnrc/pegs4/data/air521icru.pegs4dat
```

7.2 Using `pegs4.exe`

The `egs_gui` GUI can handle most PEGS4 jobs perfectly well, but there are some options which are not available via the GUI (*e.g.*, the GUI assumes `IUNRST=0` always, and the GUI cannot handle some of the `PLOT` and testing functions which are described in Report PIRS-701[3], nor can it use the `GASP` feature for scaling the density of gases in atmospheres). In these cases one needs to use `pegs4.exe` directly.

The usage is somewhat different from the previous EGSnrc or EGS4 environments since all parameters are now passed on the command line instead of as parameters to a script. To run the code it is best to go to the directory where the input file is stored (`ifile.pegs4inp` has the same format as defined in the EGSnrc manual) and issue the command:

```
pegs4.exe -i ifile [-o ofile] [-a] [-d density] [-x CrossSectData] \
                                                [-e HEN_HOUSE]
 -i ifile         => ifile.pegs4inp is the input file
 -o ofile         => if present output file is $EGS_HOME/pegs4/data/ofile.pegs4dat
                     & if not present output is $EGS_HOME/pegs4/data/ifile.pegs4dat
 -a               => if present that output is appended to [i|o]file.pegs4dat
 -d density       => if present density effect is in density.density
 -x CrossSectData => if present use $HEN_HOUSE/pegs4/CrossSectData for
                     photoelectric cross sections
 -e HEN_HOUSE     => if present store data on $HEN_HOUSE/pegs4/data
                     rather than $EGS_HOME/pegs4/data
 -h or --help     => print this help screen
```

The density effect files are only used if both the `-d` input is present and the flag `EPSTFL=1` is part of the input file. This option is normally used to introduce an ICRU Report 37 density effect correction[4, 5], but can be used for any density effect in principle.

The `-x` option is there to allow use of the enhanced low-energy photo-electric cross section data set which basically uses the XCOM photo-electric and pair production cross sections[6] instead of the original EGS4 data set. The original data are in
`$HEN_HOUSE/pegs4/pgs4pepr.dat` and the new data are in
`$HEN_HOUSE/pegs4/pgs4pepr_xcom_full.dat`. For a description of the upgraded data set, see ref [7]. We want to thank Jan Seuntjens, Fadi Hobeila and Wamied Abdel-Rahman of McGill University for making this data set available in the correct format. If this option is not present, the standard `$HEN_HOUSE/pegs4/pgs4pepr.dat` data set is used.

8 The file structure of the EGSnrcMP environment

As in the original EGS4 and EGSnrc environment, the EGSnrcMP environment uses two areas, one for keeping the system files (`$HEN_HOUSE` or `%HEN_HOUSE%` on Windows) and another for the user files (`$EGS_HOME` or `%EGS_HOME%` on Windows). System files on `$HEN_HOUSE` can be common to several users or each user could have his/her own `$HEN_HOUSE` if desired. The rationale behind having two different locations for the system and the user(s) is maintenance. While a user could be working on a specific user-code, updates and corrections can be introduced in the system without interference.

8.1 `$HEN_HOUSE`

The directory structure of EGSnrc has changed, now files are regrouped according to their function within the system. See figure 4 for a summary of the `$HEN_HOUSE`. While subdirectories `doc, pegs4, mortran3, spectra` (former `ensrc_spectra`) remain unchanged, the general system source files are placed together in `$HEN_HOUSE/src`, more problem specific files (geometry dependent, user implementation dependent) can be found in `$HEN_HOUSE/utils` and the cross section data files used by EGSnrc are placed in `$HEN_HOUSE/data`. The tutor codes and the NRC user-codes are now located in `$HEN_HOUSE/user_codes`.

The installation program can place `$HEN_HOUSE` anywhere the person doing the installation has write permission, but it is most common to use `$HOME/HEN_HOUSE` on Unix/Linux systems. On Windows systems, the `$HEN_HOUSE` area *must not* contain spaces. This limitation is due to the use of the GNU `make` utility. One should therefore use *e.g.* `C:\HEN_HOUSE` and not use `C:\Documents and Settings` or `C:\Program Files`.

```
                    $HEN_HOUSE in EGSnrcMP environment

‘----- bin                              <=all executables
 |      ‘----- g77                     <=one subdirectory for each config
 |      ‘----- pgf77
‘----- data                             <=cross section data for EGSnrc
‘----- doc                              <=pdf versions of all manuals
 |      ‘----- pirs877
‘----- gui                              <=GUI's for installation, running etc
 |      ‘----- egs_configure
 |      ‘----- egs_gui
 |      ‘----- egs_inprz
 |      ‘----- egs_install
‘----- interface                        <=source re C interface to EGSnrc
‘----- lib                              <=source re machine/OS dependencies
 |      ‘----- g77                     <=one subdirectory for each config
 |      ‘----- pgf77
‘----- makefiles                        <=standard makefiles for all user-codes
‘----- mortran3                         <=preprocessor source and makefiles
‘----- pegs4                            <=data preparation package
 |      ‘----- data                    <=some prepared data sets
 |      ‘----- density_corrections     <=ICRU 37 density effect corrections
 |       |      ‘----- compounds
 |       |      ‘----- elements
‘----- pieces                           <=misc source
‘----- previewRZ                        <=Tcl/Tk code to display RZ geometries
‘----- specs                            <=config dependent spec files
‘----- spectra                          <=useful spectra for user-codes to use
‘----- src                              <=major sources for EGSnrc
‘----- user_codes                       <=source & related files: NRC user-codes
 |      ‘----- cavsphnrc                 <=cavity calculations spheres
 |      ‘----- dosrznrc                  <=dose calculations RZ geometry
 |      ‘----- examin                    <=examine PEGS4 data sets
 |      ‘----- flurznrc                  <=fluence calculations RZ geometry
 |      ‘----- edknrc                    <=spherical energy deposition kernels
 |      ‘----- sprrznrc                  <=stopping-power ratios RZ geometry
 |      ‘----- tutor1                    <= tutorial codes
 |      ‘----- tutor1c                   <= C version of the tutorial code
.
.
 |      ‘----- tutor7
‘----- utils                            <=geometry,source,plotting,timing etc
```

Figure 4: Structure of the $HEN_HOUSE area. Note that several sub-subdirectories in the gui subdirectories have been suppressed.

8.2 $EGS_HOME

In the original EGSnrc distributions, the user's EGSnrc area was assumed to be in `$HOME/egsnrc`. This is still a logical choice for Linux/Unix users but for generality, this area is now consistently referred to as `$EGS_HOME` so the user may place it anywhere and possibly switch between multiple versions. The same restriction on the path name as for `$HEN_HOUSE` applies to `$EGS_HOME`, i.e. Windows users *should not* select *e.g.* `C:\Documents and Settings\some_user\egsnrc` as their `EGS_HOME` directory. Otherwise the structure of the user's area is very similar to the past (see Figure 8.2) although the files associated with each user-code have changed somewhat. Note in particular that the previously hidden concatenation of all source code (`.mortjob.mortran`) is no longer a hidden file.

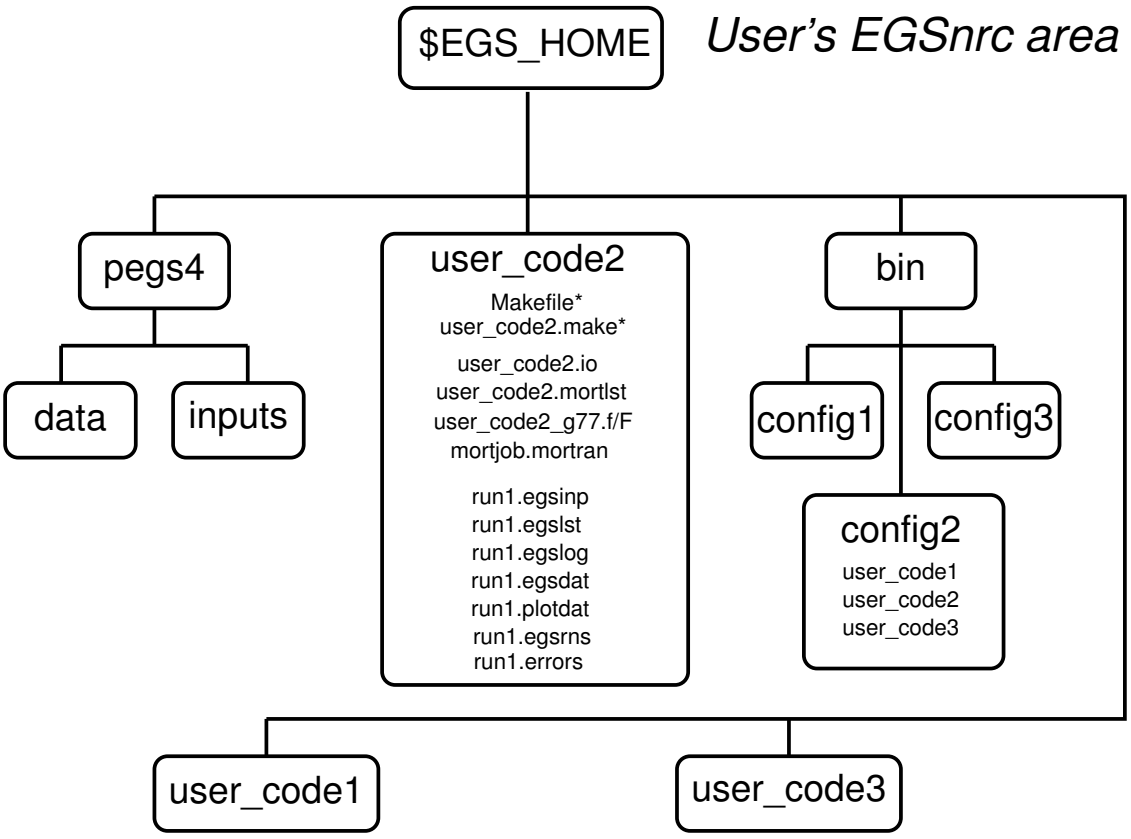


Figure 5: The user's area, `$EGS_HOME` in the EGSnrcMP environment.

9 How the EGSnrcMP environment works

9.1 $HEN_HOUSE/specs: Config files

9.1.1 What is a config file?

If a system is to work on a multi-platform environment, it needs to differentiate among combinations of operating systems and compilers for it to work properly. As described below, user-codes are compiled using `Makefiles` (see section 4.1) and the `config` file is an input for the `Makefiles` and the config name is used as part of a variety of file names. Therefore, when installing the system, the user should choose a meaningful config name that would for instance represent the settings selected by the user (compiler name, operating system, type of CPU, *etc.*). A suggestion is made to the user for the config name by the installing program, based on the operating system and CPU architecture (canonical name). The user can accept the suggested name, but any other name can be used and in particular, shorter ones may be preferable.

The config name is used by the EGSnrc system to create the binary and library directory for a given config, *i.e.*, executable and machine dependent files created for config `CONFIG_NAME` are stored in `$HEN_HOUSE/bin/CONFIG_NAME` and `$HEN_HOUSE/lib/CONFIG_NAME` respectively. All these settings are stored in a config file (`*.conf`) in `$HEN_HOUSE/specs` and the environment variable `EGS_CONFIG` must point to the current (desired) config file. On Linux/Unix systems this must be set in the user's `.cshrc` or `.bashrc` file (*e.g.*, in `.cshrc` one needs the statement `setenv EGS_CONFIG /home/course/HEN_HOUSE/specs/pgf77.conf`). Although we commonly use the config name for the config file name, *i.e.*, `CONFIG_NAME.conf`, this is not required and as with the config name, the config file name can be freely chosen by the user.

Depending on the installation procedure chosen, a config file is created by the `$HEN_HOUSE/scripts/configure` script or by the installation GUI. At any time, the user who owns the `$HEN_HOUSE` area may create additional config files using the `$HEN_HOUSE/scripts/configure` script (Unix/Linux only) or one of the 3 GUIs (`egs_install, egs_gui` or `egs_inprz`). The later method works for Windows and Linux. When running the `configure` script it is important that `$HEN_HOUSE` NOT be set from a previous (non MP) EGSnrc or EGS4 installation.

The process of creating a config file involves the following steps:

- Determine compiler name and compilation options to be used
- Check for the availability of various Fortran intrinsics (*e.g* `data_and_time, etime, hostnm`, etc.,) data types (*e.g.* `integer*8`), record length for binary I/O, etc.
- Based on the results of the previous step, create the files `machine.macros` and `machine.mortran` in `$HEN_HOUSE/lib/$my_machine` and the config file in `$HEN_HOUSE/specs`
- Compile the Mortran3 preprocessor and create the hex data file using these compiler settings and place them in `$HEN_HOUSE/bin/$my_machine`.

- Compile the PEGS4 program using the newly created Mortran3 executable and the new Fortran compiler settings

Appendix A presents examples of a config file created on a Linux system which has a pgf77 compiler (Portland Group). Also presented there are three files which are included in config files. `$HEN_HOUSE/specs/all_common.spec`, which is common to all config files and `$HEN_HOUSE/specs/unix.spec` and `$HEN_HOUSE/specs/windows.spec` which are common to config files on Unix/Linux and windows installations respectively.

The syntax of these config files is fairly simple and self-explanatory. Users are encouraged to take a look at them for a better understanding of their functions. Note that while config files can have any name, we are reserving the extension `*.spec` for specification files from the EGSnrc system and the `*.conf` for user-created config files. This convention can of course be replaced by any other.

It is worth noting that after creating a new config the user may want to compile the various GUIs for the new config. After switching to the new config, this is most easily accomplished by going to the `$HEN_HOUSE/gui` directory and typing `make`.

9.2 Pre-compiled components for Windows

On Windows (NT/2000/XP) many object and executable files are created at NRC and included in the distribution, *e.g.*, the graphical user interfaces, `egs_c_utils.obj`, *etc.*. These files can be found in `$HEN_HOUSE/pieces/windows` together with a copy of the Qt library dll and the executable for GNU `make`. At installation time they are just copied to their corresponding location.

10 Using external files: `egs_init` and `egs_finish`

Any EGSnrc simulation requires that the user-code have access to a variety of I/O files (*e.g.*, data files, listing files, standard output, standard input, *etc.*). These can be divided into two broad classes, those that are "routine" and used by all codes, and those specified by the user. In the original EGSnrc environment and the EGS4 Unix environment, the association between Fortran I/O unit numbers and external files was handled by the run scripts and the `.environment` files by linking the standard Fortran names for these files (`fort.n`) to the specific files of interest. In the EGSnrcMP environment this is done by explicitly opening and closing the files by their proper names from within the code. The advantage of doing this is that it works on Unix/Linux systems and on Windows systems.

To handle this the user must call two additional routines at the start and end of the user-code, *viz.*, `egs_init` and `egs_finish` and the user must create a `.io` file for each user-code to associate particular units with specific file name extensions. All one has to do is include a call to subroutine `egs_init` before any other executable statement in the user-code. This is called step 0 in the user-code (but really has to be the first executable step in the code and is often found in Step 2 since step 1 has no executable statements usually). Similarly, `egs_finish`

should be the last executable statement when the user-code is run (including after error conditions). Both of these subroutines are found in `$HEN_HOUSE/src/egs_utilities.mortran` and this source file is automatically included in all EGSnrc compilations.

The subroutine `egs_init` does many things.

- `egs_init` initializes arrays and default values (this replaces `BLOCK DATA` in EGSnrc since, for large arrays, the GNU compiler was very slow initializing block data)
- `egs_init` processes command-line arguments, including:
  - `-p pegs_file`
  - `-i ifile`
  - `-o ofile`
  - `-b`
  - `[-e| --egs-home] new_egs_home_location`
  - `[-H| --hen-house] new_hen_house_location`
- `egs_init` opens default EGSnrc data file units
- opens specific I/O units defined in the `.io file` found in the user-code area (`user-code.io`). See section 10.1 below for the format.
- moves files associated with this particular job to a temporary working directory which is a subdirectory of the `user-code` directory.
- prints a summary of the configuration and other system info to standard output (*i.e.*, the terminal).

The following comments apply to the above points.

The PEGS4 data file specified by `-p pegs_file`, is used if it is passed as a full pathname and found. Otherwise, the subroutine looks in turn for:
`$EGS_HOME/pegs4/data/pegs_file,`
`$EGS_HOME/pegs4/data/pegs_file.pegs4dat`
`$HEN_HOUSE/pegs4/data/pegs_file`
`$HEN_HOUSE/pegs4/data/pegs_file.pegs4dat`

If a `-i ifile` option was given as an argument, `ifile.egsinp` is opened as Fortran unit 5. The input file MUST be in the user-code directory, *i.e.*, on `$EGS_HOME`

If the run is a batch run (*i.e.*, the `-b` option was present on the command line), Fortran unit 6 is connected to an output file with a `.egslog` extension instead of going to standard output. The algorithm for determining the `.egslog` file name is the following: If the `-o ofile` option was present, the output to unit 6 will go to `ofile.egslog` If there was no `-o` option, but there is an input file specified with `-i ifile`, output to unit 6 will go to `ifile.egslog` If neither of the above is true, the output will go to `test.egslog`.

During run time, all output from the program (the `.egslog` file and all other files specified in the `.io file` (see section 10.1) are kept in the temporary working directory. After successful completion, `egs_finish` moves all output back to the user-code area and removes the temporary working directory. This implies that if, for whatever reason, the job terminates prematurely, the temporary working directory, with all the outputs, will be left behind.

The name of the temporary working directory is created using `egsrun_getpid_ifile_hostname`, if there was an input file, or `egsrun_getpid_noinput_hostname`, if there was no input file specified. Here, `getpid` is the process id returned by the `getpid()` intrinsic and `hostname` is the host name as determined by `egs_get_hostnm()`.

`EGS_HOME` is normally taken from the environment variable `EGS_HOME`. However, one can overwrite the environment variable by giving the `-e | --egs-home new_egs_home_location` command line parameter. `EGS_HOME` must be set, either via the environment or via the command line, otherwise the job will abort.

`HEN_HOUSE` is set to the value defined in `machine.macros`, which is created by the configure script or the configuration wizard in the GUI. This value can be overwritten by giving the `-H | --hen-house new_hen_house_location` option on the command line. `HEN_HOUSE` is NOT taken from the environment.

Subroutine `egs_finish` does three things.

- closes all the currently open I/O files
- moves any files on the subdirectory for this run back to the `user-code` directory
- prints a summary of the CPU time used on the standard output

10.1 `user-code.io` file format

The `user-code.io` file is specific to each user-code and is found in `$EGS_HOME/user-code` along with `user-code.mortran` *etc.*. This file is used to associate specific file names with Fortran unit numbers used within the user-code and subroutine `egs_init` then opens these files. The file format is an integer followed by a filename extension. The integer is the Fortran unit to be associated with the file in the user-code. The file extension is appended to the file name to be used for output. This will either be the name specified by the user for output (*e.g.*, by the `-o ofile` command line specification, see section 10 or, if that is not specified, then by default the input file base name (*i.e.*, `ifile` from `ifile.egsinp`). For cases in which neither `ifile` nor `ofile` are specified, the value of `test` is used. For example:

```
4    .egsdat
15   .egserr
1    .egslst
```

will result in connecting unit 4 to the file `ifile.egsdat` (or `ofile.egsdat` or `test.egsdat`, see above) to unit 4, `[i|o]file.egserr` to unit 15, *etc.*. Lines starting with a pound sign (`#`) are ignored. Note that file name extensions can not exceed 10 characters.

11 A C interface for EGSnrc

11.1 $HEN_HOUSE/interface: Interface for C-written user-codes

The EGSnrcMP system provides a C interface so that user-codes can be written in C (or C++). The necessary files are on `$HEN_HOUSE/interface`. The files are `egs_c_interface1.macros, egs_interface1.h` and `egs_interface1.c`. The standard Makefile to be included by makefiles for C user-codes is `c_makefile` in the `makefiles` subdirectory of `$HEN_HOUSE`. Examples of user-codes implemented in C are `tutor1c`, `tutor2c` and `tutor4c`.

11.1.1 Macros for the C interface: egs_c_interface1.macros

The purpose of the file `egs_c_interface1.macros` is:

- To slightly re-organize EGSnrc common blocks so that a relatively small number of common blocks needs to be exported as C structures. In particular, all variables that define a certain aspect of a cross section or the modelling of some process (*e.g.* `ibrdst`, which defines the bremsstrahlung cross section to be used, `spin_effects`, which defines whether spin effects should be modelled in electron elastic scattering, etc) are removed from the respective EGSnrc common block in which they are normally defined and combined into a single common block called `xsection_options` and are accessible from `the_xoptions`, which is a pointer to a `struct EGS_XOptions`.

- To re-define most of the arrays that have the dimension of the number of geometrical regions to be scalar variables (*e.g.* `ecut, pcut, smaxir,` etc.). The rationale behind this is that (i) the core EGSnrc system should be as much independent of the geometry as possible (ii) if the user needs a region-by-region variation of these quantities, memory can be dynamically allocated in the C written user-code depending on the actual number of regions instead of being statically allocated in the EGSnrc Fortran routines. There is a process defined that the user writing a code in C must follow to implement region-by-region variation of the various quantities.

- To re-define various EGSnrc macros that have been introduced into the EGSnrc system specifically for the purpose of providing non-Fortran interfaces to EGSnrc. One example of such a macro is `$start_new_particle;` which is defined as dummy in `egsnrc.macros` but is replaced with a call to a function that the user needs to implement in `egs_c_interface1.macros`.

11.1.2 Main interface header files: egs_interface1.h and egs_config1.h

The header file `egs_interface1.h` provides the main interface to EGSnrc. Its purpose is to define C structures corresponding to the exported EGSnrc common blocks, the EGSnrc functions visible to the user and the functions that the user must implement in order to have a functional EGSnrc user-code. `egs_interface1.h` includes the header files `egs_config1.h`

and `array_sizes.h`. The latter must be present in the user-code directory and must define the C-preprocessor macros `MXMED` to be the maximum number of different media and `MXSTACK` to be the maximum number of particles that the EGSnrc stack can hold. The former is created during the EGSnrc installation/configuration, defines the name mangling scheme of the Fortran compiler used to compile the EGSnrc Fortran routines via the C-preprocessor macros `F77_OBJECT` and `F77_OBJECT_`, and can be found in the `lib/$my_machine` subdirectory of `$HEN_HOUSE`. Different Fortran compilers generate different internal names for Fortran subroutines and functions (name mangling). For instance, given the Fortran subroutine `test`, one of the following C function names will be necessary to call this subroutine from C: `test, test_, TEST` or `TEST_`. It is even worse for a subroutine or function named `some_test`: `some_test, some_test_, some_test__, SOME_TEST, SOME_TEST_` and `SOME_TEST__` are all possible results. Because of this, one has to use `F77_OBJECT(test,TEST)` or `F77_OBJECT_(some_test,SOME_TEST)` to refer to Fortran subroutines, functions and common blocks. Because this is quite tedious, `egs_interface1.h` defines C-preprocessor macros for referring to important EGSnrc objects, *e.g.* `egsHatch()` can be used to call the `HATCH` subroutine, `egsInit()` can be used to call the initialization routine, etc. In addition, `egs_interface1.h` defines pointers to C-structures corresponding to the exported EGSnrc common blocks, *e.g.* one can use `the_stack->E` to access the array containing the energies of particles on the stack, instead of using the cumbersome `F77_OBJ(stack,STACK).E`.

11.1.3 What is needed for a C-written EGSnrc user-code

A user-code written in C must provide the following components:

- Implementation of the `void egsHowfar()` function. This function must calculate the distance `t` to the next geometry boundary along the direction of motion of the top particle on the stack. The position and direction of the top stack particle are in `the_stack->x[i], the_stack->y[i], the_stack->z[i]` (position) and `the_stack->u[i], the_stack->v[i], the_stack->w[i]` (direction). Here, `i = the_stack->np - 1`. This function must then compare this distance to the intended step length (in `the_epcont->ustep`) and

  - If you wish to discard the particle immediately, set `the_useful->idisc` to a positive value and return. If the particle is to be discarded after the step is complete, then set `the_useful->idisc` to a negative value and return.

  Otherwise,

  - just return, if `t > the_epcont->ustep`
  - set `the_epcont->ustep` to `t`, set `the_epcont->irnew` to the new region number and set `the_useful->medium_new` to the medium in the new region, if `t` $\leq$ `the_epcont->ustep`
  - If there is a region change and you wish to implement region-by-region variation of cut-off energies, maximum step size or non default mass density, then: set `the_epcont->ecut_new` to the desired value to have a non-default electron cut-off energy, `the_epcont->pcut_new` to the desired value to have a non-default photon

cut-off energy, set `the_etcontrol->smax_new` to have a maximum geometrical electron step-size restriction in the new region and set `the_useful->rhor_new` to have a non-default mass density in the new region.

- Implementation of the `void egsHownear(float *tperp)` function. This function must set `tperp` to the perpendicular distance to the closest boundary. If this is impossible (because *e.g.* the geometry is too complex), `tperp` must be set to 0 but this means that the entire calculation will be done in single scattering mode and take much longer.

- Implementation of the `void egsAusgab(const int *iarg)` function. This is where the actual scoring is being done, see section 3.7 of the EGSnrc manual for details[3].

- Implementation of the `void egsStartParticle()` function. This function is called just before the transport of a "new" particle starts (either directly from the source or a new particle from the particle stack). This function must set `the_useful->medium` to the medium index of the region the particle is in. If a region-by-region variation of threshold energies is to be obtained, `the_bounds->ecut` or `the_bounds->pcut` must be set to the desired value (depending on the particle charge).

- Provide a `main` function. The main function must

  – Call `egsInit(int argc, char **argv)` (where `argv` are the command-line arguments and `argc` their number) before accessing any EGSnrc data

  – Identify to the EGSnrc system all media that make up the geometry using the `int egsAddMedium(const char *medname, int len)` function. The return value of this function is the index of the medium in the internally maintained list of media (and so, if one calls this function with a medium name that was previously added, this function will return the index of the previously added medium).

  – Initialize cross section data needed at run time via a call to `egsHatch()`.

  – Simulate a given number of particles by placing one or more particles on the stack (`the_stack`) and calling `egsShower()`.

  – Close all Fortran I/O units and remove temporary files by calling `egsFinish()`.

The `main.c` files in the `tutor1c` and `tutor2c` directories of `$HEN_HOUSE/user_codes` use the EGSnrc C interface to implement the functionality of the traditional `tutor1` and `tutor2` user-codes. They can be taken as a starting point for developing your own user-code in C or C++.

11.1.4 The `tutor4c` user-code

The C interface as described above requires an additional file called `user_code.macros` (where `user_code` is the name of the user-code) to be placed in the user-code directory. This file defines the maximum number of media (`$MXMED`) and the maximum number of particles on the stack (`$MXSTACK`) used during the Mortran and Fortran compilation steps of the EGSnrc system. One must therefore make sure that these array sizes are the same as the

array sizes defined in the `array_sizes.h` header file. The `tutor4c` user-code provides a C implementation of `tutor4` where array sizes in the C part and in the Mortran/Fortran part are determined by a single file `array_sizes.h`. This alternative implementation requires that the EGSnrc Fortran sources are pre-processed by the C-preprocessor. On many systems (all Linux and Unix systems that we have tried and on Windows using the GNU compiler), this is automatically accomplished by having a `.F` extension for Fortran source files. Unfortunately, some Fortran compilers for Windows do not automatically pre-process a `.F` file or require weird options to do so. We have therefore deemed the `tutor4c` implementation not portable enough to designate it as the main EGSnrc C-interface implementation. But if the `tutor4c` implementation works for your operating system/compiler combination it is preferable to the main C interface.

11.1.5 Known limitations of C interface

- The Makefiles will only work with GNU `make`.
- On a Windows system, we can successfully compile and link the distributed user codes written in C using the GNU Fortran compiler and the Microsoft Visual Studio C++ compiler, but running the so created executables produces obviously wrong results. We are currently investigating the reasons for this behaviour.

12 Parallel runs

Running jobs in parallel requires special coding of the user-code. What is described in this section applies to the general purpose codes distributed with the EGSnrc system (DOSRZnrc etc).

In the original EGSnrc implementation of parallel runs for the NRC user-codes, the scripts asked each node to do (1/n)th of the calculation, where n was the number of nodes or computers used. This was inefficient when nodes had different CPU speeds and in practice meant that the calculation took as long as the slowest computer. In addition, parallel processing required a special script that had to "know" the syntax of input files for all parallelized user-codes so that it could modify the number of histories and the initial random number seed(s) when creating input files for the separate jobs. To overcome this drawback, the new system includes a parallel processing option for the NRC user-codes in which parallel processing is controlled via a "job control file" (JCF) placed in the user-code directory (the file name is `ofile.lock` where `ofile` is the name of the output file(s) that will be created by the job). This implies that `$EGS_HOME` must be the same for all nodes participating in the parallel execution. This can be accomplished, *e.g.* by placing the `$EGS_HOME` directory on an NFS file system for Unix/Linux installations or on a network share that is mapped to the same drive for Windows installation. The JCF is created by the first job, which reads from the input file the number of histories to be simulated, takes a small portion of them, writes the remainder to the JCF and starts simulating its portion of histories. Subsequent parallel jobs read the number of histories to be run and the number of jobs currently running (set to 1 by the first job) from the JCF, take a small portion of them and increment the

number of running jobs. When any job finishes execution of the small portion of histories it was simulating, it reads the JCF, takes some fraction of the remaining histories and writes the updated number of histories remaining to be done to the JCF. If no histories are left, the job performs statistical analysis, writes its data to `ofile.egsdat`, and decrements the number of running jobs in the JCF. If this number is greater than one, the jobs simply exits. If the remaining number of running jobs is zero (*i.e.* this is the only job still running), the job reads the data output by all other parallel jobs to `.egsdat` files, combines them, writes the combined results, removes the JCF and exits.

The advantage of this system is that, in the worst case, one waits for the slowest machine to execute the last group of histories, and if the groups of histories are small enough, this is a short period of time. In addition, no separate script is required that "knows" about the syntax of input files. Finally, no separate run that combines the results is required, as it was the case with the previous parallel processing implementation. The disadvantage is that each user-code must be coded to perform the above tasks.

12.1 egs_c_utils

To make the approach just described feasible, it is necessary to lock the JCF while a particular job is accessing it so that it becomes impossible for two jobs to modify the JCF at the same time. The JCF must be unlocked by the job when done with reading/writing to the file. Since it is impossible to lock/unlock a file using Fortran, tools for locking and unlocking a file were written in C. The corresponding C files are the source file `egs_c_utils.c` and the header files `egs_c_utils.h` and `egs_config1.h`. During installation, `egs_c_utils.c` is compiled and the resultant object file is copied together with the source and header files to `$HEN_HOUSE/lib/$my_machine`.

To see how these routines are used, please check the source for the DOSRZnrc or CAVRZnrc codes.

12.2 Parallel job submission

If the user-code is set up to handle them, parallel jobs can be submitted on Unix/Linux systems using one of the GUIs or the `exb` command. For both GUIs (`egs_gui` or `egs_inprz`), the only difference between submitting a single batch job (see section 5.1) and a parallel job is that the number of jobs must be set to greater than one in the relevant input box. In a similar way, when using the `exb` command, the only difference between submitting a single batch job (see section 6.2) and a parallel job is that one has to pass an additional argument of the form `p=njob` to the `exb` script, *e.g.*:

```
exb cavrznrc my_input my_pegs_data batch=pbs p=10
```

will initiate a `cavrznrc` run with 10 parallel jobs using `my_input.egsinp` as input file, `my_pegs_data` as a PEGS4 data set and PBS as the queuing system. It is important to note that there should not be any spaces for arguments of the from `xxx=yyy` passed to the `exb` script.

13 Efficiency considerations

For some of the standard NRC EGSnrc user-codes, we have obtained efficiency improvements, of the order of 30 to 40% on a given machine, through a variety of steps.

13.1 Static vs non-static

In the past, the EGSnrc system required 'static' switches on the Fortran compilers because EGS was written in the times when all variables in subroutines were saved from one call to the next. This is not standard Fortran and the compiler switches used to ensure the variables are all saved caused some inefficiency (of the order of 10%). We have ensured that these switches are no longer needed in the EGSnrc system itself and have done the same for the user-codes distributed with the system (CAVRZnrc etc). However, users with their own user-codes must ensure that this is the case for their user-codes before using the default switches which we now employ.

13.2 Fortran compilers

We have also found that Fortran compilers can vary a great deal in the speed of the code they produce on a given machine. When using the GNU g77 compilers, we have found that using version 3.3 of the gcc compiler can increase the execution speed of the calculation by more than 20% compared to using version 2.95.3. Using the Portland Groups pgf77 compiler can further increase the efficiency by another 5% compared to using gcc 3.3.

13.3 Random number generators

In table 7 of the EGSnrc Manual[8] there is a comparison of the timing of a particular calculation using different 'luxury levels' for the RANLUX random number generator.

We have found that by recoding the RANMAR random number generator to include a function subroutine to calculate a group of random numbers whenever the current supply has run out, we (a) avoid the frequent use of the subroutine call and (b) reduce the size of the Fortran source considerably because the random number generator is used so often that when the entire generator is coded in-line, the total length of the code increases substantially. The result is that for one of our major production codes (CAVRZnrc) there was a significant improvement in computation time, just because the executable is considerably smaller and fits into the fast memory.

With this change in place, we have found that using the RANMAR random number generator is about 10% faster than using the RANLUX generator with a luxury level of 1.

14 How to upgrade your old EGS user-code

If you have written an EGSnrc user-code, the major change required to use the EGSnrcMP environment is to insert the subroutine calls `call egs_init` as step 0 in the code and `call egs_finish` as the last statement in the code (see section 10, page 33). In addition you need to create a `user-code.io` file on `$EGS_HOME/user-code` (see section 10.1, page 36) and a `user-code.make` file on the same area (see section 4.1.1.i, page 18). The final step is then to copy a generic Makefile to the user-code area (see appendix B.2, page 53 for a Makefile example) and follow the directions in the header of the Makefile.

If your user code makes the assumption that local variables are automatically saved, you have to either find the appropriate switch for your compiler (*e.g.* `-fno-automatic` for the GNU compiler, `-static` for many Unix compilers, etc.) or you have to determine which variables really need to be static and save them using `save` statements in the appropriate subroutines.

If you want to adapt your user code to the new parallel processing implementation, use as a starting example the implementation in `cavrznrc.mortran` by searching for pieces of code enclosed between `#ifdef HAVE_C_COMPILER` and `#endif`. You will of course need a working C compiler that is capable of compiling `egs_c_utils.c`.

If you are starting from an EGS4 user code you must first convert it to using EGSnrc. This is described in detail of Section 5 of the EGSnrc Manual, Report PIRS-701[3].

15 Using the system at NRC

This section is just for people who are working at NRC.

At NRC the system is maintained under CVS control with the repository on the area `/home/cvsroot/HEN_HOUSE`.

To get the `HEN_HOUSE` and all the associated files onto your own area, execute:
`/home/cvsroot/CVSROOT/update_egs`.
This builds the system on `$HOME/HEN_HOUSE`, creating the subdirectory if not there and only updating changed files if you already have the system. This script also compiles various codes if needed because of changes (`mortran3, pegs4`). Note that the compilations are done for 3 compilers and need the following statement in your `.login` or `.cshrc` file:
`setenv LD_LIBRARY_PATH /home/iwan/gcc33/lib:$LD_LIBRARY_PATH` to pick up the libraries for gcc3.3.

To have access to the standard files, you need to have:
`setenv HEN_HOUSE`
and
`setenv EGS_CONFIG $HEN_HOUSE/specs/$your config file`
in your `.cshrc` and/or `.profile`.

15.1 Using CVS

To change a file on your local area:
`cvs edit filename` this makes it editable
`cvs ci [filename|directory name|blank]`
`blank` means check in/commit all files in this directory or below which have been changed.
`filename` means just this file
`directory` means just contents of that directory and below.
All options leave just readonly versions of each file affected. Note that if the `ci` fails, use `cvs update` which will merge your changed version with that in the repository and then allow you to `ci` (possibly after manually resolving problems with the merge).

To print a record of log messages:
`cvs log filename`: prints a record of the log messages for this file.

To get the changes that others have made to the system, and merge the changes into a file you are working on (but don't be editing it when you do this):
`cvs update [filename|directory|blank]`
`blank` means update all files in this directory or below which have been changed.
`filename` means update just this file
`directory` means update contents of that directory and below.
All of the above will try to merge changes with any files that you have editable and leave them editable.

The above command will **not** pick up any new directories in the repository unless you add `-d` right after the command `update`.

The preferred alternative is to execute `/home/cvsroot/CVSROOT/update_egs` which updates everything on `$HOME/HEN_HOUSE` and also rebuilds any files which are dependent on changes that have been made.

To add a directory or file:
First create the file or directory. Then:
`cvs add [directory|filename]` where both are full pathnames
and then for the files only, you must:
`cvs ci filename`

To compare file to what is in repository:
`cvs diff filename` does a diff against repository.

15.2 Creating a Distribution

On the NRC system, go to `$HOME/HEN_HOUSE/tools/distribution`. Make sure that you have environment variable `QTDIR` set. For some reason the default of `/usr/lib/qt3` does not work but `/home/course/qt_current` does work. Then issue the command `make linux_distribution` which leads to the Unix/Linux distribution being created on this area. You can create the entire distribution (including Windows specific files) by using just `make`. Note, however, that the various Windows executables and object files must have been created in advance on the Windows 2000 server by going to `$HEN_HOUSE/pieces/windows` and executing `make` there. This will only succeed if you have set up your environment (path including Visual Studio binary directory, GNU compiler binary directory and make binary directory, environment variable `QTDIR` set to point to the latest Qt Windows installation and path including `$(QTDIR)/bin`.

Note that the list of files included in the distribution is defined in `$HOME/HEN_HOUSE/admin/egsnrc_list` and near the top of this file the version identifier pre-pended to all tar files (*e.g.*, `V4_`) is defined.

The actual distribution is created by the program `make_distribution.cpp` which is on `$HOME/HEN_HOUSE/tools/distribution`.

A complete distribution also requires the install script, either `$HOME/HEN_HOUSE/tools/install_egs` or the gui equivalent.

16 Known restrictions

- **Red Hat 9.0**: Some of the graphical user interfaces, `egs_install` and `egs_inprz`, developed using the Qt library, hang and freeze the whole OS system when using the GNOME desktop environment and the Qt library version 3.1 that come with Red Hat 9.0. For reasons beyond our comprehension, the freeze does not occur with Red Hat 9.0, Qt 3.1 when using the KDE desktop environment

- **SUSE 9.0**: The GNU compiler distributed with SUSE 9.0 (a pre-release of version 3.3) is broken and fails to compile the `egs_inprz` GUI. If you are running SUSE 9.0 you should install the GNU compiler from source (see `http://gcc.gnu.org/`) or look for an updated rpm package from SUSE.

- **Windows**: Makefiles for the EGSnrcMP system will **only** work with GNU `make.exe`

- **Windows**: When writing user codes in C and using the Microsoft C/C++ compiler in combination with the GNU Fortran compiler, it is not possible to use single precision versions of the math library functions such as `acosf, atanf, cosf, logf,` etc. (use of these functions results in random crashes and/or wrong numerical results). **Solution**: Use the double precision math library functions instead.

- **Windows**: When running an EGSnrc user code compiled with the GNU fortran compiler from a drive that has its access permissions restricted and in addition there is a virus scanner monitoring each I/O operation running, execution fails with an error

coming from the GNU I/O library (`endfile: truncation failed in endfile...`)
Solution: Don't restrict access permissions or run from a drive where you have full permissions.

- **Windows**: The Lahey Fortran 95 compiler version 5.7 or earlier does not invoke automatically the C pre-processor on Windows. But on Linux it does ! This means that compiling the NRC RZ user-codes will fail.
  Solution: If the EGSnrcMP system is being used **only** on Windows, remove/comment-out all code between the #ifdef #endif constructs, including them, throughout the NRC RZ user-codes and any other user-code using this construct.

- **C interface**: The Makefiles for the user codes written in C only work with the GNU version of make.

- **Mac OSX with GNU compilers**: The building of the `tutor1c` user code fails because of an undefined reference to the `acosf` function. **Solution**: Replace the call to `acosf` in `egsAusgab()` with a call to `acos` with a double precision argument.

Appendix A: An example config file

Note that for all the files shown in the appendices, one should check that current versions of the files on the system for changes since these will not always be the current versions.

A.1: pgf77.conf

The following config file was created by the script `$HEN_HOUSE/scripts/configure` to use the pgf77compiler on a linux system. Note that this file includes the two files, `all_common.spec` and `unix.spec`.

```
#*************************************************************************
# EGSnrc config file
#
# Created by configure version 1.0 on Wed Nov 12 12:11:10 EST 2003
#
# Attention: all changes you make to this file may be lost next time
# you run configure.
#*************************************************************************
DSEP = /
my_machine = pgf77
canonical_system = i686-pc-linux-gnu
make_prog = make

HEN_HOUSE = /usr/people/course/HEN_HOUSE/
SPEC_DIR = $(HEN_HOUSE)/specs/
#
# Include the standard Unix spec file
include $(SPEC_DIR)unix.spec
#
# Include definitions common for all systems.
# These are mainly directory names, executable names, etc.,
# which are constructed from the previous definitions.
include $(SPEC_DIR)all_common.spec
#
# Fortran compiler name and options
F77 = pgf77
FCFLAGS =
FOPT = -fast
FDEBUG = -g -C
FLIBS =
FOUT = -o
#
# C compiler name and options
CC = gcc
C_FLAGS = -O2
```

```
#
# You can use the following make varibale for
# including additional object files in the link
# process
USER_EXTRA_OBJECTS =
#
# We have a C compiler that succesfully compiled egs_c_utils.c
# For simplicity in the Makefiles, we always include egs_c_utils.o
# in the link step, even if the functions are not used. This is not
# too wasteful as egs_c_utils.o is quite small.
EGS_EXTRA_OBJECTS = $(HEN_HOUSE)/lib/$(my_machine)/egs_c_utils.o
#
# FC_FLAGS gets used for compiling the EGSnrc fortran routines and for
# linking for EGSnrc user-codes written in C. We set FC_FLAGS by
# default to be given by $(FCFLAGS) $(FOPT). This is OK for most
# compilers. Unfortunately, some Fortran compilers insert a default
# main function and then complain about multiply defined main
# (theirs and the main of the user-code written in C), unless a special
# flag is passed. For instance, the PGI compiler needs -Mnomain.
# As I don't know how to test for this feature, it is left up to you
# to read the documentation of your compiler and adjust FC_FLAGS in
# case it does not work.
FC_FLAGS = $(FOPT) $(FCFLAGS) -Mnomain
```

A.2: all_common.spec

The following file (`$HEN_HOUSE/specs/all_common.spec`) is included in the config file and thereby into all standard Makefiles. Note in particular that this piece of code is where the list of "standard" SOURCE codes are defined. These can be overridden for a specific user-code in the `user-code.make` file found on `$EGS_HOME/user-code` (see Appendix B.2, page 55).

```
#############################################################################
#
# $Id  all_common.spec,v 1.3 2003/11/14 01:12:02 iwan Exp $
#
# This file is included by EGSnrc makefiles via the configuration
# file. It contains definitions of various directories, the default
# random number generator, the default set of sources, etc.
# These definitions are the same on all plarforms.
#
# Initial version: Iwan Kawrakow, March 2003.
#
# Copyright Iwan Kawrakow and the National Research Council of Canada
# iwan@irs.phy.nrc.ca
#
# This file is free software; you can redistribute it and/or modify
```

```
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
#############################################################################
#
# The standard Makefile
EGS_MAKEFILE = $(HEN_HOUSE)makefiles$(DSEP)standard_makefile
#
# The standard Makefile for user codes written in C.
EGS_C_MAKEFILE = $(HEN_HOUSE)makefiles$(DSEP)c_makefile
#
# Relative directories for executables and libraries
BIN_SUBDIR = bin$(DSEP)$(my_machine)$(DSEP)
LIB_SUBDIR = lib$(DSEP)$(my_machine)$(DSEP)
#
# Main installation executables and libraries
EGS_BINDIR = $(HEN_HOUSE)$(BIN_SUBDIR)
EGS_LIBDIR = $(HEN_HOUSE)$(LIB_SUBDIR)
#
# User executables and libraries
USER_BINDIR = $(EGS_HOME)$(BIN_SUBDIR)
USER_LIBDIR = $(EGS_HOME)$(LIB_SUBDIR)
#
# EGSnrc sources
EGS_SOURCEDIR = $(HEN_HOUSE)src$(DSEP)
#
# Utilities coming with EGSnrc
EGS_UTILS = $(HEN_HOUSE)utils$(DSEP)
#
# The default random number generator
# You can overwrite either on the command line or in your
# Makefile
RANDOM = $(EGS_SOURCEDIR)ranlux
#
# User code directory
EGS_USERCODES = $(HEN_HOUSE)user_codes$(DSEP)
#
# The main area user code directory
MAIN_UCODE_DIR = $(EGS_USERCODES)$(USER_CODE)$(DSEP)
#
# The fortran file (without extension) to be created by the Mortran compiler
FORTRAN_FILE = $(USER_CODE)_$(my_machine)
#
# The executable
EXECUTABLE = $(USER_BINDIR)$(USER_CODE)$(WHAT)$(EXE)
#
```

```
# What about OMEGA stuff? I'm puting it in here for now, so
# that I can play around with the local makefiles.
OMEGA = $(HEN_HOUSE)omega$(DSEP)
#
# Mortran stuff
MACHINE_MORTRAN = $(EGS_LIBDIR)machine.mortran
MACHINE_MACROS = $(EGS_LIBDIR)machine.macros
MORTRAN_EXE = $(EGS_BINDIR)mortran3.exe
MORTRAN_DATA = $(EGS_BINDIR)mortran3.dat
#
# The standard set of files
SOURCES = $(EGS_SOURCEDIR)egsnrc.macros \
   $(MACHINE_MACROS) $(RANDOM).macros \
   $(USER_CODE).mortran \
   $(RANDOM).mortran $(EGS_UTILS)nrcaux.mortran $(MACHINE_MORTRAN) \
   $(EGS_SOURCEDIR)egs_utilities.mortran $(EGS_SOURCEDIR)egsnrc.mortran
#
# PEGS4 stuff
PEGS4_EXE = $(EGS_BINDIR)pegs4.exe
#
# Replacement for various scripts
# Do we need this ?
EGS_COMPILE = $(EGS_BINDIR)egs_compile$(EXE)
#
# The fortran file extension
FEXT = f
#*********************************************************************
# The following set of definitions is for interfacing EGSnrc to C/C++
#*********************************************************************
#
# C/C++ interafece file directory
#
C_INTERFACE = $(HEN_HOUSE)interface$(DSEP)
#
# Include directories for the C/C++ compiler
C_INCLUDES = -I$(HEN_HOUSE)lib$(DSEP)$(my_machine) -I$(HEN_HOUSE)interface -I.
#
# The set of mortran files needed for the C/C++ interface
C_SOURCES =   $(EGS_SOURCEDIR)egsnrc.macros \
$(MACHINE_MACROS) $(C_INTERFACE)egs_c_interface1.macros \
$(RANDOM).macros\
$(USER_CODE).macros \
$(RANDOM).mortran \
$(MACHINE_MORTRAN) \
$(EGS_SOURCEDIR)egs_utilities.mortran \
$(EGS_SOURCEDIR)egsnrc.mortran
```

A.3: unix.spec

The following file, $HEN_HOUSE/specs/unix.spec, is used on Unix/Linux systems and contains those aspects of the Makefile which are different on Unix and Windows systems.

```
#*********************************************************************
# $Id  unix.spec,v 1.4 2003/11/14 01:12:02 iwan Exp $
#
# This file gets used by Make on Unix systems.
#
# Copyright Iwan Kawrakow and the National Research Council of Canada
# Initial version: Iwan Kawrakow, March 2003
#*********************************************************************
# The suffix of an executable
EXE =
SHELL = /bin/sh
# Sometimes I want to just leave a line blank in the output.
# When I just say echo, stupid Windows prints ECHO is off.
# => I'm using echo $(empty) and defining $(empty) to be really empty on Unix
empty =
# Check if a file exists and if not, copy it from the second argument.
# If the file exists, check if it is more recent than the other
GET_FILE = if test -f $@; then \
          echo $(MAIN_UCODE_DIR)$@ is more recent than local version; \
        else \
          cp $(MAIN_UCODE_DIR)$@ $@; \
          echo Copied $(MAIN_UCODE_DIR)$@ to local area; \
        fi
CHECK_USER_CODE = \
        if test -f $(USER_CODE).mortran; then \
          if test -f $(MAIN_UCODE_DIR)$(USER_CODE).mortran; then \
            $(MAKE) -s $(USER_CODE).mortran \
              depend=$(MAIN_UCODE_DIR)$(USER_CODE).mortran \
              check_message="$(the_message)"; \
          else \
            echo "$(USER_CODE).mortran does not exist on HEN_HOUSE"; \
          fi \
        else \
          if test -f $(MAIN_UCODE_DIR)$(USER_CODE).mortran; then \
            cp $(MAIN_UCODE_DIR)$(USER_CODE).mortran $(USER_CODE).mortran; \
            echo "Copied $(USER_CODE).mortran from HEN_HOUSE area"; \
          fi \
        fi
# Some system dependent commands
REMOVE = rm -f
CHECK_DIR = test -d $@ ||
MKDIR = ( echo Creating directory $@ && mkdir $@ )
```

A.4: windows.spec

```
#*****************************************************************
# $Id  windows.spec,v 1.5 2003/11/14 01:12:02 iwan Exp $
#
# This file gets used by Make on Windows systems.
#
# Initial version: Iwan Kawrakow, March 2003
# Copyright Iwan Kawrakow and the National Research Council of Canada
# iwan@irs.phy.nrc.ca
#*****************************************************************************
# The file extension of executables
EXE = .exe
# Sometimes I want to just leave a line blank in the output.
# When I just say echo, stupid Windows prints ECHO is off.
# => I'm using echo $(empty) and defining $(empty) to be realy empty on Unix
#    but to be a . on Windows
empty = .
#
# Check if a file exists and if not, copy it from the second argument.
# If the file exists, check if it is more recent than the other
FILE_TEST = if not exist $1 ( copy $2 $1 & \
                echo "Copied $1 from $(dir $2) to local area" ) \
                else ( echo "$1 on $(dir $2) is more recent than local $1" )
# Use the above in a make rule for a file
#
GET_FILE = $(call FILE_TEST,$@,$<)
#
CHECK_USER_CODE = \
        if exist $(USER_CODE).mortran ( \
          if exist $(MAIN_UCODE_DIR)$(USER_CODE).mortran ( \
            $(MAKE) -s $(USER_CODE).mortran \
              depend=$(MAIN_UCODE_DIR)$(USER_CODE).mortran \
              check_message="$(the_message)" \
        ) \
        else ( echo $(USER_CODE).mortran does not exist on HEN_HOUSE ) \
        ) \
        else ( \
          if exist $(MAIN_UCODE_DIR)$(USER_CODE).mortran ( \
            copy $(MAIN_UCODE_DIR)$(USER_CODE).mortran $(USER_CODE).mortran \
            echo Copied $(USER_CODE).mortran from HEN_HOUSE area \
        ) \
        )
# Some system dependent commands
REMOVE = del /Q /F /S
CHECK_DIR = if not exist $@
MKDIR = echo Creating directory $@ && mkdir $@
```

Appendix B: Examples of a Makefile

The use of Makefiles is discussed in section 4.1, page 18.

B.1: `Makefile` (tutor1)

The `Makefile` is found on the same area as the user-code which is to be compiled. It is usually a very simple piece of code. All it really does is define the specific `USER_CODE` for the rest of the `Makefile` and then includes several other files which are part of the `Makefile`, *viz.*:

- the current config file (*i.e.*, `$EGS_CONFIG` – see an example config file above in Appendix A.1, page 47 and a general discussion of config files in section 9.1.1, page 32);
- `USER_CODE.make` from the same area as the user-code;
- the standard_make file (which does the bulk of the work). This is `$HEN_HOUSE/makefiles/standard_makefile` which is defined as `EGS_MAKEFILE` in `$HEN_HOUSE/specs/all_common.spec` (see appendix A.2 above) which is included as part of the config file.

```
#*********************************************************************
#
#  $Id  Makefile,v 1.2 2003/11/03 19:45:37 iwan Exp $
#
#  A generic Makefile for a EGSnrc user code that uses mortran source
#  files only.
#  To use it, copy it to your usercode directory and
#    - set the name of your user code in the line USER_CODE = ...
#    - create a file named xxx.make where xxx is the name of your user code
#      In the simplest possible case this file is empty.
#      You can use it to specify an alternative set of sources,
#      select the random number generator (ranlux vs ranmar), etc.
#      See the .make files in tutor1-tutor7 and the NRC RZ codes
#      for examples.
#
#  Initial version: Iwan Kawrakow, March 2003
#
#*********************************************************************
# First include the active EGSnrc configuration.
# EGS_CONFIG is a environment variable that must be
# set to point to the EGSnrc config file.
# This file defines the compiler name, compiler options, various
# directories, the standard set of Mortran sources needed to
# build a user code, etc. The configuration file is created
# during the EGSnrc installations and can be found in
```

```
# $HEN_HOUSE/specs.
include $(EGS_CONFIG)
# Then define the user code name
#
USER_CODE = tutor1
# User code specific definitions for Make
#
include $(USER_CODE).make
# That's all we need. Now we can use the standard EGSnrc Makefile to
# build the code with optimization, no optimization, debug and to also
# clean up. We have the following targets defined:
#
# make                    Build the user code executable with optimization
# make opt                turned on.
#
# make noopt              Build the user code executable with optimization with
#                         optimization turned off.
#
# make debug              Build a user code executable for debugging
#
# make fortran            Mortran step only.
#
# make clean              Remove the fortran file, mortjob file, mortlst file
#                         and the various executables.
#
include $(EGS_MAKEFILE)
```

B.2: An example user-code `make` file: `ranmar_test.make`

The following is an example of a user-code's individual `.make` file. It is chosen as an example because it changes the random number generator from the default `ranlux` generator to the optional `ranmar` generator and it redefines the pieces of SOURCE code which make up the user-code (in this case adding `$EGS_UTILS/timing.macros` and removing `$EGS_SOURCEDIR/egsnrc.mortran`). These default SOURCES are defined in `$HEN_HOUSE/specs/all_common.spec` (see section 16, page 48).

```
#************************************************************************
#
# $Id  ranmar_test.make,v 1.2 2003/12/08 04:28:56 dave Exp $
#
# Make definitions for ranmar_test
# Initial version: Iwan Kawrakow, March 2003
#
#************************************************************************
# The RNG
#
RANDOM = $(EGS_SOURCEDIR)ranmar
# We don't use parallel preprocessing => don't link egs_c_utils.o
#
EGS_EXTRA_OBJECTS =

# The mortran sources
# Note: order is important!
# Note: don't forget the leading tabs on continuation lines!
#
SOURCES = \
        $(EGS_SOURCEDIR)egsnrc.macros \
        $(EGS_UTILS)timing.macros\
        $(MACHINE_MACROS) \
        $(RANDOM).macros\
        $(USER_CODE).mortran \
        $(RANDOM).mortran \
        $(MACHINE_MORTRAN) \
        $(EGS_SOURCEDIR)egs_utilities.mortran
```

B.3: $HEN_HOUSE/makefiles/standard_makefile

```
# $Id  standard_makefile,v 1.8 2003/11/14 01:15:03 iwan Exp $
#**********************************************************************
#
#  The main portion of a Makefile for the EGSnrc system.
#  This file is included from all user code makefiles.
#  Initial version: Iwan Kawrakow, March 2003
#
#
#   Copyright Iwan Kawrakow and the National Research Council of Canada
#   iwan@irs.phy.nrc.ca
#
#   This file is free software; you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation; either version 2 of the License, or
#   (at your option) any later version.
#
#**********************************************************************
# We want to be able to build 3 different targets:
#    executable with optimization turned on (default)
#    executable without optimization
#    executable with debug symbols included
# The following is for distinguishing between the 3
#   $(WHAT) gets appended to the user code name. It is empty for a default
#           build (optimization turned on), is = _noopt for no optimization
#           and _debug for debug symbols included.
#   $(OPTLEVEL_F) is the optimization or debug flags passed to the
#                 fortran compiler. Depending on the target (opt, noopt or
#                 debug), it gets replaced with the appropriate compiler
#                 flags.
#   $(OPTLEVEL_C) is not used right now
# FOPT and COPT are defined in the config file EGS_CONFIG
#
WHAT =
OPTLEVEL_F = $(FOPT)
OPTLEVEL_C = $(COPT)
depend =
the_message = $(USER_CODE) on HEN_HOUSE area is more recent than local version
check_message = $(empty)
EXE_DIR = $(EGS_HOME)bin$(DSEP)$(my_machine)
# The following is the rule for building the executable
#   The first line says that the executable is dependent on the
#   fortran file. Because of this dependence, make will rebuild the
#   executable whenever the prerequisite (the fortran file) is more
#   recent than the executable using the rule on the second line
#   (note the leading tab!!! without it it will not work)
```

```
# F77 and FLFLAGS are defined in the configuration file.
# OPTLEVEL_F is set depending on whether we build optimized, noopt or
# debug executable.
# $@ is an automatic make variable and gets replaced with the name
# of the target that initiated the invocation of this rule
# ( $(EXECUTABLE) in this case, which expands to something like
#  $HOME/egsnrc/bin/Linux/cavrznrc)
# So, the actual command passed by make to the sehll will look
# something like
#   g77 -O3 -o /usr/people/iwan/egsnrc/bin/Linux/dosrznrc dosrznrc_Linux.F
#
$(EXECUTABLE): $(FORTRAN_FILE).$(FEXT) $(EGS_EXTRA_OBJECTS) $(EGS_USER_OBJECTS) $(EXE_DI
        @echo $(empty)
        @echo $(empty)
        @echo Fortran compiling $(FORTRAN_FILE).$(FEXT) using flags '$(FCFLAGS) $(OPTLEV
        @$(F77) $(FCFLAGS) $(OPTLEVEL_F) $(FOUT)$@ $(FORTRAN_FILE).$(FEXT) $(EGS_EXTRA_O

# The following 2 rules are to make sure that the user executable directory
# exists.
#
$(EXE_DIR): $(EGS_HOME)bin
        @$(CHECK_DIR) $(MKDIR)
$(EGS_HOME)bin:
        @$(CHECK_DIR) $(MKDIR)
# In order to determine whether the executable needs to be re-built,
# make will examine its pre-requisits ($(FORTRAN_FILE).$(FEXT) in this case).
# The following says that $(FORTRAN_FILE).$(FEXT) is dependent on the
# files listed in $(SOURCES). If the fortran file does not exist, or
# any of its pre-requisits listed in $(SOURCES) is more recent,
# it will get re-made using the rule on the second line.
# MORTRAN_EXE is the name of the mortran compiler defined in EGS_CONFIG
# MORTRAN_DATA is the name of the mortran hex data file defined in EGS_CONFIG
# The -d option tells the mortran compiler to use the next argument as the
# name of the hex data file
# The -f option tells the mortran compiler to use all subsequent arguments
# until another -x option as names of files that need to be concatenated
# in the given order to make the mortjob.
# The -o7 option tells the mortran compiler to write the resulting fortran
# output to the file name given by the next argument (7 us the unit mortran
# uses for that => -o7)
# The -o8 option tells the mortran compiler to write the resulting listing
# output to the file name given by the next argument.
#
$(FORTRAN_FILE).$(FEXT): $(SOURCES)
        @echo Mortran compilation for $@
        @$(MORTRAN_EXE) -d $(MORTRAN_DATA) -f $(SOURCES) -o7 $@ \
        -o8 $(FORTRAN_FILE).mortlst
```

```
# The following rule is there just for convinience so that you can say
# make fortran
# to perform the mortran step only (corresponds to m user_code)
#
fortran: $(SOURCES)
        @echo Mortran compilation of $(USER_CODE)
        @$(MORTRAN_EXE) -d $(MORTRAN_DATA) -f $(SOURCES) \
        -o7 $(FORTRAN_FILE).$(FEXT) -o8 $(FORTRAN_FILE).mortlst
# The following rule tells make that $(USER_CODE).mortran is dependent on
# a file with the same name on the $HEN_HOUSE/$USER_CODE/ area.
# (DSEP is defined in EGS_CONFIG and is / for Unix and \ for Windows)
# GET_FILE is a system dependent shell script defined in EGS_CONFIG that
#  - copies the mortran file from the main area to the user area,
#    if the mortran file is not on the user area
#  - prints a warning if the mortran file is on the user area but the
#    main area file is more recent
#
$(USER_CODE).mortran: $(depend)
        @echo $(check_message)
check:
        @$(CHECK_USER_CODE)
# Build a optimized user code
opt:
        @$(MAKE) -s $(EXECUTABLE)
# Build a user code without optimization
noopt:
        @$(MAKE) -s WHAT=_noopt OPTLEVEL_F= OPTLEVEL_C=
# Build a user code for debugging
debug:
        @$(MAKE) -s WHAT=_debug OPTLEVEL_F="$(FDEBUG)" OPTLEVEL_C="$(CDEBUG)"
# Clean-up the user code directory.
clean:
        @echo $(REMOVE) mortjob.mortran $(FORTRAN_FILE).$(FEXT) $(FORTRAN_FILE).mortlst
        @$(REMOVE) mortjob.mortran $(FORTRAN_FILE).$(FEXT) $(FORTRAN_FILE).mortlst
        @$(MAKE) -s WHAT=_debug execlean
        @$(MAKE) -s WHAT=_noopt execlean
        @$(MAKE) -s WHAT= execlean
# Remove the executable.
execlean:
        @echo $(REMOVE) $(EXECUTABLE)
              $(REMOVE) $(EXECUTABLE)
# A 'phony' target always gets remade. As opt, noopt, debug, clean and
# execlean have no list of prerequisits so that make can determine whether
# they need to be remade, we declare them as phony.
#
.PHONY: opt noopt debug clean execlean
```

17 References

[1] D. W. O. Rogers, I. Kawrakow, J. P. Seuntjens, B. R. B. Walters, and E. Mainegra-Hing, NRC User Codes for EGSnrc, Technical Report PIRS–702(RevB), National Research Council of Canada, Ottawa, Canada, 2003.

[2] E. Mainegra-Hing, User Manual for inputRZ, a GUI for the NRC RZ user-codes, Technical Report PIRS–801, National Research Council of Canada, Ottawa, Canada, 2002.

[3] I. Kawrakow and D. W. O. Rogers, The EGSnrc Code System: Monte Carlo simulation of electron and photon transport, Technical Report PIRS–701, National Research Council of Canada, Ottawa, Canada, 2000.

[4] ICRU, Stopping powers for electrons and positrons, ICRU Report 37, ICRU, Washington D.C., 1984.

[5] S. Duane, A. F. Bielajew, and D. W. O. Rogers, Use of ICRU-37/NBS collision stopping powers in the EGS4 system, NRCC Report PIRS-0173, Ottawa, March (1989).

[6] M. J. Berger and J. H. Hubbell, XCOM: Photon Cross Sections on a Personal Computer, Report NBSIR87–3597, NIST, Gaithersburg, MD20899, 1987.

[7] J. P. Seuntjens, I. Kawrakow, J. Borg, F. Hobeila, and D. W. O. Rogers, Calculated and measured air-kerma response of ionization chambers in low and medium energy photon beams, in *Recent developments in accurate radiation dosimetry, Proc. of an Int'l Workshop*, edited by J. P. Seuntjens and P. Mobit, pages 69 – 84, Medical Physics Publishing, Madison WI, 2002.

[8] I. Kawrakow and D. W. O. Rogers, The EGSnrc Code System: Monte Carlo simulation of electron and photon transport, Technical Report PIRS–701 (4th printing), National Research Council of Canada, Ottawa, Canada, 2003.

Index