ESP-r Developers Guide

Edited by Jon W. Hand
Energy Systems Research Unit
of the University of Strathclyde

Version of 25 April 2012

Table of Contents

1 Introduction

This document covers a number of topics of interest to the ESP-r development community. It begins with a short history of how ESP-r moved from a small group of developers into a world-wide development community. It documents how an initially person-centric development process has scaled to a process that supports the contribution of scores of developers as well as maintaining the quality of the ESP-r distribution. Some sections have been revised and extended from community contributed sources in the *archive* and *manual* folders of the ESP-r distribution. Some sections are extracted from the IBPSA conference paper *Documentation of open-source simulation - addressing multiple points of interest* presented at BS 2009 in Glasgow.

The ESP-r development community is diverse. There are the hard-core geeks who focus on the calculation engine and others who evolve the underlying data structures and there are also others who make valuable contributions by updating the documentation in the code and in the handbooks as well as contributing new example models. And the user community is also active in contributing notices of glitches that they detect as well as wish lists.

Those who are new to the ESP-r community may find some portions of this document *peg the geek-meter*. Hopefully we can work past the jargon and to an understanding of how the community works, some tips for getting ESP-r running on your computer, and guidance for contributing to the evolution of ESP-r.

For active contributors this is both a reference and a working document which itself evolves to reflect the current working practices in the community. The latter is particularly important because many developers never physically meet.

There are sections related to the organization of the ESP-r distribution, suggestions for coding styles, procedures for checking the syntax of code and the numerical robustness of methods. There are also templates for creating the audit trail of the changes submitted and the notifications broadcast to the ESP-r community.

1.1 How we got to this point

In June 2002 the Energy Systems Research Unit of the University of Strathclyde in Glasgow (ESRU) announced that the simulation suite ESP-r would become an open source software project under the GNU license. The butterfly that set this storm in motion was the author's reading of "rebel code" by Glyn Moody. This book discussed the benefits and drawbacks of making software available beyond its original development community and freeing others to explore new uses. It argued that one can make a business plan around open source software. The next flap of the wings was passing the book to Prof. Joe Clarke of ESRU who decided to buy into the idea. The ESP-r development community debated this and adopted the (at the time) radical idea that the future of simulation lay in opening it up so that others could build on it and use it for purposes that no one in the existing community could imagine.

The decision to open source carried with it a number of technical and philosophical issues necessitating changes in how the developer and user community worked and communicated. Although many thousands of open source applications exist, few could be described as million line virtual physics laboratories. Many of the challenges since 2002 are unique to the whole-building technical domain while others confront open source projects in general.

Decisions were required on how to co-ordinate the contributions of existing and new developers so that the ESP-r distribution maintained its robustness as well as becoming a better platform for exploratory developments in simulation.

The traditional sequence of tasks undertaken by developers and the archivist in ESRU had evolved over a decade. The process might have seemed pedantic to outsiders, but there were few glitches in the million lines of code. The process relied on at degree of paranoia as well as the maintenance of a strict regime within which the actors performed their tasks.

Another mark of open software is adaptation to the goals of a community rather than its founding authors. Procedures must become sufficiently robust and largely independent of the individuals. The archivist role transferred from Joe Clarke in ESRU to Ian Beausoleil-Morrison initially at Natural Resources Canada and later at Carlton University in Ottawa.

One of the early tasks, when opening up ESP-r, was to scale up without becoming a burden on the archivist:

- Passing code to an archivist in ESRU relied on a manual regime of enforced by convention - these needed to be documented and codified.
- Detecting errors in coding and changes in predictions were manual processes. These processes needed to become part of the work flow as well as a design issue for new facilities.

- The transfer of files had a limited audit trail and required considerable attention to detail.
- User access to the source as a set of compressed archive files on a file server was inefficient. It was Linux and Unix platform-centric.

Many of the above issues were rooted in a person- centered version control system. Clearly what was required was a software based version control. ESP- r, as a community, was a late adopter of version control.

In 2001, 2003 and 2004 source code repositories were implemented at different development sites. These made use of CVS (concurrent versioning system) and were used to co-ordinate group coding and testing cycles. This diverse testing ran in parallel with the archivist's tasks.

In 2005 the repository moved from CVS to a versioning system named Subversion (svn). This allowed for a clearer audit trail, easier manipulation of files and folders and more options for merging and testing different development branches. It also automated the distribution of information about changes as they happened, provided facilities to view changes made by others.

ESP-r began with a core of developers who had access to a Sun Solaris box named sigma. Currently there are scores of developers and users who, with a few keystrokes, can access the repository from any location and who can know within minutes the latest changes. This virtual development environment is made possible because the ESP-r repository is held on an externally hosted environment with a domain (espr.svn.cvsdude.com).

The transition from a machine named sigma to a virtual development community with more than 60 branches and almost 8000 commits (sometimes as many as a dozen commits in a day) required time and the testing of many ideas. There was a lot of scratching of heads. Why must we jump through all of these hoops? Why can't you just take my five line change? Why does everyone get to see my mistakes?

The answers to such questions are essentially to ensure the quality of the ESP-r distribution and to hopefully identify glitches as soon as possible. The rituals surrounding the use of Subversion are in support of these goals. Such rituals take time to become a habitual. It certainly seems like magic (and black magic to novices) and working with an experienced developer can speed the process.

Of course, documenting procedures requires iteration. Actual practice evolves in subtle ways that are not part of the published checklist. Indeed, this document must also be a work-in-progress and requires that others test it and update it to reflect the current state of ESP-r.

2 The ESP-r distribution structure

The source distribution of ESP-r contains many resources for the development community. Beyond the contents of this document, those wishing to understand the organization of the code or the details of existing functions and subroutines within the Fortran, C and C++ code base can browse the *ESP-r Common Coding Resources* document.

Documentation of source code tends to lag the evolution of the code and readers are advised to also review the source code. This document should be considered work-in-progress. Check for updates regularly.

2.1 Source code layout

The source code of ESP-r is subdivided into a number of topic-focused folders. Each of the folders separates the source files by functional task, data structure and/or analysis domain. There is a rough order within the roughly million lines of code. There are chaotic corners which have yet to be consolidated and there is some duplication which awaits consolidation.

Below is the overall layout of the source distribution along with notes on the contents of each folder:

```
|-- src
|   |-- archive     documents for developers
|   |-- bin         scripts (batch command files) for developers and users
|   |-- bitmaps     images used by X11 version
|   |-- cetc        code from Natural Resources Canada and XML
|   |-- climate     climate data sets (ASCII versions)
|   |-- databases   materials, optics, constructions, plant components
|   |-- env         example Unix/Linux dot files
|   |-- esruXXX     ESP-r module specific source code
|   |-- esrucom     ESP-r common source code
  ...
|   |-- include     header files used for Fortran and C code
|   |-- lib         library code and user interface code
|   |-- manual      documentation about ESP-r and operating system variants
|   |-- shocc       library of occupant preferences
|   |-- training    example models for use in workshops and training
|   |-- tutorial    holds additional contextual help text for interfaces
|   `-- validation  models for use in formal validation e.g. BESTEST
`-- tester
    |-- additional_tests   infrequently used automated test models
    |-- scripts            scripts (batch command files) for running automated tests
    `-- test_suite         test models for automated tests
```

One key aspect of source code evolution is testing of changes and the tester folder includes a number of scripts to automate the process and over 250 test models to be run as part of the formal testing process. The tester folder are listed below:

```
tester
|-- additional_tests
|   |-- A42_combustion_cogen_comp_tests  notes to be added
|   |-- A42_fuel_cell_comp_tests         notes to be added
|   |-- ASHRAE-140                       notes to be added
|   `-- HOT3000_test_cases               notes to be added
|-- scripts
`-- test_suite
    |-- Annex42_fuel_cell                notes to be added
    |-- alberta_infil_model              notes to be added
    |-- ascii_dbs                        notes to be added
    |-- basesimp                         notes to be added
    |-- bld_PV                           notes to be added
    |-- bld_ground_reflectivity          notes to be added
    |-- bld_hc_ISO15099                  notes to be added
    |-- ccht_benchmark                   notes to be added
    |-- cellular_offices                 model with cellular offices
    |-- cetc_battery_model               notes to be added
    |-- elec_gain_into_zone              notes to be added
    |-- esru_benchmark_model             similar to validation/benchmark/QA
    |-- h3kreports                       notes to be added
    |-- idealized_hvac                   notes to be added
    |-- multi_year_simulations           notes to be added
    |-- plt_SDHW                         notes to be added
    |-- plt_adsorption_storage           notes to be added
```

```
    |-- plt_boundary_conditions          notes to be added
    |-- plt_elec_net                     notes to be added
    |-- plt_electric_HWT                 notes to be added
    |-- plt_lookup_table                 notes to be added
    |-- plt_pre_A42_PEMFC_model          notes to be added
    |-- plt_pre_A42_SOFC_model           notes to be added
    |-- plt_radiant_floor                notes to be added
    |-- plt_solar_collector              notes to be added
    |-- plt_trnsys_wrapper               notes to be added
    |-- plt_zone_heat_gain_coupling      notes to be added
    `-- pv_example                       notes to be added
```

The validation of ESP-r is also supported by a number of standard test models and scripts (automated command files) in the validation folder.

```
src
`-- validation
    |-- BESTEST
    |   |-- 195    test solid conduction
    |   |-- 200    test long wave radiation exchange at windows
    |   |-- 210    test long wave radiation external exchange on
    |   |-- 215    test long wave radiation external exchange off
    |   |-- 220    test long wave radiation internal exchange
    |   |-- 230    test infiltration
    |   |-- 240    test internal gains
    |   |-- 250    test exterior solar
    |   |-- 270    test south exterior solar
    |   |-- 280    test cavity albedo
    |   |-- 290    test south horizontal overhang
    |   |-- 300    test east and west external solar
    |   |-- 310    test east and west overhang and fins
    |   |-- 320    test thermostat dead-band
    |   |-- 395    test solid conduction
    |   |-- 400    test surface convection and long-wave exchange
    |   |-- 410    test infiltration
    |   |-- 420    test internal heat generation
    |   |-- 430    test external solar incident
    |   |-- 440    test internal solar absorptance
    |   |-- 600    test south solar transmission
    |   |-- 600FF
    |   |-- 610    test south overhang
    |   |-- 620    test east and west solar transmission
    |   |-- 630    test east and west overhangs and fins
    |   |-- 640    test night setback
    |   |-- 650    test venting
    |   |-- 650FF
    |   |-- 800    test thermal mass with no solar
    |   |-- 810    test thermal mass with solar
    |   |-- 900    test thermal mass and solar interaction
    |   |-- 900FF
    |   |-- 910    test south overhang and thermal mass
    |   |-- 920    test east and west mass and solar interaction
    |   |-- 930    test east and west shading and mass interaction
    |   |-- 940    test night setback and mass interaction
    |   |-- 950    test venting and mass interaction
    |   |-- 950FF
    |   |-- 960    test test passive inter-zone transfer
    |   |-- 990
    |   |-- climate     climate files for use with BESTEST
    |   `-- dbs         databases for use with BESTEST
    |-- CEN
    |   |-- 13791
    |   `-- 15265
    |-- analytical
    |   `-- conduction01
    `-- benchmark
        |-- CFD
        |   |-- Archive_Feb2004    archive of standard assessments
        |   `-- Models             models for testing CFD
        `-- QA
            |-- Archive_linux_X11_040309
            |-- Archive_linux_X11_230209   archive of earlier predictions
            |-- benchmark_model     a set of archaic test models
            |-- model         test models with older geometry format
            `-- model_1.1     test models with new geometry format
```

The documentation associated with ESp-r is found in the manual and the archive folders.

```
src
|-- manual
|   |-- Adding_features
|   |-- Data_model           formal description of the ESP-r data model
|   |-- ESRUlib              notes about library subroutines (out-of-date)
|   |-- Implement
|   |-- Manual               text for the manual (out-of-date)
|   |   `-- Figs             files for figures to the manual
|   `-- OS
|       |-- Apple            instructions for OSX install
|       |-- Cygwin           instructions for Cygwin install
|       |-- Linux            instructions for Linux install
|       `-- Native_windows   instructions for Native Windows install
```

There are a number of example models for use with training workshops and courses and these are located in the training folder structure.

```
src
|-- training
|   |-- 3_windows         model with different flow network window representations
|   |-- CFD_room          model with CFD domain
|   |-- acoustic
|   |   `-- EOS_atrium    model including acoustic calculations
|   |-- basic             a simple model with numerous variants
|   |-- burdie            a house with moisture issues
|   |-- cellular_bc       base case version of two cellular offices
|   |-- cellular_bound    cellular offices with upper and lower bounding zones
|   |-- cellular_contam   cellular offices with contaminate tracking
|   |-- cellular_cvvt     cellular offices with idealized CV air supply
|   |-- cellular_earth    cellular offices with earth tube air supply
|   |-- cellular_flh      cellular offices with floor heating
|   |-- cellular_furn     cellular offices with furniture and internal mass
|   |-- cellular_hires    cellular offices with higher resolution geometry
|   |-- cellular_hvac     cellular offices with HVAC
|   |-- cellular_hybrid   cellular offices with hybrid ventilation
|   |-- cellular_natv     cellular offices with operated windows
|   |-- cellular_pv       cellular offices with PV embedded in facade
|   |-- cellular_shd      cellular offices with shading obstructions
|   |-- cfd
|   |   |-- IEA_A20       IEA Annex 20 models
|   |   |-- M_Age         model including mean age of air
|   |   |-- RoomVent98    models used for RoomVent 1998 paper
|   |   |-- displ_vent    models demonstrating displacement ventilation
|   |   `-- rad_htg       model with radiant heating
|   |-- cg_ctl
|   |   |-- coupling      coupling of ESp-r and Radiance
|   |   |-- daylit_coef   model using Radiance daylight coefficients
|   |   |-- el_chrom      model with electro chromic optical controls
|   |   `-- static
|   |-- chp
|   |   |-- sport_cen     model of sports centre with co-generation
|   |   `-- unit
|   |-- constr
|   |   |-- adapt         model with adaptive thermophysical properties
|   |   `-- tp_sub
|   |-- el_chr_ctl        model with electro chromic optical controls
|   |-- flow              model demonstrating network flow
|   |-- gridding          model with 2D conduction gridding
|   |-- sun_space         house with a sun space
|   |-- svph              house with a sun space and solar ventilation preheating
|   |-- mould             model which includes mico-toxin parameters
|   |-- network           model with network flows
|   |-- office            a portion of an office building
|   |-- office_dfs        a portion of an office building with a double facade
|   |-- office_doctor     a portion of an medical practice (matches example in ESP-r Cookbook)
|   |-- office_vent       a portion of an office building with controlled facade vent
|   |-- pattern           a folder with sample operation files
|   |-- pid               a model for demonstrating PID controls
|   |-- plant
|   |   |-- ac_pp           a model with primitive part representations of air conditioning
|   |   |-- ahu             a model with air handler plant components
|   |   |-- coil_pp         primitive part representation of a fan coil
|   |   |-- conv_ac_sys
|   |   |-- hvac_bas
```

```
|   |   |-- hvac_vav        VAV system represented by plant components
|   |   |-- mixed_ac_sys
|   |   |-- solar           a model with various solar components
|   |   |-- vent_detailed   a model with detailed mechanical ventilation
|   |   |-- vent_simple     a model with simple mechanical ventilation
|   |   `-- wch             a model with web central heating
|   |-- pv_facade           research facility with PV embedded facade
|   |-- rad_room_heat       explicit representation of radiant ceiling heating
|   |-- trombe_wall          explicit representation of a Trombe-Michelle wall
|   `-- sunroom
|
```

2.2 Mixed language implementation issues

ESP-r is a mixed language environment. Essentially the data model and the decision about what to present to the user and how to interpret user actions is handled in the Fortran source. The C code in lib implements the directives about what to display and captures and passes back user actions (mouse movements and keystrokes). The goal is for the code in the application source folders to be substantially isolated from the underlying graphic API via the use of intermediate code in the lib folder. There are a few exceptions to this goal. Both Fortran and C offer the possibility to include or exclude blocks of code via so-called #ifdef statements. For example:

```
C IEEE callback (Solaris)
#ifdef SUN
#ifdef F90
      external SIGFPE_bps
      integer SIGFPE_bps
#endif
#endif
```

identifies a Solaris computer and Fortran 90 so that a floating point exception handler can be used. And

```
#ifdef GCC4
      call CTIME(ICTIME,ECTIME)
#else
      character*24 CTIME
      ECTIME = CTIME(ICTIME)
#endif
```

associates particular blocks of code with the GCC version 4 compiler. And

```
#ifdef OSI
      integer impxe,impye,iw,irpxe,irpye,inoe,ipflg,iuresp  ! for use with evwmenu
#else
      integer*8 impxe,impye,iw,irpxe,irpye,inoe,ipflg,iuresp  ! for use with evwmenu
#endif
```

defines the size of integer variables depending on a compiler and operating system combination. The code relies on very few #ifdef statements. There are also different library source files that are linked into executables to allow ESP-r to use either the X11 graphic primitives or the GTK graphic library primitives.

2.3 The src folder contents

The source code and databases are held in the src folder structure as shown below

```
src
|-- archive     documents for developers
|-- bin         command files (scripts) for developers and users
|-- bitmaps     images used by X11 version
|-- cetc        code from Natural Resources Canada
|   `-- h3kreports  XML report generation code
|       `-- xsl
|-- climate     climate data sets (ASCII versions)
|-- databases   materials, optics, constructions, plant components
|   `-- UK_NCM  patterns of occupancy used in UK national calculation method
|-- env         example Unix/Linux dot files
|-- esruaco     source code specific to the acoustics module
|-- esrub2e     source code specific to BEMS import module
```

```
|-- esrubld     source code for zone solver
|-- esrubps     source code for the multi-domain solver (bps)
|-- esruc2e     source code for air flow pressure coefficients
|-- esruclm     source code for the clm module
|-- esrucnv     source code for importing and exporting to 3rd party tools
|-- esrucom     common code used by various modules
|-- esructl     source code for global controllers
|-- esrudbm     source code for generic databases
|   '-- db      sample generic databases
|-- esrudfs     source code for the CFD solver and gridding setup
|   '-- Info
|-- esrue2r     source code for interface to Radiance (e2r)
|-- esrueco     source code for environmental impacts module (eco)
|-- esrugrd     source code for 2D and 3D gridding (grd)
|-- esruish     source code for shading and insolation pre-calculator (ish)
|-- esrumfs     source code for mass flow networks (mfs)
|-- esrumld     source code for mico-toxin assessments (mld)
|-- esrumrt     source code for surface-to-surface view factors (espvwf)
|-- esrunet     source code for an iconic network sketch module (net)
|-- esrupdb     source code for the plant template library manager (pdb)
|-- esrupdf     source code for electrical network manager
|-- esrupfs     source code for electrical network solver
|-- esruplt     source code for plant component solver
|-- esruprj     source code for the project manager module (prj)
|-- esrures     source code for the results analysis module (res)
|-- esrurun     source code for converting 3rd party descriptions into ESP-r models
|-- esruvew     source code for a hidden line wire-frame viewer
|-- esruvld     source code for validation and uncertainty studies
|-- include     header files for fortran and c code
|-- lib         library code and user interface code
|-- shocc       library of occupant preferences (placeholder)
|-- tutorial    holds contextual help text for many of the ESP-r modules
|-- training    holds exemplar models (see above)
|-- validation  holds models used to validate ESP-r
```

The use of a common code base for the ESP-r modules is demonstrated by looking at the contents of the esrubps (the ESP-r simulator) and esrucom folders:

```
ls esrubps
BC_data.F      Makefile         bmatsv.F  input.F   simcon.F
Lookup_data.F  SiteUtilities.F  bps.F     reslib.F  tdrecbps.F

ls esrucom
MultiYear_climate.F  e3dviews.F   emkcfgg.F    mcdbscn.F     scsys.F
bsimsel.F            ecasctl.F    emoist.F     nwkrewr.F     senrwl.F
c2fdum.F             ecdbscn.F    enetmisc.F   plelev.F      setup.F
cfdrw.F              econstr.F    enetrewr.F   pltcfg.F      sort.F
cfgrid.F             econtrol.F   eroper.F     psychro.F     spmisc.F
common3dv.F          edatabase.F  esgrid.F     rcdblist.F    startup.F
cread3dv.F           egeometry.F  esru_misc.F  readTrnsys.F  startup.c
ctlexp.F             egrid.F      esystem.F    rnor.F        tdfile.F
ctprob.F             egtgeom.F    fanger.F     rwipv.F
ctread.F             emfnetw.F    filelist.F   rwroam.F
dossupport.F         emkcfg.F     item.F       rwsbem.F
```

2.4 Exploration techniques

The numerical engine of ESP-r only has a few files in its folder. At compile time the instructions for building the module bps (esrubps/Makefile) are to use source files in esrucom (and other folders). The extent of the modular structure is further demonstrated by the number of references to esrucom/egeometry.F in the various modules:

```
grep -ni egeometry.F */Makefile

esrub2e/Makefile:33:egeometry.F:
esrubps/Makefile:184:egeometry.F:
esrucnv/Makefile:34:egeometry.F:
esrudbm/Makefile:34:egeometry.F:
esrudfs/Makefile:43:egeometry.F:
esrue2r/Makefile:62:egeometry.F:
esrueco/Makefile:59:egeometry.F:
esrugrd/Makefile:59:egeometry.F:
esruish/Makefile:59:egeometry.F:
```

```
esrumrt/Makefile:54:egeometry.F:
esrunet/Makefile:28:egeometry.F:
esruprj/Makefile:95:egeometry.F:
esrures/Makefile:77:egeometry.F:
esrurun/Makefile:144:egeometry.F:
```

So what if we wanted to know where in the source distribution the subroutine georead from the above egeometry.F is used. The output of the command gives the folder and file name as well as the position in the file where georead is called and places where it is mentioned in comments (lines beginning with a 'C').

```
grep -ni georead */*.F

esruaco/acoesp.F:196:      call georead(ITMP,LGEOM(ICOMP),ICOMP,0,iuout,IER)
esruaco/acoesp.F:564:             call georead(ITMP,LGEOM(nZnog(J)),nZnog(J),0,iuout,IER)
esruaco/acoesp.F:747:             call georead(ITMP,LGEOM(nZnog(k)),nZnog(k),0,iuout,IER)
esruaco/acoesp.F:997:             call georead(ITMP,LGEOM(nZnog(k)),nZnog(k),0,iuout,IER)
esrubld/bcfunc.F:5241:        call georead(ifil+1,lgeom(icomp),icomp,0,itu,ier)
esrubld/bcfunc.F:6860:        call georead(iunit,lgeom(ICMP),ICMP,0,itu,ier)
esrubld/bcfunc.F:6869:        call georead(iunit,lgeom(ICMP),ICMP,0,itu,ier)
esrubld/casual.F:1071:        call georead(iunit,lgeom(icomp),icomp,0,itu,ier)
esrubps/bmatsv.F:1187:                  call georead(igu,LGEOM(icomp),icomp,1,iuout,IER)
esrubps/input.F:1073:         call georead(IUNIT,LGEOM(IZ),IZ,1,ITU,IER)
esrubps/input.F:1927:         call georead(IFIL+1,LGEOM(IZ),IZ,0,itru,IER)
esrucnv/e2vdxf.F:78:          call georead(ITA1,LGEOM(newfoc),newfoc,1,IUOUT,IER)
esrucnv/e2vdxf.F:143:         call georead(ITA1,LGEOM(newfoc),newfoc,1,IUOUT,IER)
esrucnv/e2vdxf.F:413:         call georead(ITA1,LGEOM(mz),mz,1,IUOUT,IER)
esrucnv/e2vdxf.F:451:         call georead(ITA1,LGEOM(mz),mz,1,IUOUT,IER)
esrucnv/e2vdxf.F:595:         call georead(ITA1,LGEOM(newfoc),newfoc,1,IUOUT,IER)
esrucnv/e2vdxf.F:646:         call georead(ITA1,LGEOM(newfoc),newfoc,1,IUOUT,IER)
esrucnv/e2vrml.F:186:         call georead(ITA1,LGEOM(newfoc),newfoc,1,IUOUT,IER)
esrucom/common3dv.F:688:          call georead(IUF,LGEOM(ifoc),ifoc,1,iuout,ier)
esrucom/common3dv.F:1322:          call georead(IUF,LGEOM(IVALS(IZ)),IVALS(IZ),1,iuout,IER)
esrucom/egeometry.F:26:C  GEOREAD: Reads V1.1 zone geometry data as ASCII strings, with or without
esrucom/egeometry.F:182:C the code should use georead to scan the file so return
esrucom/egeometry.F:1192:C ******************** GEOREAD
esrucom/egeometry.F:1193:C GEOREAD reads V1.1 zone geometry data (LGEOMF) from a user-constructed data
esrucom/egeometry.F:1213:      SUBROUTINE GEOREAD(IUNIT,LGEOMF,ICOMP,IR,ITRU,IER)

 . . .
/egeometry.F:2379: 1002 write(outs,'(3a)') 'GEOREAD: conversion error in...',
esrucom/egeometry.F:4934:C using georead or egomin and pass across into the appropriate array.
esrucom/egeometry.F:4942:        call georead(IFIL+1,LGEOM(ICOMP),ICOMP,1,iuout,ier)
esrucom/esystem.F:2493:          call georead(iuf,LGEOM(ICOMP),ICOMP,1,itru,ier)
esrucom/esystem.F:2741:            call georead(iuf,LGEOM(ICOMP),ICOMP,1,itru,ier)
esrucom/plelev.F:135:         call georead(IUF,LGEOM(newfoc),newfoc,0,IUOUT,IER)
esrucom/plelev.F:281:           call georead(IUF,LGEOM(newfoc),newfoc,0,IUOUT,IER)
esrue2r/e2r.F:1380:           call georead(ITA1,LGEOM(ICOMP),ICOMP,1,IUOUT,IER)
esrue2r/e2rform.F:371:        call georead(ITA1,LGEOM(ICOMP),ICOMP,1,IUOUT,IER)
esruprj/bnlthp.F:119:         call georead(IUNIT,LGEOM(ICOMP),ICOMP,1,IUOUT,IER)
esruprj/e2eplus.F:1582:         call georead(ITA1,LGEOM(mz),mz,1,IUOUT,IER)
esruprj/edcasctl.F:188:       call georead(IUF,LGEOM(ICOMP),ICOMP,0,ITRU,IER)
esruprj/edcfd.F:871:             call georead(IFIL+1,LGEOM(nznog(1)),nznog(1),1,IUOUT,IER)
esruprj/edcfd.F:1626:             call georead(IFIL+1,LGEOM(nznog(1)),nznog(1),1,IUOUT,IER)
esruprj/edcon.F:157:       call georead(IFIL+1,LGEOM(ICOMP),ICOMP,1,IUOUT,IER)
esruprj/edcondb.F:1653:              call georead(IFIL+1,LGEOM(ICOMP),ICOMP,1,iuout,ier)
esruprj/edcondb.F:2955:            call georead(IUF,LTMP,IZ,1,iuout,IER)
esruprj/edgeo.F:466:              call georead(IFIL+2,LGEOM(IC2(ioc)),IC2(ioc),0,iuout,IER)
esruprj/edgeo.F:1129:                  call georead(IFIL+2,LGEOM(IC2(ioc)),IC2(ioc),
esruprj/edmrt.F:122:       call georead(IUF,LGEOM(ICOMP),ICOMP,1,IUOUT,IER)
esruprj/edobs.F:778:         call georead(IUF,LGEOM(ICOMP),ICOMP,0,ITRU,IER)
esruprj/edonecon.F:111:       call georead(IUO,LGEOM(IC1(IFOC)),IC1(IFOC),0,iuout,IER)
esruprj/edonecon.F:418:         call georead(IUO,LGEOM(IC1(IFOC)),IC1(IFOC),0,iuout,IER)
esruprj/edonecon.F:458:         call georead(IUO,LGEOM(IC2(IFOC)),IC2(IFOC),0,iuout,IER)
esruprj/edspmtl.F:628:       call georead(IFIL+1,LGEOM(IZ),IZ,1,iuout,IER)
esruprj/edspmtl.F:677:       call georead(IFIL+1,LGEOM(IZ),IZ,1,iuout,IER)
esruprj/edspmtl.F:823:         call georead(IFIL+1,LGEOM(IZ),IZ,1,iuout,IER)
esruprj/edspmtl.F:857:         call georead(IFIL+1,LGEOM(IZ),IZ,1,iuout,IER)
esruprj/edtopol.F:581:             call georead(IFIL+1,LGEOM(IZ),IZ,0,itru,IER)
esruprj/edtopol.F:635:             call georead(IFIL+1,LGEOM(IZ),IZ,0,itru,IER)
esruprj/edzone.F:623:         call georead(IFIL+1,LGEOM(IC),IC,1,iuout,IER)
esruprj/edzone.F:1291:           call georead(IFIL+1,LGEOM(IX),IX,1,iuout,IER)
esruprj/edzone.F:1390:       call georead(IFIL+1,LGEOM(IW2),IW2,1,iuout,IER)
esruprj/edzone.F:1571:         call georead(IFIL+1,LGEOM(ir),ir,1,IUOUT,IER)
esruprj/emeta.F:841:           call georead(IFIL+2,LGEOM(ICOMP),ICOMP,1,iuout,IER)
esruprj/emeta.F:929:           call georead(IFIL+2,LGEOM(ICOMP),ICOMP,1,iuout,IER)
esruprj/folders.F:1601:        call georead(IUF,LGEOM(IZ),IZ,1,iuout,IER)
```

```
esruprj/hcfmk.F:254:      call georead(IUF,LGEOM(ICOMP),ICOMP,0,iuout,IER)
esruprj/insert.F:161:             call georead(IFIL+2,LGEOM(iotherzone),iotherzone,0,
esruprj/insert.F:202:             call georead(IFIL+2,LGEOM(iotherzone),iotherzone,0,
esruprj/mfprb1.F:827:           call georead(IFIL+1,LGEOM(IZ),IZ,0,ITRU,IER)
esruprj/mfprb1.F:1701:        call georead(IFIL+1,LGEOM(IZ),IZ,1,iuout,IER)
esruprj/mksbem.F:3519:          CALL GEOREAD(IUF,LGEOM(ICOMP),ICOMP,1,IUOUT,IER)
esruprj/mksbem.F:4352:          call georead(IUF,LGEOM(ICOMP),ICOMP,1,iuout,IER)
esruprj/prescoef.F:579:          call georead(IUF,LGEOM(IC),IC,1,iuout,IER)
esruprj/prescoef.F:702:            call georead(IUF,LGEOM(IZU),IZU,1,iuout,IER)
esruprj/prj.F:779:            call georead(IFIL+1,LGEOM(ICOMP),ICOMP,1,iuout,ier)
esruprj/prj.F:866:            call georead(IFIL+1,LGEOM(index),index,1,iuout,IER)
esruprj/prjqa.F:838:            call georead(IUF,LTMP,IVALS(IZ),1,iuout,IER)
esruprj/prjqa.F:1696:          call georead(IUF,LTMP,IVALS(IZ),1,iuout,IER)
esruprj/read3dv.F:196:        call georead(IFIL+1,LGEOM(nznog(IZ)),nznog(IZ),1,IUOUT,IER5)
esrures/enerbs.F:65:      call georead(IUNIT,LGEOM(IZONE),IZONE,0,IUOUT,IER)
esrures/enerbz.F:66:      call georead(iunit,LGEOM(IZONE),IZONE,0,iuout,ier)
esrures/fabcon.F:927:      call georead(IUNITF,LGEOM(ICOMP),ICOMP,0,IUOUT,IER)
esrures/fabtmp.F:53:      call georead(IUNITF,LGEOM(ICOMP),ICOMP,0,IUOUT,IER)
esrures/moget.F:4220:        call georead(IUNIT,LGEOM(IZ),IZ,0,IUOUT,IER)
esrures/replsim.F:128:        call georead(IUNIT,LGEOM(J),J,1,IUOUT,IER)
esrures/stats.F:2145:        call georead(iunit,LGEOM(izone),izone,0,iuout,ier)
esruvld/anlytc.F:909:      call georead(IUnit,LGEOM(IComp),IComp,1,IUOUT,IER)
```

Most of the modules of ESP-r make use of georead so scan in the definition of a zones coordinates and surface attributes. Indeed, some modules do this many times (an artifact of how information is held within the data structure). From the list above we see there is one instance of the phrase *subroutine egomin* so that also tells us where the subroutine code is.

If, instead we wanted to know how many places update zone geometry files we would want to search for instances of the subroutine geowrite:

```
grep -ni geowrite */*.F
```

```
esrucom/egeometry.F:28:C GEOWRITE: Write a geometry file (GEN V1.1 type) based on infor-
esrucom/egeometry.F:2399:C ************* GEOWRITE
esrucom/egeometry.F:2400:C GEOWRITE to write a geometry file (GEN V1.1 type) based on infor-
esrucom/egeometry.F:2415:      SUBROUTINE GEOWRITE(IFILG,GENFIL,ICOMP,ITRU,iwf,IER)
esruprj/clickonbitmap.F:1809:                call geowrite(IFIL+2,LGEOM(ICOMP),ICOMP,iuout,3,IER)
esruprj/clickonbitmap.F:2210:              call geowrite(IFIL+2,LGEOM(ICOMP),ICOMP,iuout,3,IER)
esruprj/edcfg.F:1928:            call geowrite(iuf,LGEOM(NCOMP),NCOMP,ITRU,3,IER)
esruprj/edcfg.F:1988:              call geowrite(iuf,LGEOM(NCOMP),NCOMP,ITRU,3,IER)
esruprj/edcfg.F:4341:            call geowrite(IUF,LTMP,IICOMP,ITRU,3,IER)
esruprj/edcfg.F:4427:                call geowrite(IUF,LTMP,IICOMP,ITRU,3,IER)
esruprj/edcfg.F:4531:            call geowrite(IUF,LTMP,IICOMP,ITRU,3,IER)
esruprj/edcfg.F:4651:                call geowrite(IUF,LTMP,IICOMP,ITRU,3,IER)
esruprj/edcon.F:696:           call geowrite(IUF,LGEOM(ICOMP),ICOMP,ITRU,3,IER)
esruprj/edcondb.F:2990:              call geowrite(IUF,LTMP,IZ,ITRU,3,IER)
esruprj/edgeo.F:486:              call geowrite(IFIL+2,LGEOM(IC2(ioc)),IC2(ioc),
esruprj/edgeo.F:636:             call geowrite(IFIL+2,LTMP,ICOMP,iuout,3,IER)
esruprj/edgeo.F:807:             call geowrite(IFIL+2,LTMP,ICOMP,iuout,3,IER)
esruprj/edobs.F:30:C << to keep the data refreshed so geowrite calls can be made.
esruprj/edobs.F:242:          call geowrite(IUF,LGEOM(ICOMP),ICOMP,iuout,3,IER)
esruprj/edobs.F:549:              call geowrite(IUF,LGEOM(ICOMP),ICOMP,iuout,3,IER)
esruprj/edonecon.F:434:          call geowrite(IUO,LGEOM(IC1(IFOC)),IC1(IFOC),iuout,3,IER)
esruprj/edonecon.F:471:          call geowrite(IUO,LGEOM(IC2(IFOC)),IC2(IFOC),iuout,3,IER)
esruprj/edtopol.F:1666:            call geowrite(IFIL+2,LGEOM(IZ),IZ,iuout,3,IER)
esruprj/edtopol.F:2980:                call geowrite(IUF,LTMP,IVALS(IZ),iuout,3,IER)
esruprj/edtopol.F:3058:              call geowrite(IUF,LTMP,IZ,iuout,3,IER)
esruprj/edtopol.F:3205:              call geowrite(IUF,LTMP,IVALS(IZ),ITRU,3,IER)
esruprj/edzone.F:461:        call geowrite(IFIL+2,LGEOM(ICOMP),ICOMP,iuout,3,IER)
esruprj/edzone.F:768:            call geowrite(IFIL+1,LGEOM(ic),ic,iuout,3,IER)
esruprj/edzone.F:1321:               call geowrite(IFIL+2,LGEOM(IX),IX,iuout,3,IER)
esruprj/emeta.F:795:      NCOMP=NCOMP+1  ! Temporarily update NCOMP for geowrite use.
esruprj/emeta.F:800:      call geowrite(IFIL+2,LGEOM(ICOMP),ICOMP,iuout,3,IER)
esruprj/emeta.F:919:            call geowrite(IFIL+2,LGEOM(ICOMP),ICOMP,iuout,3,IER)
esruprj/emeta.F:1002:            call geowrite(IFIL+2,LGEOM(ICOMP),ICOMP,iuout,3,IER)
esruprj/emeta.F:1029:          call geowrite(IFIL+2,LGEOM(ICOMP),ICOMP,iuout,3,IER)
esruprj/folders.F:1617:            call geowrite(IUF,LGEOM(IZ),IZ,iuout,3,IER)
esruprj/insert.F:178:               call geowrite(IFIL+2,LGEOM(iotherzone),iotherzone,
esruprj/insert.F:217:               call geowrite(IFIL+2,LGEOM(iotherzone),iotherzone,
esruprj/insert.F:587:            call geowrite(IFIL+1,LGEOM(ICOMP),ICOMP,iuout,3,IER)
esruprj/insert.F:1004:              call geowrite(IFIL+1,LGEOM(ICOMP),ICOMP,iuout,3,IER)
esruprj/insert.F:1023:              call geowrite(IFIL+1,LGEOM(ICOMP),ICOMP,iuout,3,IER)
esruprj/mksbem.F:3573:         CALL GEOWRITE(IUF,LGEOM(ICOMP),ICOMP,ITRU,3,IER)
esruprj/mksbem.F:4388:         call geowrite(IUF,LGEOM(ICOMP),ICOMP,ITRU,3,IER)
esruprj/prj.F:4345:              call geowrite(IUF,LGEOM(ICOMP),IComp,IUOUT,3,IER)
```

```
esruprj/prj.F:4620:                  call geowrite(IUF,LGEOM(ICOMP),IComp,IUOUT,3,IER)
esruprj/prj.F:5020:                call geowrite(IUNIT,LGEOM(ICOMP),ICOMP,ITRU,3,IER)
esruprj/prjqa.F:2684:                 call geowrite(IFIL+2,LGEOM(iz),iz,iuout,3,IER)
esruprj/prjqa.F:2704:                 call geowrite(IFIL+2,LGEOM(iz),iz,iuout,3,IER)
esruprj/prjqa.F:2716:               call geowrite(IFIL+2,LGEOM(iz),iz,iuout,3,IER)
esruprj/prjqa.F:2769:              call geowrite(IFIL+2,LGEOM(iz),iz,iuout,3,IER)
esruprj/prjqa.F:2778:                call geowrite(IFIL+2,LGEOM(iz),iz,iuout,3,IER)
esruvld/anlytc.F:1034:      call geowrite(IUF,LGEOM(ICOMP),IComp,IUOUT,3,IER)
```

There are fewer instances of geowrite and almost all of them are in the esruprj folder which is associated with the module prj. Prj is the ESP-r module primarily responsible for maintaining the composition of user models and controlling access to the simulator and results analysis modules.

Enquiries about the data structures used by ESP-r can be demonstrated by searching for two different variables - one which is held in an include file and one where common blocks are defined in the body of the source code. Within the include/geometry.h file is the definition of a common blocks named g9 and g4:

```
C G9 holds information on children of a surface and its parent.
      integer nbchild   ! how many children (up to 4) for each connection
      integer nbgchild  ! how many grand children (up to 4) for each connection
      integer ichild    ! list of children for each connection
      integer igchild   ! list of grand children for each connection
      integer iparent   ! parent surface connection (zero is no parent)
      integer igparent  ! grandparent surface connection (zero is no grandparent)
      common/G9/nbchild(MCON),nbgchild(MCON),ichild(MCON,4),
     &          igchild(MCON,4),iparent(MCON),igparent(MCON)

C Default solar distribution and shading directives.
      integer ndp,idpn
      common/g4/ndp(MCOM),idpn(MCOM,3)
```

The documentation of g9 is extensive while the two variables in common block g4 are rather terse (in terms of their names and documention) so a search for where ndp and idpn are used might provide additional information:

```
grep -ni idpn */*.F

esrubld/solar.F:1735:              inssur(1)=IDPN(ICOMP,1)
esrubld/solar.F:1736:              inssur(2)=IDPN(ICOMP,2)
esrubld/solar.F:1737:              inssur(3)=IDPN(ICOMP,3)
esrucnv/zipcnv.F:278:C IDPN defines the default plane numbers.
esrucnv/zipcnv.F:279:      read(ioin,*) NDP(IC),(IDPN(IC,i),i=1,3)
esrucnv/zipcnv.F:336:        write(iotmp2,'(4i4,a)')NDP(IC),IDPN(IC,1),IDPN(IC,2),IDPN(IC,3),
esrucom/egeometry.F:527:C IDPN defines the default plane numbers.
esrucom/egeometry.F:534:      IDPN(ICOMP,1)=IV
esrucom/egeometry.F:536:      IDPN(ICOMP,2)=IV
esrucom/egeometry.F:538:      IDPN(ICOMP,3)=IV
esrucom/egeometry.F:544:          if(IDPN(ICOMP,I).NE.-1.AND.IDPN(ICOMP,I).NE.0)then
esrucom/egeometry.F:545:            if(IDPN(ICOMP,I).LT.IX.OR.IDPN(ICOMP,I).GT.NSUR)then
esrucom/egeometry.F:1093:     &  (IDPN(ICOMP,J),J=1,3),'# default insolation distribution'
 . . .
esrucom/esru_misc.F:457:        IDPN(IX,1)=0
esrucom/esru_misc.F:458:        IDPN(IX,2)=0
esrucom/esru_misc.F:459:        IDPN(IX,3)=0
esrucom/esru_misc.F:2279:      COMMON/SG4/NDP1,IDPN1(3)
esrucom/esru_misc.F:2296:      IDPN1(1)=IDPN(ICOMP,1)
esrucom/esru_misc.F:2297:      IDPN1(2)=IDPN(ICOMP,2)
esrucom/esru_misc.F:2298:      IDPN1(3)=IDPN(ICOMP,3)
 . . .
esruprj/clickonbitmap.F:2080:            IDPN(ICOMP,1)=0
esruprj/clickonbitmap.F:2081:            IDPN(ICOMP,2)=0
esruprj/clickonbitmap.F:2082:            IDPN(ICOMP,3)=0
esruprj/edgeo.F:2413:      if(IDPN(ICOMP,1).gt.0)then
esruprj/edgeo.F:2414:        ioc1=IZSTOCN(icomp,IDPN(ICOMP,1))
esruprj/edgeo.F:2416:      if(IDPN(ICOMP,2).gt.0)then
esruprj/edgeo.F:2417:        ioc2=IZSTOCN(icomp,IDPN(ICOMP,2))
esruprj/edgeo.F:2601:          IDPN(ICOMP,1)=IS
esruprj/edgeo.F:2602:          IDPN(ICOMP,2)=0
 . . .
esruprj/edgeo.F:7751:C IDPN defines the default plane numbers.
esruprj/edgeo.F:7760:      IDPN(ICOMP,1)=IV
esruprj/edgeo.F:7762:      IDPN(ICOMP,2)=IV
esruprj/edgeo.F:7764:      IDPN(ICOMP,3)=IV
```

```
esruprj/edgeo.F:7770:          if(IDPN(ICOMP,I).NE.-1.AND.IDPN(ICOMP,I).NE.0)then
esruprj/edgeo.F:7771:            if(IDPN(ICOMP,I).LT.IX.OR.IDPN(ICOMP,I).GT.NSUR)then
esruprj/edzone.F:443:      IDPN(ICOMP,1)=0
esruprj/edzone.F:444:      IDPN(ICOMP,2)=0
esruprj/edzone.F:445:      IDPN(ICOMP,3)=0
esruprj/edzone.F:1549:        IDPN(IX,1)=IDPN(IX+1,1)
esruprj/edzone.F:1550:        IDPN(IX,2)=IDPN(IX+1,2)
esruprj/edzone.F:1551:        IDPN(IX,3)=IDPN(IX+1,3)
```

Because the common block is defined in geometry.h the source code blocks only contain statments that use the variable. For example in solar.F IDPN is copied to another variable INSSUR. IDPN is found on the left side of equations in egeometry.F (related to file reading) and esru_misc.F (related to clearing data structures) and in edgeo.F IDPN is linked to variables for user interactions.

This pattern of coding presumes that you will know to look for variables in include files if there is no documentation within the source code block.

An example of a common block that is defined within the code blocks is COMMON/MFLOW7/LVALCM(MCMV) where LVALCM is a string variable.

```
grep -ni LVALCM */*.F

esrucom/emfnetw.F:73:C     LVALCM - short description of each valid component type
esrucom/emfnetw.F:154:      COMMON/MFLOW7/LVALCM(MCMV)
esrucom/emfnetw.F:164:      CHARACTER LVALCM*60,LTPCMP*60,CMNAM*12,NDNAM*12
esrucom/emfnetw.F:295:   45   LTPCMP(ICMP)=LVALCM(IC)
esrucom/nwkrewr.F:783:      COMMON/MFLOW7/LVALCM(MCMV)
esrucom/nwkrewr.F:808:      character NWICNTXT*72,LVALCM*60
esrucom/nwkrewr.F:912:   45       LTPCMP(NCMP)=LVALCM(IC)
esrudfs/cfdat.F:110:      COMMON/MFLOW7/LVALCM(MCMV)
esrudfs/cfdat.F:158:      CHARACTER LVALCM*60,LTPCMP*60,CMNAM*12,NDNAM*12,outs*124
esrudfs/cfdat.F:808:        LTPCMP(NCMP)=LVALCM(22)
esrumfs/mfcdat.F:27:      COMMON/MFLOW7/LVALCM(MCMV)
esrumfs/mfcdat.F:29:      CHARACTER*60 LVALCM
esrumfs/mfcdat.F:33:      LVALCM(1)=
 . . .
/mfprb1.F:1332:      COMMON/MFLOW7/LVALCM(MCMV)
esruprj/mfprb1.F:1340:      CHARACTER LVALCM*60,LTPCMP*60,CMNAM*12,CMPID*12, MOD*1
esruprj/mfprb1.F:1381:          WRITE(clist(ic),'(I4,A,A)') IVALCM(IC),' : ',LVALCM(IC)
esruprj/mfprb1.F:1396:          LTPCMP(IFCMP)=LVALCM(IC)
esruprj/mfprb1.F:2098:      COMMON/MFLOW7/LVALCM(MCMV)
esruprj/mfprb1.F:2139:      CHARACTER LVALCM*60,LTPCMP*60,CMNAM*12
esruprj/mfprb1.F:2391:        LTPCMP(NCMP)=LVALCM(7)
 . . .
esruprj/mfprb1.F:2701:                    LTPCMP(NCMP)=LVALCM(7)
esruprj/mfprb1.F:2744:                  LTPCMP(NCMP)=LVALCM(7)
```

In this case the basic rule of ESP-r development has been followed and there is at least one definition of the variable in the place where it is scanned in from file (emfnetw.F). Other common blocks and variables may be documented in multiple places, but typically once per source file.

And there are also legacy blocks of code which are substantially devoid of documentation and even some common blocks for obscure facilities which are short on documentation. Are they obscure because they are poorly documented - probably. Is it less efficient to work with undocumented variables - absolutely!

Currently there is an evolution of coding practices to move common block definitions into include files and regularize their documentation and explicitly typing each variable.

Searching tasks are mostly related to finding patterns. The tool **grep** used in the above examples is a key tool for pattern searches. It is available on all of the computing platforms used for ESP-r development work (on Windows it comes as part of the MSYS environment along with other useful tools such as **find**.

Further details about the source files and the API of ESP-r can be found in the document *Structure of the ESP-r Source Code Archive* (January 2008). The raw troff document is held in the source code distribition in the *archive* folder.

3 Supported Platforms and Development Environments

ESP-r was initially a suite of tools running on Sun workstations and then with the advent of Linux running on lower cost personal computers the code was adapted to also run on Linux. There are a few lines of code which required adaptation for Solaris and Linux platforms and there are almost no differences in user interactions and in administrative tasks.

ESP-r implicitly assumes a range of operating system services and file protections. For example, that corporate databases and example models are held in folders where normal users can read but not overwrite such files. On other computing platforms such protections are either enforced in a different way or absent. Thus a novice user can redefine the conductivity of steel in a corporate database and thus cause many other models, even models used by other people on that machine to alter their predictions.

Note that ESP-r assumes the computer environment is using a USA or UK locale and that real numbers use a period as a decimal point and that a comma, tab or space is a separator between data. Use of a locale which uses a comma for a decimal point will cause extensive corruption of ESP-r models.

There is also a restriction that names of entities use an ASCII character set rather than extended character set. These dependencies are related to the underlying Fortran source code read and write statements. ESP-r has been observed to have problems with some Asian keyboards and locales.

3.1 Compilers

Solaris and Enterprise Linux computers support the Sun Studio F90/C/C++ compiler suite. Other platforms rely on the GNU compiler collection (versions 4.1 or newer). The former is particularly useful for development work as Sun Studio supports IEEE floating point exceptions (e.g. divide by zero) and array bounds checking (e.g. asking for the 12th value of an array of size 10).

Note that recent Linux distributions tend to have version 4.1/2/3/4 of the GNU compilers and ESP-r currently has very few sensitivities to which version of the compiler is used. ESP-r can not be compiled with F77 compilers.

There is also a general issue with 64 bit computers - there are slight differences in predictions and occasional risk of some graphical displays are incorrect or graphic tasks causing program crashes. ESP-r is currently a bit more robust when running on 32 bit computers.

3.2 Linux

For many groups, Linux computers are the preferred platform for production simulations and for high levels of security. Corporate databases can be protected from casual corruption if care is taken when installing ESP-r and in setting folder permissions.

Linux supports both the X11 and GTK interface as well as text mode operation (for automated work). On Linux machine with substantial memory some data recovery tasks are speeded up because simulation results files tend to be scanned from memory rather than from the binary file.

Information about setting up several version of Ubuntu are included in the source distribution in the folder src/manual/OS/Linux. There are also hints on the ESRU web download page.

For those interested in enterprise class environments such as Red Hat Enterprise Linux and its clone CentOS, ESP-r has been deployed for production tasks. Again, there are a few issues with 64 bit computers under which GTK versions should be considered work-in-progress. X11 and text-only use on enterprise machines is the more common approach.

3.3 Solaris

This was the original platform for ESP-r and was known for the numerical robustness of the Sun Studio compilers and the extreme security of the operating system. The Sun Studio compilers are able to identify some numerical issues which are not possible with the GNU compilers. Few developers and practitioners use Sun machines however the Sun Studio compilers freely available for enterprise class Linux boxes those needing access to the numerical robustness has additional choices.

3.4 OSX

With the advent of OSX, Apple computers offer many of the same compilers and low level operating system services as Linux and so it has been possible to port ESP-r to Apple computers using PPC or Intel chips. There are a few minor differences in operating system services (file name case sensitivity is incomplete and users folders are found in /Users rather than /home. As with Linux and Unix there are no A/B/C/D drive letters in OSX.

In terms of use the interface is the same as is offered on Linux. Because it does not follow the full OSX look and feel rules, some users find this confusing. The reliance on the X window environment also requires additional steps in setting up OSX machines.

OSX supports the GNU compiler collection as well as X11 libraries and source code conventions. OSX 10.5 and 10.6 support the X11 interface compiled either to 32bit or 64bit while the GTK interface must be compiled for 64bit because the GTK libraries are only available in 64bit. For development work it is necessary to install the so-called *fink or MacPorts* facilities as well as X11 support. Information about setting up an OSX machine is included in the source distribution in the folder src/manual/OS/Apple. There are also hints on the ESRU web download page.

3.5 Cygwin under Windows

Because of the differences in compilers and operating system services it took some time to realize a version of ESP-r that runs natively on Windows computers. The initial approach to ESP-r running on Windows computers was to use an emulation environment called Cygwin. Cygwin provides the compilation environment required by ESP-r as well as translating many operating system requests and providing a similar command line interpreter (shell scripting) as one would find on a Linux machine. The same automation scripts that work on Linux tend to work under Cygwin. Such scripts are different in syntax and tend to perform more complex operations in comparison to DOS batch files because they are based on an extended command language. Cygwin can also host X11 and GTK versions of ESP-r whereas the Native Windows version is compiled with the GTK or a pure-text interface.

Again there few code differences required for development and use of ESP-r on Cygwin. In terms of user experience, ESP-r thinks it is running on a Linux box and the same user interactions apply.

Cygwin supports the usual GNU compiler collection and development tasks are essentially the same as on Linux. File permissions are less strict than Linux and thus care should be exercised to avoid overwriting files that ESP-r assumes have strict permissions. Information about setting up Cygwin is included in the source distribution in the folder src/manual/OS/Cygwin.

3.6 Native Windows

The native Windows version of ESP-r is an *almost complete* port of the facilities available on other computer platforms. This version works on Windows XP and W7 computers and some users report stability under Vista. There has been little or no testing of 64-bit versions of Windows.

The underlying graphic libraries currently restrict some functions (this is work-in-progress). The major differences are found in the facilities provided by the operating system and in the layout and conventions of the file system.

Development for Native Windows currently requires the MSYS collection of tools in addition to MinGW, a port of the GNU compiler collection. Information about setting up MSYS is included in the source distribution in the folder src/manual/OS/Native_windows.

3.7 File names & character sets

ESP-r currently has a limited ability to cope with spaces in file names or non-ASCII characters. In some Asian locations new user accounts with simpler login names often will improve the operation of ESP-r. ESP-r also has limits on the length of file names and paths. These limit where ESP-r can be installed as well as how deeply nested model folders can be before file names become truncated. For this reason, pre-compiled versions of ESP-r are designed to be in `C:\Esru\esp-r` rather than in `C:\Program Files\Esru`. ESP-r models work better in `C:\Esru\Models` rather than `C:\Documents and Settings\Fred\My Current Models\`

4 Installing ESP-r

Many users of ESP-r will acquire a working version of ESP-r via the *download* page of the ESRU web site. For those who require the most current state of ESP-r or a non-standard version of ESP-r this is done via the source repository used by the development community. This also allows the ESP-r community to fulfill the requirements of the GNU public license to distribute the source code as well as ensure that those who alter the code are also able to conform to the license requirements to return their changes as a contribution to the ESp-r community.

4.1 Preparation

Before you proceed to setup a set of folders to hold the ESP-r source code and the destination folders for the compiled executables and databases consider the purpose of your ESP-r installation. A single user on a laptop who may wish to compile ESP-r but who does not anticipate much development work has considerable flexibility. There are only a few of the subversion commands required to get access to the current official distribution or the current development branch. The ESP-r Install script provides most of the choices needed. As covered in subsequent sections there are some additional tasks needed to ensure that users accounts get access to the functionality of ESP-r.

The source and the destination folders for casual developers on might reside in their own area of the computer e.g. /home/fred/Src/ with executables in /home/fred/esp_fred. Such an install would typically not require administrator access. For those who would rather install to more traditional locations of /usr/esru or /opt/esru) administrator access is required. Unix/Linux/OSX often place optional (user supplied) software in /opt. There is a tradition within the Energy Systems Research Unit of the University of Strathclyde to place ESP-r distributions in either /usr/esru or /home/esru and this is still observed by a number of developers. Indeed the development test regime retains several dependencies on the existence of a folder /usr/esru. At some point in the future this requirement in the testing process will be relaxed.

Someone who will be an active developer may have a number of source folders e.g. /home/fred/cvsdude/development_branch, /home/fred/cvsdude/fred and install the standard development_branch version to /usr/esru and their own working version to /home/fred/esru_fred. The computer will need to be setup so that the PATH environment variable can point to either of these versions of ESP-r. One technique is to create a folder /home/fred/bin and create a link (in Unix/Linux/OSX/Cygwin a link is a special type of file which is actually a pointer to a real file in another location) to each of the relevant ESP-r executables and ensure that /home/fred/bin is EARLY in the PATH definition. The script below (assuming the user is named fred) will define a link to each ESP-r executable if it is passed the full path to the executables:

```
#!/bin/csh
# create a link in /home/fred/bin to current esp-r executables
echo "use is"
echo "bin_link_to [folder with executables]"
foreach i ( aco bps c2e cfg clm dfs e2r ecnv eco grd ish mfs mld mrt pdb prj res viewer )
  if ( -f $1/$i )then
    if ( -f /home/fred/bin/$i ) rm -f /home/fred/bin/$i
    ln -s $1/$i /home/fred/bin/$i
    echo $i " is now pointing to "$1/$i
  endif
end
```

Note the first line of the script invokes the **csh** command interpretor to process the commands (the *foreach* syntax is specific to that interpretor). You can find a copy of this **link_to** script in the source distribution bin folder. Most users on Linux or OSX will default to the **bash** command interpretor. If you want to know more about command interpretors there are any number of books about Linux which discuss available command interpretors.

The result of running the above script for a set of ESP-r executables in the folder */Users/jon/esru_jwhgfort/esp-r/bin* would look like:

```
lrwxr-xr-x    1 jon      staff          38 Feb 12 11:39 aco -> /Users/jon/esru_jwhgfort/esp-r/bin/aco
lrwxr-xr-x    1 jon      staff          38 Feb 12 11:39 bps -> /Users/jon/esru_jwhgfort/esp-r/bin/bps
lrwxr-xr-x    1 jon      staff          38 Feb 12 11:39 c2e -> /Users/jon/esru_jwhgfort/esp-r/bin/c2e
lrwxr-xr-x    1 jon      staff          38 Feb 12 11:39 clm -> /Users/jon/esru_jwhgfort/esp-r/bin/clm
lrwxr-xr-x    1 jon      staff          38 Feb 12 11:39 dfs -> /Users/jon/esru_jwhgfort/esp-r/bin/dfs
lrwxr-xr-x    1 jon      staff          38 Feb 12 11:39 e2r -> /Users/jon/esru_jwhgfort/esp-r/bin/e2r
lrwxr-xr-x    1 jon      staff          39 Feb 12 11:39 ecnv -> /Users/jon/esru_jwhgfort/esp-r/bin/ecnv
lrwxr-xr-x    1 jon      staff          38 Feb 12 11:39 eco -> /Users/jon/esru_jwhgfort/esp-r/bin/eco
lrwxr-xr-x    1 jon      staff          38 Feb 12 11:39 grd -> /Users/jon/esru_jwhgfort/esp-r/bin/grd
```

```
lrwxr-xr-x    1 jon      staff          38 Feb 12 11:39 ish -> /Users/jon/esru_jwhgfort/esp-r/bin/ish
-rwx------    1 jon      staff         443 Dec 23  2008 link_to
lrwxr-xr-x    1 jon      staff          38 Feb 12 11:39 mfs -> /Users/jon/esru_jwhgfort/esp-r/bin/mfs
lrwxr-xr-x    1 jon      staff          38 Feb 12 11:39 mld -> /Users/jon/esru_jwhgfort/esp-r/bin/mld
lrwxr-xr-x    1 jon      staff          38 Feb 12 11:39 pdb -> /Users/jon/esru_jwhgfort/esp-r/bin/pdb
lrwxr-xr-x    1 jon      staff          38 Feb 12 11:39 prj -> /Users/jon/esru_jwhgfort/esp-r/bin/prj
lrwxr-xr-x    1 jon      staff          38 Feb 12 11:39 res -> /Users/jon/esru_jwhgfort/esp-r/bin/res
lrwxr-xr-x    1 jon      staff          41 Feb 12 11:39 viewer -> /Users/jon/esru_jwhgfort/esp-r/bin/viewer
```

Someone who is administrating ESP-r in a company with many users and many projects will ensure that the standard version of ESP-r is installed with file and folder permissions that prevent others from altering corporate files. Such security is possible if one login account maintains and installs ESP-r and other login accounts only have permissions to read and access the ESP-r applications and databases. Many groups thus create a login account with a name such as 'esru' to maintain ESP-r and to ensure that only people logged into that account are able to update or modify ESP-r.

There is also an issue for development work that some of the QA tests required in the submission process assume that there is a version of ESP-r installed in the folder /usr/esru and that there are ESP-r executables in the folder /usr/esru/esp-r/bin. The pattern to follow is to acquire the current development_branch of ESP-r and install it to /usr/esru and then separately install their own version of ESP-r in a different location such as /home/fred/esru_fred.

```
sudo mkdir /usr/esru
sudo chown fred /usr/esru
sudo chgrp staff /usr/esru
```

Optional Linux software is typically installed to the /opt folder. Users would like to follow this pattern should first check that /opt exists. If is does then issue the following command to create and own (e.g. if your login name is fred and your group is staff):

```
sudo mkdir /opt/esru
sudo chown fred /opt/esru
sudo chgrp staff /opt/esru
```

4.2 Quick steps to Installing ESP-r

If have read this far and you want to just grab the current tested version of ESP-r and compile it to the /opt/esru folder structure here are the steps you would take:

```
cd
mkdir Src
cd Src
mkdir cvsdude
cd cvsdude
svn checkout https://espr.svn.cvsdude.com/esp-r/branches/development_branch
cd development_branch/src
./Install -d /opt/esru --gcc4 --reuse_ish_calcs
```

You will be asked a few questions as part of the install process.

- Your computer identifies ... if correct say y
- Compiler choice ... most people will choose option 2
- XML output ... you need this if you will be doing validation tests or if you want to output predictions in csv and/or XML output.
- SQLite support... a new feature, if you don't know about it say n
- Graphics library ... select one. Note it is possible to host a version of ESP-r with X11 interface as well as one with GTK interface on your computer - they must live in different locations. You will need to run the Install script multiple times to do this.
- Retain debugging symbols ... if you say y then if it crashes it might give you more clues as to where and why. Also needed if you are going to be using a debug tool.
- Install database files ... if the first time or they have changed then you should answer y
- Install training files ... these are exemplar models (accessed if you use the open existing command in the Project manager. If this is the first time or they have changed then say y. The command line --reuse-ish-calcs speeds up the process of installing the training models.

.IP "• " 2 Proceed ... this is your chance to bail-out or continue y

At the end of the process you might want to issue the following command to clean up the source folders.

```
make clean
```

4.3 Installing ESP-r with a personal branch

Lets say the developer had already contacted the archivist to request a source code branch and that the branch was named *fred*. Subversion commands given by the archivist will create a branch in the database maintained by subversion and associate it with a particular person so that any changes made continue to be associated with the person. To work on the code or the models or databases requires that the developer *checks out* their branch and subversion will create a so-called *sandbox* in the users computer which includes their source/models/databases as well as hidden files and folders which keep track of any changes that they make in their *sandbox*. The command sequence would be as follows:

```
cd
mkdir Src
cd Src
mkdir cvsdude
cd cvsdude
svn checkout https://espr.svn.cvsdude.com/esp-r/branches/fred
cd fred/src
```

Remember that the *sandbox* is a local copy of the branch that is associated with this developer and any changes made remain within the *sandbox* until specific subversion commands are given which instruct changes to be passed back into the subversion repository database. Indeed, the developer, can also *checkout* other branches to find out the details of other developer's work. Would this allow others to corrupt a branch that they did not own? No. The commit command is password protected so you other can only look.

The above commands would apply to the command line subversion clients that are available on most computer platforms. For Windows there is a graphic tool named TortoiseSVN <http://tortoisesvn.tigris.org> which adds functionality to Windows Explorer so that a right-click on a folder provides many of the subversion commands. There is a section defining subversion commands in a later section.

If the standard compile instruction does not work because you have several versions of the compiler on your computer then you might want to adapt your Install command:

```
./Install -d /opt/esru --gcc4 --compiler_version -4.1
```

You can find out more about the source code compile process by giving the command:

```
./Install --help
```

4.4 Environment variables and files

When ESP-r is initially compiled several types of information are embedded in the executables e.g. where ESP-r is installed. Information on where to find example models and what databases to initially load is scanned in from text files. One of these text files is called `esprc` and the standard version is assumed to be in the installation sub-folder `esp-r`. Its contents are listed below and the meaning of the tokens is presented after the listing in Figure 1.

```
*ESPRC
*gprn,rectangular dump,import
*tprn,Text dump,/tmp/tx_dump
*gxwd,screen dump,import -window root
*cad,CAD package,xzip,ZIP
*image_display,TIF,display
*image_display,XBMP,display
*image_display,GIF,display
*image_display,XWD,display
*journal,OFF
*editor,editor,nedit
*report_gen,Reporting tool,xfs
*exemplars,Exemplars,/Users/jon/esru_prj_dev/esp-r/training/exemplars
*validation_stds,Validation standards,/Users/jon/esru_prj_dev/esp-r/validation/stds_list
*db_defaults,Defaults,/Users/jon/esru_prj_dev/esp-r/default
```

```
*db_climates,climatelist,/Users/jon/esru_prj_dev/esp-r/climate/climatelist
*end
```

Figure 1 A typical esprc file.

The file is in tag - data, data format. Typically the first token is a label and the second token is either an executable to be invoked or the name of a file to be used. To alter this initial specification use a text editor and change the relevant token as required. Look in the `preferences` menu of the Project Manager to access the details of this file.

The initially created version of the *esprc* file is held in the ESP-r installation folder. If a user wants a custom version of this file to use they should copy it to their *home* folder with the name *.esprc*.

- *ESPRC - this is the file type tag. It must be the first line
- *gprn - commands associate with capturing a rectangular section of the screen. The 2nd token **import** is the executable (on Linux) which captures a section of the screen.
- *tprn - commands associated with dumping the current text feedback buffer to file will write to the file identified in the second token.
- *gxwd - a variant of *gprn but which captures the whole screen.
- *cad - instructions for a CAD tool to invoke. The second token is the executable and the third token is a key word describing the type of file it creates.
- *image_display - commands related to the display of model-associated images. The second token is a key word identifying the format of the file and the third token is the name of the executable to invoke to display that type of image. There can be several *image_display lines in the *esprc* file.
- *journal - turns on a time-stamp facility which logs user actions and the key words are ON and OFF.
- *editor - which ASCII text editor to invoke if an external application is required.
- *report_gen - not used
- *exemplars - the name of the file to read which includes a list of models which can be accessed and where they are stored. The initial contents of the *exemplars* file is for use in ESP-r workshops but the contents can be edited to include other models.
- *validation_stds - the name of a file to read with information needed to commission standard tests
- *db_defaults - the name of a *default* file which holds a list of initial databases. If you want to use an alternative list of initial databases edit this file or include a reference to an alternative list of databases.
- *db_climates - the name of a *climatelist* file which holds a list of climate data sets and their location. If you want to use an alternative list edit the file or provide the name of an alternative file.

4.5 Default file assumptions

The second file which is commonly scanned when ESP-r modules start is the *default* file. The name of this file is included in the *esprc* file. The file is a tag - data format and is typically found in the installation folder. An example of this file is listed below (Figure 2). Note that the path */Users/jon/esru_prj_dev* points to an installation made for testing purposes and this path was generated as the test version of ESP-r was compiled based on the directives given at the time.

```
*ESP-r Defaults
*ipth /Users/jon/esru_prj_dev/esp-r
*cfg /Users/jon/esru_prj_dev/esp-r/training/basic/cfg/bld_basic.cfg
*ctl /Users/jon/esru_prj_dev/esp-r/training/basic/ctl/bld_basic.ctl
*mfn /Users/jon/esru_prj_dev/esp-r/training/basic/networks/bld_basic_af1.afn
*dfd /Users/jon/esru_prj_dev/esp-r/training/cfd/template.dfd
*pnf /Users/jon/esru_prj_dev/esp-r/training/plant/vent_simple/cfg/vent.cfg
*res /Users/jon/esru_prj_dev/esp-r/databases/test.res
*mfr /Users/jon/esru_prj_dev/esp-r/databases/test.mfr
*clm /Users/jon/esru_prj_dev/esp-r/climate/clm67
*prs /Users/jon/esru_prj_dev/esp-r/databases/pressc.db1
*prm /Users/jon/esru_prj_dev/esp-r/databases/material.db3.a
*mlc /Users/jon/esru_prj_dev/esp-r/databases/multicon.db3
*opt /Users/jon/esru_prj_dev/esp-r/databases/optics.db2
*evn /Users/jon/esru_prj_dev/esp-r/databases/profiles.db2
*pdb /Users/jon/esru_prj_dev/esp-r/databases/plantc.db1
*ecdb /Users/jon/esru_prj_dev/esp-r/databases/elcomp.db1
```

```
*mcdb /Users/jon/esru_prj_dev/esp-r/databases/mscomp.db1
*icdb /Users/jon/esru_prj_dev/esp-r/databases/icons.db1
*mldb /Users/jon/esru_prj_dev/esp-r/databases/mould.db1
*sbem /Users/jon/esru_prj_dev/esp-r/databases/SBEM.db1
*end
```

Figure 2 A typical default file.

As with the previous files the name of the file is associated with a specific topic and/or dialogue within the user interface. These dialogues associated with specific types of model files require a default name and the default file names are scanned in via the *default* file rather than being hard-coded into the interface. The name of the file can be altered by editing the file.

- *ESP-r Defaults - this must be the initial line of the file.
- *ipth - this is the path to where ESP-r has been installed based on the specific commands given during the installation process
- *cfg - this is a default file name for a model configuration file (useful for demonstration purposes)
- *ctl - this is a default file name for control loop definitions
- *mfn - this is a default file name for an air flow network
- *dfd - this is a default file name for a CFD domain description
- *res - this is a default file name for a zone predictions (results) file. This file should be created during the install process so that it is easy to demonstrate ESP-r.
- *mfr - this is a default file name for mass flow predictions
- *clm - this is a default file name for climate data. This climate file should be created during the install process.
- *prs *prm *mlc *opt *evn *pdb - these are default file names of databases (in case the user request a default database. Many users will change the name of the database files to suite the needs of their work. This file can be accessed via the `preferences` menu of the Project Manger.

The last ASCII file which is used by ESP-r modules on a regular basis is the so-called *climatelist* file. This file is referenced by the *esprc* file (see above discussion) and includes a list of the climate data sets that were installed on the computer. When the interface of one of the ESP-r modules presents a list of available climate data it scans this file.

Each time you want to add climate data to your computer you should edit this file with a text editor so that the listing will include the new file. There is a detailed discussion of how to use **clm** to add new climate files in Chapter 6 of the ESP-r Cookbook. A portion of this file is shown below (Figure 3).

```
*CLIMATE_LIST
*group  ESRU standard climates
# WARNING: Keep this file up to date with current directory structure !
*item
*name   Default UK clm Climate
*aide   Climate data as distributed with ESP-r for testing purposes.
*dbfl   /usr/esru/esp-r/climate/clm67
*winter_s 2  1  12  3  30  10  31 12
*spring_s 13 3  14  5   4   9  29 10
*summer_s 15 5   3  9
*winter_t 6  2  12  2  20 11 26 11
*spring_t 17 4  23  4   2 10  8 10
*summer_t 3  7   9  7
*avail  ONLINE
*help_start
Location is 52.0N and 0.0E. The solar radiation is Direct Normal.
 Month       Minimum  Time           Maximum  Time          Mean
 Jan          -6.4 @20h00 Sun  8    12.7 @14h00 Sun 29     3.8
 Feb          -1.9 @ 5h00 Tue 14    12.2 @13h00 Thu  2     5.2
 Mar          -0.8 @24h00 Fri 31    16.1 @15h00 Tue 21     6.8
 Apr          -1.9 @ 2h00 Sat  1    19.4 @15h00 Mon 17     7.1
 May           0.0 @ 3h00 Wed  3    22.7 @14h00 Thu 11    10.4
 Jun           5.0 @ 2h00 Fri  9    21.1 @15h00 Tue  6    13.6
 Jul           9.4 @ 3h00 Mon  3    27.7 @12h00 Mon 17    18.0
 Aug           7.7 @ 4h00 Sat  5    24.4 @12h00 Tue  1    15.6
 Sep           5.0 @ 6h00 Thu 21    22.2 @12h00 Tue 26    13.5
 Oct           2.2 @ 5h00 Mon 30    19.4 @13h00 Sat  7    10.8
```

```
 Nov          -0.8 @ 5h00 Mon 27    14.4 @14h00 Sat 11     5.2
 Dec          -4.2 @ 1h00 Sat  9    12.7 @ 9h00 Sat 23     3.8
 All   -6.4 @20h00 Sun  8 Jan  27.7 @12h00 Mon 17 Jul   9.5
 Typical winter week begins Monday 6 Feb,
 Typical spring week begins Monday 17 April,
 Typical summer week begins Monday 3 July.
 Typical autumn week begins Monday 2 October.
 Typical winter week begins Monday 20 November,
*help_end
*item
*name   ALBUQUERQUE NM USA iwec 723650
*aide   ALBUQUERQUE NM USA iwec 723650 was sourced from US DoE web Sep 2005
*dbfl   /usr/esru/esp-r/climate/USA_NM_Albuquerque_iwec
 . . .
```

Figure 3 A typical section of a climatelist file.

The *climatelist* file includes the following types of information:

- a display name for the climate data (as seen the the interface list)
- a brief documentation about the climate data
- its location on the computer
- the start and end dates of each of five seasons (winter from 1 Jan, spring, summer, autumn, winter ending 31 Dec). These dates typically were supplied by a person who knows the climate of the region and the social customs of the region.
- the start and end dates of a typical week in each season. There is an facility in the **clm** module which searches for typical weeks based on heating and cooling degree days and solar radiation patterns.
- a block of text up to 60 lines which provides a summary of the climate. This block is auto-generated within **clm** and you can edit it and extend it if required.

5 Code Documentation

The ESP-r community has evolved guidelines for coding that is included in the ESP-r distribution. Ideally, one would judge code documentation by whether others are able to understand the purpose of subroutines, follow procedural logic and understand looping structures.

There is also a need for clarity in data structures such as common blocks and local variables as well as the parameters which are passed into and returned from subroutines and functions.

Clarity is a challenge. Extremes tend not to work e.g. *ij* and *loop_for_number_of_boilers_counter* both have drawbacks. If a common block is used a dozen times in one source file does it obscure the code if it is fully documented each time?

ESP-r contains much legacy code. Some of this requires passion to digest even if compilers can do it without complaint. Where the author of the code is still active they may be able to re-code but some code the loss of the initial flow diagram presents a considerable barrier for reverse engineering.

The diverse backgrounds in the development community has resulted in a number of coding and documentation 'styles'. Being open source, there is limited scope to enforce coding styles. What follows is a set of general principles and examples of patterns/styles of coding and documentation. These guidelines have evolved over time in response to the evolution of FORTRAN an C coding conventions and trends within the ESP-r development community and are considered during the submission of code.

5.1 General Principles

- Clear and concise documentation has the same importance as bug-free code.
- The source code should be well documented through the use of appropriately chosen variable and subroutine names and a transparent code structure.
- Details about the operation of code sections should be provided within related code annotations and not in external documentation.
- Invest the necessary time to ensure that your code is understandable to your colleagues (and to yourself in a few years time!).
- Use proper English sentences to document code—this includes capitalization and punctuation. Strive for clarity as incoherent language leads to confusion and ambiguity.
- When adding new functionality maintain consistency in style with existing source code. For example, coefficient generator subroutines for different plant component types will usually differ only in the equations used to calculate matrix equation coefficients. To install a new plant component it is usually possible to adopt the pattern of an existing component.
- Be consistent in the use of variable names throughout the code. For example, all plant component coefficient generators produce coefficients for the plant matrix solver. These are local variables to each subroutine and are passed in the calling statement. These local variables are named 'COUT' in each plant coefficient generator. Therefore, for consistency's sake it is preferred to use this same variable name in a new plant coefficient generator. sp

5.2 Documentation patterns

The source code should be well commented and these comments should precede the code fragments to which they relate. If code is related to methods published in the literature or in reports include a reference within the source code (see example below). This is especially important if the document shows the code works as expected.

Comments enchance understanding, but they can obscure the code it poorly implemented. The layout of comments can be used to logically group blocks of code. Two examples that illustrate acceptable commenting styles follow.

```
      <preceding code fragment>

C Loop through each of the selected zones and scan the Operations file
C if it exists. If not, insert default crack connection.
      do 38 izt=1,izn
        iz=ivals(izt)
        <following lines of code>

C Increment pointer to the current zone.
        nodeforcurrent=nodeforcurrent+1
```

```
      <preceding code fragment>

C Passed parameters for cfgtogg
#ifdef OSI
      integer icfg_type  ! model cfg type
      integer icfgz      ! if non-zero then there are zones
      integer icfgn      ! if non-zero then there are networks
      integer icfgc      ! if non-zero then cfg file known
      integer icfgdfn    ! if non-zero then cdf domain exists
      integer iicfgz     ! there are zone related images
      integer iicfgn     ! there are network related images
      integer iicfgc     ! there are control related images
      integer iicfgdfn   ! there are cfd related images
#else
      integer*8 icfg_type,icfgz,icfgn,icfgc,icfgdfn,iicfgz
      integer*8 iicfgn,iicfgc,iicfgdfn
#endif

      <preceding code fragment>
C Combine the comment from the first line with the one after the zone CTYPE.
        ipra=lnblnk(phrasea)
        iprb=lnblnk(phraseb)
        iprc= 63 - ipra
        iwidth=ipra + iprb + 1
        if(iwidth.lt.64)then
          write(zdesc(ICOMP),'(3a)') phrasea(1:ipra),' ',
     &      phraseb(1:iprb)
          lnzdesc(ICOMP)=lnblnk(zdesc(ICOMP))  ! update the length of this string.
        else
          write(zdesc(ICOMP),'(3a)') phrasea(1:ipra),' ',
     &      phraseb(1:iprc)
          lnzdesc(ICOMP)=lnblnk(zdesc(ICOMP))  ! update the length of this string.
        endif
       <next code fragment>
```

```
        <preceding code fragment>

C-------------------------------------------------------------------------------
C Calculate the molar flow rate of each gas constituent.
C-------------------------------------------------------------------------------
C-------N2 in air and fuel flowing into FCPM does not react.
        Ndot_FCPMexh_N2  = chi_air_N2*Ndot_FCPM_air
     &                   + chi_fuel_N2*Ndot_FCPM_fuel
C-------Ar in air flowing into FCPM does not react.
        Ndot_FCPMexh_Ar  = chi_air_Ar*Ndot_FCPM_air
C-------O2 in exhaust comes from fuel and excess air.
        Ndot_FCPMexh_O2  = chi_fuel_O2*Ndot_FCPM_fuel
     &                   + lambda_FCPM*Ndot_FCPM_O2_stoich

        <next code fragment>
```

5.2.1 Subroutine descriptions

Each subroutine should start with a high-level explanation of its purpose and how this is achieved. An example follows.

```
C Subroutine XYZ calculates the four heat loss factors for the foundation
C in the zone under consideration. Correlation coefficients for
C the 'corner-correction method' are used. The factors are placed into
C Common Block BSHLF for use in later heat loss calculations.
```

It is also useful to define the parameters passed with the subroutine in terms of their data types and what each parameter is used for.

```
C ************* EMKGEO
C Generic routine to write a geometry file (GEN type) based on infor-
C mation currently held in common blocks G0 G1 G3 G4 G6.  It is
C assumed that this information has been checked.
C GENFIL is the name of the file to be written to (any existing file
C by this name is overwritten).
      SUBROUTINE EMKGEO(IFILG,GENFIL,ICOMP,iwf,IER)
#include "building.h"

C geometry.h provides commons G0/G2/G4/prec17/precz/c20.
#include "geometry.h"

      integer lnblnk       ! function definition

C Parameters
      integer IFILG        ! file unit
      character GENFIL*72  ! file name
      integer ICOMP        ! the zone number
      integer iwf          ! 3 create/overwrite, 4 confirm before overwriting.
      integer IER          ! IER 0 OK IER 1 problem
```

5.2.2 Citing papers and references

Citations to algorithms and data sources should be given at the beginning of a subroutine. These citations should be complete as a colleague may need to locate the paper or report in the future. Refer to proprietary reports (internal reports, drafts reports, private sources) only when the information is not available in the public domain (conference proceedings, journals, theses). An example follows.

```
C Smith A and Jones B (1999), 'Heat Transfer Coefficient Correlations for
C Building Energy Modelling', Int. J. Heat and Mass Transfer, 38(8), pp856-884.
```

Then, further down in the source code immediately preceding the algorithm cite the appropriate reference from those specified at the beginning of the subroutine and include the specific page numbers, table numbers, equation numbers *etc*, as applicable. This can be helpful to your colleagues (and to yourself) in tracing bugs. An example follows.

```
      elseif( icor .EQ. 5 )then

C Coefficient correlation for a wall with a radiator located under
C a window (Smith and Jones publication, Table 2, Equation 6).
        hc = 2.30*(dt**0.24)
```

5.2.3 Describing assumptions

Whenever an assumption is made, add a comment specifying where the assumption came from. If the assumption can be referenced to a paper, report or book it should be. If it came from a discussion, explain why the assumption was made.

Annotating blocks of code

Use comments to mark ends of blocks. This is useful in identifying blocks when IF, DO or WHILE constructs extend over many lines of code or where there are multiple embedded loops or conditional code blocks. An example follows.

```
      do jj=1,mpcdat

C Has iteration for this plant additional output been requested?
        if ( iPlt_Output_Iter_Flag(ii,jj) .ne. 1 ) then
          < line of code >
          < line of code >
          < line of code >
        endif   ! <- matches  if ( iPlt_Output_Iter_Flag(ii,jj)...
      enddo   ! <- matches do jj = 1, ...
```

5.2.4 Grouping lines of code

Code annotations should be succinct but sufficiently detailed so that the purpose of every line of the code is obvious. If in doubt, say more. Annotate code by grouping lines logically. An example follows.

```
        <preceding code fragment>

C Override the calculated values if user has specified a convection file
C with fixed coefficients (ie. 'type 1' control over convection calculations).
C-------Does a convection file exist?
        IF( IHC(ICOMP).EQ.1 ) THEN
C---------Has the user specified fixed coefficients in the convection file?
          Ltype1 = 0
          DO 22 k=1,NHCFP(ICOMP)
            if( iCTLTP(ICOMP,k).eq.1 ) Ltype1=1
   22     CONTINUE

        <following code fragment>
```

5.2.5 In-line comments

Comments can be added to the end of any code line using the '!' character. This makes commenting blocks of variable definitions and short code lines easier to read. However, in-line comments must not be embedded within statements spanning multiple lines as some compilers will experience problems when parsing such comments.

6 Coding Style and use of FORTRAN/C

6.1 Source code files and subroutines

Long subroutines can be cumbersome to read, understand and test. It is good practice to keep the size of subroutines small and use calls to other subroutines when a distinct set of calculations need to be performed. If there is no distinct grouping of calculations, it is preferable to keep all calculations together. Further, if a set of equations is used in more than one area, the set should be relocated into its own subroutine.

When adding significant new functionality, related subroutines should be grouped into a file and this file located within an appropriate directory. There is no practical limit to file size, however files should contain subroutines that perform related functions or relate to a common theme or purpose. For example, the static template, coefficient generator and related subroutines for a new plant component should each be grouped into a single file and this file added to the *plt* directory.

Saved variables, global variables and common blocks

Any code that depends on static variables should explicitly declare these variables with a SAVE statement.

When multiple routines reference a large set of global variables held in a COMMON statement, consider placing the COMMON declarations in a header file. This reduces the risk of mismatches between the type and dimension of declared global variables, which invariably lead to segmentation faults.

6.1.1 Equations

Equations should be evaluated in the same manner that they would be written in a scientific publication. Avoid combining terms (and hence distorting clearness) in an attempt to produce computationally efficient code.

6.1.2 Working with existing files

When working with existing source code files maintain compatibility with the style.

- Code line length should be confined to 72 characters. Unless you are specifically working in F90 free form, in which case the source file name must end with .f90.
- Use the '&' character for line continuations (6th column).
- **Do not use tab characters** within the source code, including comment lines. If using a file editor which allows tab characters, be certain to configure it such that tab characters are converted to spaces upon saving the file.
- All loop statements (IF-THEN, DO, WHILE) should be indented by 2 characters.

GOTO statements

GOTO statements should be avoided. There are plenty of alternatives in modern FORTRAN (e.g. the WHILE statement). A possible exception is when working with an existing subroutine that makes extensive use of GOTO statements in which case it may be clearer to maintain the style.

Floating point comparisons

All possibilities for division by zero must be trapped at the highest possible level and floating point comparisons avoided because variation in compilers and machine architectures can produce unexpected behavior when the differences in values approach machine precision. For instance, the following example may produce inconsistent results. The way real numbers are held makes it unlikely that a comparison of the real number y and the constant 0.0 would be evaluated as true.

```
real x, y, z
if ( y .eq. 0.0 ) then
  stop   "Divide by zero error in Subroutine ABC: y is zero!"
else
  z = x / y
endif
```

By comparing the difference between the floating numbers with a tolerance (presumably one several orders of magnitude less than the variation between the variables), these inconsistencies can be eliminated. A utility subroutine eclose is provided for this.

```
real x, y, z
logical close
call eclose(y,0.0,0.0001,close)
if ( close ) then
  stop "Divide by zero error: y is zero!"
else
  z = x / y
endif
```

6.2 Variable and subroutine names

Variable and subroutine names should be descriptive and clearly delineated from other names. Most modern FORTRAN compilers can support long names.

Generic variable names (e.g. INTEGER i, j, k; LOGICAL Done) for loop controls are discouraged. Instead, use descriptive names (e.g. INTEGER iZone, iSurface, iLayer; LOGICAL Loop_Unconverged).

When working with numerical constants or integer flags, define meaningful symbolic parameters to represent the constant or flag.

Documenting local and common block variables

COMMON block variables should be documented in the subroutine where they are first introduced. When variables are defined in the code, the comment should include the units of the variable (e.g., J, kW, °C, K *etc*).

Local variables should be documented in the subroutines in which they are used.

Two examples that reveal acceptable styles for documenting variables follow.

```
C Maximum infiltration ('finfmax') and ventilation ('fvntmax') flow
C rates (m^3/sec) for each zone. Variable 'icompforinf' is the component
C number associated with unique infiltration flow paths while 'isrczforvent'
C is the source zone associated with the largest ventilation rate.
      dimension finfmax(mcom), fvntmax(mcom), icompforinf(mcom)
      dimension icompforvent(mcom),isrczforvent(mcom)
      integer icompforinf,icompforvent,isrczforvent
      real finfmax,fvntmax
```

```
      real fCyl_Volume       ! Cylinder gas volume (m3)
      real fCyl_solid_mass   ! Mass of cylinder wall (kg)
      real fCyl_solid_Cp     ! Specific heat of cylinder wall (J/kg oC)
      real fCyl_UA_ambient   ! Heat transfer coeff. between cylinder
                             !   & ambient (W/oC)
```

6.2.1 Explicit declarations

Implicit type casting must be avoided. All new subroutines must include an 'IMPLICIT NONE' statement. This requires that all variables be defined before use and forces programmers to give proper consideration to their name, use, type and documentation. Explicit declarations also reduce debugging effort and strengthen confidence in the program's validity, as mismatches in variable names will produce errors at compile time.

Many existing ESP-r subroutines use the IMPLICIT rule to automatically define all variables that start with the letters 'I' through 'N' as INTEGER and all other variables as double precision REAL. This naming convention should be followed when altering existing subroutines that use the IMPLICIT rule, but any new variables added to the routine should be explicitly declared.

6.2.2 Type casting

Data must often be converted between real and integer formats during execution. Rather than rely on the compiler's native type casting behavior, such conversions must always be done explicitly:

```
real x, y
integer i

x = 1                    ! <- Implicit casts
i = x                    !    (bad practice)
y = i

x = 1.0                  ! <- Explicit casts
i = int(x)               !    (good practice)
y = float(i)
```

6.2.3 FORTRAN/C parameter passing conventions

ESP-r technical modules are primarily written in FORTRAN with graphics implemented in C (e.g. X11 or GTK library calls) and some facilities making use of C++ code. An intermediate layer of C code exists to mediates between the low level graphics calls and the primary FORTRAN code. Typically FORTRAN calls C although there are cases where the reverse is true.

There are a number of established patterns for passing integers, reals, characters and arrays between the two languages, which work with a number of compilers and across platforms. Examples can be found in esru_lib.F and esru_x.c both located in the 'lib' directory. (For example, for every string array passed to C, the C parameter list includes an additional 'int' that holds the array length.)

As a rule, FORTRAN never calls directly to the low level graphic functions although C will occasionally call FORTRAN to request information.

A limited number of C++ source files are associated with ESP-r. Passing conventions are less established for this.

In some cases '#ifdef' statements are used to signal differences between the X11 and GTK implementations or to differentiate between F90 and F77 code. Ifdefs are not needed for Windows/Linux/Unix differences as an *isunix* function is provided.

7 Quality Assurance Tasks

To ensure that the evolution of ESP-r is robust the ESP-r community has agreed that a broad range of tests are necessary. The objectives of testing is to:

- identify code which includes syntax errors, variables which are not correctly typed as well as code logic which could cause numerical errors or might work inconsistently across a range of computer and operating system types,
- demonstrate to other ESP-r developers and users that source code additions and modifications function as expected,
- ensure that the changes you've made behave consistently on all supported platforms, and
- ensure that changes do not interrupt the work of other ESP-r users and developers around the world.

Although the initial regime was supported by familiarity within a small community, the expanding community required evolved procedures which recognized that:

- each commit introduces the risk of errors and there is a considerable benefit in identifying these as early as possible
- developers who are focused on one facet of ESP-r may not realize that their work may have unintended consequences,
- the audit trail built into source code control tools is a powerful aid to testing new features

The testing method is multi-faceted. The code has to pass a syntax check (discussed further below), it had to compile on multiple platforms, over a hundred example models had to be installed successfully and the predictions from simulations on test models had to be within a specific tolerance.

What was initially a ritual undertaken by the core developers been codified and some parts have been automated so that others could participate more easily.

7.1 Identifying faulty code

New entrants to the development process often begin with the view that code that compiles must be correct. The archivist has a more specific set of requirements:

- it must compile,
- if the code relates to an interface the response to users actions must have been tested,
- code which reads files should be well tested and if file formats change there should be some means for older models to be used and/or updated,
- code associated with calculations must not introduce unexplained changes.

7.2 Syntax checking techniques

The identification of faulty code relies on several complementary techniques:

- the use of the reporting facilities of the compiler
- third party syntax tools such as forcheck[1]
- run time error traps via compilers or debuggers

The Forcheck static analyzer[1] inspects code for inconsistencies and errors that might otherwise be missed by compilers. Some developers (including the author) use these tools prior to compilation and others rely on the automatic invocation of syntax tools triggered by a submission to the repository.

To make best use of tools such as forcheck, you'll first need to ensure that the ESP-r source code folders include the full set of source files associated with a particular module. This is most easily accomplished by answering "yes" at the Install script prompts: "Retain debugging symbols? (y/n) [y]".

Forcheck output is generally verbose, and can be even more so if it's not configured to respect language extensions available in modern compilers. A Forcheck configuration file (esp-r.cnf) suitable for use with ESP-r is available in the tester/scripts folder; to use it you must first set the FCKCNF environment variable. Some users prefer to copy the file esp-r.cnf to the installed location of Forcheck (typically /usr/local/lib/forchk). Using the bash shell, enter the following command:

[1] http://www.forcheck.nl/

```
export FCKCNF="/path/to/forcheck/configuration/esp-r.cnf"
```

To carry out a syntax check, move to the esru folder corresponding to the ESP-r binary you wish to test (you can also create a script to run the tests in multiple folders). For instance, to test the ESP-r Project Manager *prj*, move to the folder "esruprj". To test an ESP-r binary when linked to the X11 graphics library, invoke Forcheck using the following command:

```
forchk -I ../include *.F *.f90 ../lib/esru_ask.F ../lib/esru_blk.F ../lib/esru_libNonGTK.F
```

To test an ESP-r binary when linked with the GTK library, invoke forcheck using the following command

```
forchk -I ../include *.F *.f90 ../lib/esru_ask.F ../lib/esru_blk.F ../lib/esru_libGTK.F
```

7.3 Understanding syntax reports

Forcheck will produce a report identifying errors and warnings and advise related to your source. Pay particular attention to portions of the report pertaining to files you've changed. And because your changes may conflict with source code in other locations (e.g. esrucom) it is also necessary to scan other portions of the report.

The following are extracts from a syntax report from Forcheck. The first extract is the header of the file with the version number and compiler emulation used:

```
      F O R C H E C K  (R)  V13.7.02
Copyright (c) 1984-2007  Forcheck b.v. All rights reserved
Licensed to: University of Strathclyde, Mechanical Engineering, UK
PC/Linux (), serial: 9611386
/usr/local/lib/forcheck/esp-r.cnf

   -- gfortran compiler emulation

   -- Fortran 95 syntax

   -- scanning input files
   -- program unit analysis
   . . .
```

Note: there will be a slightly different report generated depending on which compiler you are emulating. In the *program unit analysis* section each of the source files and subroutines will be listed along with information messages, warnings and errors.

```
   -- file: e3dviews.F

      - program unit: LENS

      - program unit: MATPOL

      - program unit: MATPOLS

      - program unit: CLIPFL

      - program unit: CLIPSUR

      - program unit: PLNBX

      - program unit: PLNOFSUR

      - program unit: CUTPOL

      - program unit: CUTSUR
      . . .
```

This is what we am for! Subroutines with no messages. When we do get information messages or warnings or errors this indicates that there is still work to do.

```
   -- file: aco.F
      - program unit: REVTIME
(file: aco.F, line:     171)
    694              write (outs,'(2a,3(F6.2,5x),F7.5)')DESCR(1:6),'@',
```

```
    695                                          THR,TTMax,RHMax,FMax
  DESCR
(file: aco.F, line:     694)
**[313 I] possibly no value assigned to this variable
  THR
(file: aco.F, line:     695)
**[313 I] possibly no value assigned to this variable
```

The variables DESCR and THR might not have a value at this point in the code. They might have been set within a logic statement that is not always used (e.g. if()....then()....endif). The code may work ok, but the advise is to look at the logic to determine if these variables should be initialized at the start of the subroutine.

```
   -- file: acoesp.F
      - program unit: OPENDB
  ICLN
(file: acoesp.F, line:      51)
**[325 I] input variable unreferenced
  . . .
      - program unit: ZONEDISP
  CLOSE
(file: acoesp.F, line:     306)
**[681 I] not used
  . . .
  /C1/
(file: egeometry.F, line:    3987)
**[676 I] none of the objects of the common block is used
  . . .

      - program unit: CHECKSORT
  CHECKSORT, dummy argument no  4 (IER)
(file: eroper.F, line:    2474)
**[557 I] dummy argument not used
 . . .
   INICNN
(file: nwkrewr.F, line:     676)
**[313 I] possibly no value assigned to this variable
    678             write(ddatrib(INICNN,JJ,2),'(a)') WORD(1:12)
 . . .

      - program unit: EVSET
   4631       ELSEIF(IVERT.EQ.19.AND.INPIC.GE.1)THEN
  INPIC
(file: ../lib/esru_libNonGTK.F, line:    4631)
**[313 I] possibly no value assigned to this variable
```

Are these variables unreferenced or unused because of a typographic error? Did you intend to use a variable but forgot? Is it left over from a prior version of the code? For example, the report on CHECKSORT indicates a parameter IER is not used. Were you intending to return an error state but forgot to set this variable in the code of the subroutine?

The write statement which include INICNN could be a serious issue. If INICNN has not been defined then it may be treated as a zero and this may be outside the range of the array and cause a crash.

The ELSEIF()THEN statement message could also result in the flow of the ELSEIF logic occasionally being wrong. An undefined variable INPIC would result in the wrong ELSEIF statement being used.

```
   -- file: ADS_storage_tanks.F

      - program unit: ADS_TANK_FUEL_STATIC_TEMP
    397       tank_DHW_draw = ADATA(IPCOMP,13)
  tank_DHW_draw = ADATA(IPCOMP,13)
(file: ADS_storage_tanks.F, line:     397)
**[699 I] implicit conversion of real or complex to integer

  . . .
**[699 I] implicit conversion of real or complex to integer
   2391         needHTOperLoad=(1-relaxFact)*needHTOperLoad +
   2392                     relaxFact*(MAX(0.0,(neededHeatTOper - pIntHeat)))
  1
(file: CETC_BATTERY.F, line:    2391)
**[344 I] implicit conversion of constant (expression) to higher accuracy
          (Specify the constant (expression) in the appropriate data type kind)
```

```
 . . .

- program unit: DG_CONTROLLER_INITIALIZE
   1436          if(HWT_coeff_a .gt. 1 .or. HWT_coeff_a .lt. 0) then
  1
(file: DG_controller.F, line:    1436)
**[344 I] implicit conversion of constant (expression) to higher accuracy
          (Specify the constant (expression) in the appropriate data type kind)
```

There are several issues. Firstly, tank_DHW_draw looks like it might be a real number but is has been defined as an integer. Second, the value of ADATA() is a real and the compiler is being forced to cast this value to a less accurate representation when it assigns it to tank_DHW_draw. For clarity the code should use an NINT() to explicitly cast the variable.

In the second case, if the value '1' should have been '1.0' then the compiler would not have to work as hard and the logic would have been clearer.

In the third case we are not testing for equality so the logic will work as intended. The computer is, however, forced to do extra steps to cast the constant to a real representation.

```
   1768         IF( DC_required_by_PCU == PCDATP(IPCOMP,1) ) THEN
  DC_required_by_PCU == PCDATP(IPC...
(file: Annex42_fuel_cell.F, line:    1768)
**[340 I] equality or inequality comparison of floating point data
          (comparing real data for (in)equality is potentially risky)
```

This information message signals logic that probably will not work as intended. Because of the way real numbers are represented this IF()THEN statement will rarely, if ever, be executed as intended. What is required is to find out if two real numbers are very close to each other in value and for this there is a library function eclose().

```
   -- file: CETC_BATTERY.F

      - program unit: POWOC_CETC_BATTERY
    256         IF (batDemandP .EQ. 0. ) THEN
  batDemandP .EQ. 0.
(file: CETC_BATTERY.F, line:     256)
**[342 I] eq.or ineq. comparison of floating point data with zero constant
```

We should assume that batDemandP is never exactly zero (to the full range of the type real) and thus the block of code that follows will not be called.

```
      - program unit: CSTATE_NAME
    291          cState_Name = 'water'
  cState_Name = 'water'
(file: h2_matrix_library.F, line:     291)
**[383 I] truncation of character constant (expression)
    293          cState_Name = 'air'
  cState_Name = 'air'
(file: h2_matrix_library.F, line:     293)
**[383 I] truncation of character constant (expression)
```

This message says that cState_Name (a string variable) is not large enough to hold the string defined on the right side of the equation. Some compilers might treat this by only putting in sufficient characters for the string variable. Other compilers might treat this as a string buffer overflow. String buffer overflows are nasty. They can corrupt memory. They can also cause the application to crash.

```
   3451          call add_to_xml_reporting (
   3452                 AIMAG(ENODVLT(iElec_node)),
   3453                 H3K_rep_NAME,
   3454                 'units', '(radians)',
   3455                 'Electrical network node: V angle')
  ADD_TO_XML_REPORTING, dummy argument no  1
(file: h3k_report_data.F, line:    3455)
**[690 I] data-type length inconsistent with data-type length at first ref.
          (The data-type length is explicit in one instance and implicit in the other)
```

This could indicate a problem between two different instances of AIMAG. In one place and explicit type has been given and in the other the type is assumed. For some compilers this may not be a problem if the implicit assumption is the same as the type.

```
  FDHW_COLDMAINTEMP, referenced in FDHW_WATERDRAW, argument no  1 (IMONTH)
**[616 E] input or input/output argument is not defined
```

The subroutine FDHW_COLDMAINTEMP is passed an argument IMONTH and this argument is used within the subroutine but the calling code has not defined the value of IMONTH. Some compilers will treat IMONTH as a zero. If the code in the subroutine is able to accept a zero value this compiler assumption should not cause a problem. If the code in the subroutine is not expecting a zero then the wrong value will be calculated or a numerical error result.

```
  /HVAC_H3KNAMES/, declared in INITIALIZE_SYSTEM_TYPES
**[233 I] common block inconsistently included from include file(s)
```

The syntax check has found that there are two different definitions for the common block HVAC_H3KNAMES. This may or may not cause a problem for the application. Or the common block may be defined in two different include files. This information may be helpful in guiding the developer to the locations in the code which can then be manually checked.

```
      - program unit: ARCHIVEIT
  3
(file: cadio.F, line:    2283)
**[124 I] statement label unreferenced
(file: cadio.F, line:    2283)
**[ 84 I] no path to this statement
```

This probably signals that the code has been revised and some of the logic can no longer be reached. This may not cause problems for the compiler but is potentially confusing to those reading the code.

The archivist has the option of requiring reported warnings and errors to be fixed prior to taking the code into the main development branch.

8 Working with the ESP-r repository

A version control system (subversion) is used to facilitate the management of the ESP-r source code archive. The archive holds the current and past states of the ESP-r source code and databases and example models and documentation within a database. Subversion also supports the concept of different simultaneous versions of ESP-r via separate named branches within the repository.

The version control system provides commands to add and remove and update files within the repository based on permissions established by the Archivist. Individuals in the development community *check out* particular versions of ESP-r or states of ESP-r into local *sand boxes* on their computer and changes made within a sand box may eventually be merged back into the repository and shared with others if it passes a testing regime.

To re-iterate, the *repository* is held remotely. Developers may have one or more local *sand boxes* which are created via subversion requests to the repository. Alterations in a local sand box **remain local to their computer and unknown to the repository** until subversion commands (e.g. *add, delete, move, commit*) are given.

Subversion is well documented on the web and there are a number of tutorials and books on subversion which the novice developers are advised to read. The **last section of this document** gives many examples of the use of subversion so reading that before you proceed could save you time.

A schematic representation of the ESP-r repository is given in Figure 4.

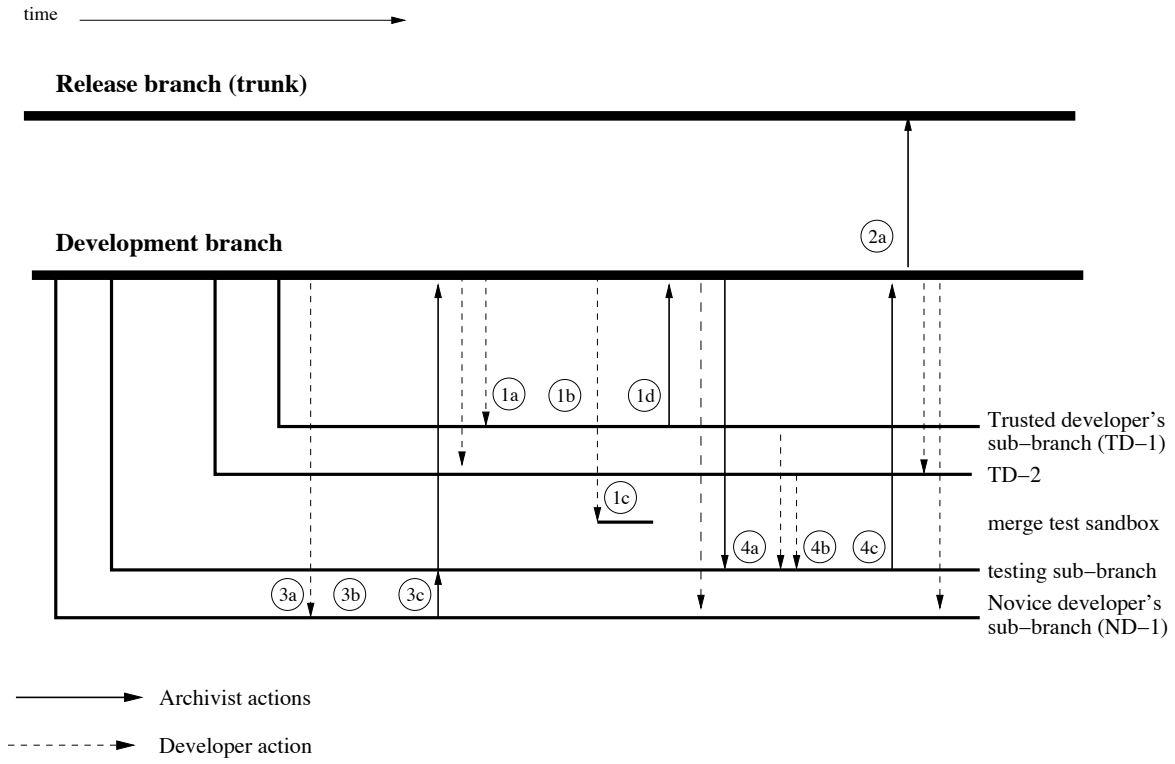

Figure 4: Schematic representation of the ESP-r repository.

The process associated with contributing code includes a number of steps. The intent is to maintain the quality of the ESP-r distribution and identify potential bugs before they become an issue for the community.

Any interested party can, at any time, download the source representing the latest, quality assured version of ESP-r.

The development_branch is updated periodically following the completion of quality assurance procedures (represented by '2a' in Figure 4). This procedure comprises the simulation of a number of pre-constructed models and the comparison of predictions against archived results corresponding to previous releases. The procedure also entails testing on various supported operating systems and several compilers.

When the development_branch is updated a summary of the changes is provided. All interested parties have access to the development_branch through subversion (all may read, only the Archivist can write to the development_branch). Binary distributions of ESP-r for a number of operating systems and computer types are created several times a year from the development_branch.

8.1 Work flow within the repository

The following discussion provides examples of many of the steps shown in Figure 4. The archivist is under no obligation to take your contributions unless you demonstrate that the changes **do no harm**. The discussion also reflects the risks associated with corrupting your work and recommendations for backups and additional checks along the way.

Developers who wish to contribute source code should contact the Archivist and request that a branch be created. Developer-specific sub-branches (e.g. ND-1 and TD-1 in Figure 4) start as a copy of the *development_branch*, and modifications made by a developer remain in the sub-branch until they are *merged* into the development_branch by the Archivist (at which point all others in the development community have access to the changes.

The development community has evolved a number of strategies which enhance working with the repository and with others in the community. A few of the strategies are:

- Keep in sync with the development branch. When notice is given of a change in development_branch check out a fresh sandbox of your branch and merge in the development_branch changes. If all goes well commit the result of the merge.
- Subversion commits should be documented (what issue was addressed, what was changed, what will users notice, what will other developers notice, is it work-in-progress or a completed task, how was it tested, what were the results of the test, what compilers and operating systems were used, results of syntax checks and QA tests run). Figure 5 is an example of the kind of message that all developers get as changes are made in a branch.

```
COMMIT LOG MESSAGE
~~~~~~~~~~~~~~~~~~~~~~~~
- Reduce differences between Install scripts.

  Include Solaris F90 CC library dependencies (libstdc++ is
  differently named in Sun Studio).

  This commit to allow testing on different platforms (Solaris,
  32 bit, Solaris 64 bit, OSX PPC and Intel etc.)

Testing
- Compile GCC 3.4 X11 with and without -m32 compiler directive
  on a 32 bit Ubuntu 8.4 and run tester.pl script with no
  differences reported.



CHANGE-SET SUMMARY
~~~~~~~~~~~~~~~~~~~~~~~~
A summary of these changes is available at:

http://node9.cvsdude.com/trac/espr/esp-r/changeset/3736

CHANGE LOG
~~~~~~~~~~~~~~~~~~~~~~~~
   U   branches/Jon_Hand/src/Install
   U   branches/Jon_Hand/src/Install_32
   U   branches/Jon_Hand/src/Install_RH_studio

REVISION HISTORY
~~~~~~~~~~~~~~~~~~~~~~~~

Revisions:
 - http://node9.cvsdude.com/trac/espr/esp-r/log/?verbose=on
```

Figure 5: Example of broadcast message of a new contribution.

- Subversion commits **should be atomic**. For example, if a common block is altered, change all instances of the common block and, after testing, commit this as one commit. If an example model has also been updated commit this separately from the common block changes. If an equation is changed which will alter predictions commit this change separately. If you merge in changes made in the development_branch into your branch **commit the merge as a separate commit**.

- If possible compile on more than one platform so that compiler specific issues are identified. If you have access to a syntax checker use this to identify issues before they are committed (see the discussion elsewhere about how this saves time).
- The testing procedures are *intended* to limit the chance that ESP-r crashes for other users. Take the time to test and debug interface changes. If an interface dialogue has been updated try it. The automated testing is not focused on interface issues. Open existing models as well as checking that new models or zones or components can be created. If possible, get others involved in the testing. They will almost always provide useful feedback as well as using facilities in unexpected ways.
- Back up work-in-progress which has not yet been committed into the repository.
- Periodically review the log of your branch as well as the development branch to ensure that all relevant commits have been accounted for. Annotating a copy of the subversion log file to indicate which commits have been taken and which are pending is a useful technique.

8.2 Merging changes from the development branch

Merging changes from the development_branch is done via subversion *merge* commands, represented by '3a' in Figure 4. Some human intervention (on the part of developer ND-1 aka fred) will be required if there are conflicts detected during the merge process. If there are no conflicts developer ND-1 should issue a *commit* command immediately after the merge from the development branch. If there are conflicts these should be documented and manually resolved prior to committing the merge.

The development branch (thick line in Figure 4) is the primary channel by which contributions from developers are shared. Changes are only incorporated into the development_branch after a sequence of tests have been passed. Once the changes are in place a message will be sent when the archivist commits changes to the development_branch. Extracts from such a message are shown below:

```
COMMIT LOG MESSAGE
~~~~~~~~~~~~~~~~~~~~

This commit merges the changes c4340 of the sub-branch 'Jon_Hand'
into 'development_branch'.

Summary of changes:
- Further data typing to improve 64 bit implementation.
  Users on 64 bit platforms will notice fewer interface
  (line drawing) glitches.
- Focus on climate data and seasons with data structures
  moved into header files, code consolidation and longer
  climate file name string. Users will not notice this
  but developers will find better documentation,
  additional data typing and more common code.
  . . .
- Move data structures related to integrated performance view
  into a header file and make explicit. Users will not notice
  and developers will find better documentation.
- Correct a glitch which often caused application failure
  in GTK version (occasional with GCC 3.4/4.1 and often with
  GCC 4.3 and newer). Users of Native Windows version will
  notice greater stability.
- Update documentation in the manual/OS folder to reflect
  version 11.7

Testing summary:
- Compiled with GCC 4.1.2 on Ubuntu Linux X11.
- Compiled with GCC 4.3.2 on Cygwin.
- Ran tester.pl against the latest development branch with no reported
  differences!
- Interactive tests of prj and clm and bps and res during development
  stages of the commits that were re-merged. Both X11 and GTK
  versions were tested earlier.

All sub-branches of 'development_branch' should now be synchronized
with 'development_branch'. These actions are to be completed by the owners
of these sub-branches following the instructions included in
src/archive/subversion.trf.

CHANGE-SET SUMMARY
~~~~~~~~~~~~~~~~~~~~

A summary of these changes is available at:
```

```
http://node9.cvsdude.com/trac/espr/esp-r/changeset/4350

CHANGE LOG
~~~~~~~~~~~~~~~~~~~~
   _U  branches/development_branch/
   U   branches/development_branch/src/cetc/h3koutput.F
   U   branches/development_branch/src/climate/climatelist
   U   branches/development_branch/src/esrubld/blibsv.F
   U   branches/development_branch/src/esrubld/input.F
   . . .
   U   branches/development_branch/src/esrurun/Makefile
   U   branches/development_branch/src/esruvew/azalts.F
   U   branches/development_branch/src/include/esprdbfile.h
   A   branches/development_branch/src/include/ipvdata.h
   A   branches/development_branch/src/include/seasons.h
   U   branches/development_branch/src/lib/esp_draw.c
   U   branches/development_branch/src/lib/esru_lib.F
   U   branches/development_branch/src/lib/esru_nox.c
   U   branches/development_branch/src/lib/esru_x.c
   A   branches/development_branch/src/manual/OS/Apple/esp-r_v11.7_osx_precomp.readme.txt
   U   branches/development_branch/src/manual/OS/Apple/instructions_for_esp-r_osx_installer.txt
   A   branches/development_branch/src/manual/OS/Apple/setup
   U   branches/development_branch/src/manual/OS/Apple/setup_osx
   D   branches/development_branch/src/manual/OS/Cygwin/esp-r_v11.6_cygwin_precomp.readme
   A   branches/development_branch/src/manual/OS/Cygwin/esp-r_v11.7_cygwin_precomp.readme
   . . .
   A   branches/development_branch/src/manual/OS/Native_windows/Windows_esp-r_development_may09.rtf
   U   branches/development_branch/src/validation/CEN/13791/shade/ref_fullshd.cfg
   U   branches/development_branch/src/validation/CEN/13791/shade/ref_noshd.cfg
   . . .
   U   branches/development_branch/src/validation/CEN/15265/Test_8/Test_8.cfg
   U   branches/development_branch/src/validation/CEN/15265/Test_9/Test_9.cfg

REVISION HISTORY
~~~~~~~~~~~~~~~~~~~~


Revisions:
- http://node9.cvsdude.com/trac/espr/esp-r/log/?verbose=on

Source tree:
- http://node9.cvsdude.com/trac/espr/esp-r/browser

~~~~~~~~~~~~~~~~~~~~
ESP-r Central is a source code repository that provides access to ESP-r
source code for all users and developers.

-- Provided by, CVSDude, http://cvsdude.com. Professional CVS and SVN outsourcing --
```

The above message includes several sections of interest to developers. The first item is the source branch of the revisions and which revisions of that branch were included. This is followed by a high level summary of the changes that are included. Curious developers could look at the log of the contributing branch for further information. This is followed by a list of the files that have changed (M), added (A), deleted (D). If these files are also files that have been recently modified in the fred branch then it is well worth a closer inspection. It might also be useful to make a backup of the fred branch files that were modified in the development_branch in case there are problems in the merge.

Lets assume that the user is named fred and keeps his source code in /home/fred/Src/cvsdude and to setup for the merge a fresh version of the fred branch will be checked out, a local version of the distribution will be compiled to /home/fred/esru_pre_merge (including databases and example models) and the syntax checked on all of the modules (via the scripts all_fckX11 and checkdiff).

Working with a freshly checked out local copy of the fred branch rather than an existing sandbox prevents glitches.

The command sequence to implement this is:

```
cd
cd Src/cvsdude
svn checkout https://espr.svn.cvsdude.com/esp-r/branches/fred fred_merge
cd fred_merge/src
./Install -d /home/fred/esru_pre_merge --gcc4
```

The title of the message about the development branch included the information that it related to revision 4350. Lets assume that the fred branch has been kept up to date with the development_branch and thus the only

changes needed will be from revision 4349:4350. The command sequence is:

```
cd /home/fred/Src/cvsdude/fred_merge/
svn merge -r 4349:4350 https://espr.svn.cvsdude.com/esp-r/branches/development_branch
```

The process can take some time to complete. There will be messages as files are modified and/or added and deleted from the fred branch to reflect the recent changes in the development_branch. Particular attention should be paid to warnings about conflicts (see a later section for details of how to respond to conflicts).

If you are happy with the merge process then you can commit this merge into your branch (make sure your commit message documents the specific range of revisions taken in and that it brings your branch up to date with a specific revision of the development_branch). You might want to carry out a test Install prior to committing the merge. It is important to commit the merge (and any manual conflict resolutions) separately from other changes.

8.3 Committing changes into your branch

When you checkout your branch into a local sandbox and work in the sandbox nothing that happens in the sandbox can impact what is in the repository of your branch until you specifically commit the changes. Used well, subversion can allow you to try out new ideas and toss them away if you find they do not work and remember them (and potentially share them easily with others) if they do work.

Those who use subversion regularly develop habits which help them to work more quickly, reduce risk from human and machine glitches as well as making life easier for the archivist. The *Work flow within the repository* section has some good hints.

Atomic commits and good documentations of commits are key. If you want to see how others have managed the commit process you could check out one of the developer branches and extract a log of the changes made in that branch:

```
svn log -v https://espr.svn.cvsdude.com/esp-r/branches/Jon_Hand >jwh_log.txt
nedit jwh_log.txt
```

If you have a personal branch you should automatically see the commits made by others. If you want a quick look at what changed in a commit look in the email message for a line that includes:

```
A summary of these changes is available at:

https://espr.trac.cvsdude.com/esp-r/changeset/XXXX
```

where XXXX is a specific number.

Assuming that you made two relate changes, one in esrucom and the other in the lib folder. To commit these changes on a Linux computer or Cygwin or OSX you would issue the following commands:

```
SVN_EDITOR=nedit
export SVN_EDITOR
svn commit lib esrucom
Sending        esrucom/edatabase.F
Sending        lib/esru_lib.F
Transmitting file data ..
Committed revision 4533.
```

The SVN_EDITOR is an environment variable that tells subversion what editor to use so you can type in a message about the commit. Some people prefer to compose their messages in advance and paste the text into an editor.

8.4 Getting your changes into the development_branch

The archivist has several tasks to carry out. A message needs to be composed which describes the change and the process goes more quickly if you prepare a synopsis of your contribution that follows the pattern seen above.

The archivist will also be interested in whether the new contribution will result in any change to the numerical predictions. In the source distribution is a folder named tester and it contains test models and a Perl script which runs hundreds of tests to identify changes in predictions.

One use of the tester.pl script is to exercising both the official development branch version of ESP-r and the current fred branch across all of the models of the testing regime. The second approach is to first generate an archive of the predictions of the development branch (which can be re-used) and then generate an archive of the current fred branch predictions.

To carry out either of these we need to checkout the current version of the development branch, install it to the standard location (/usr/esru) and then run the tester.pl script. The sequence of commands needed for this are shown below (note that the ./tester.pl command needs to be on one line):

```
sudo mkdir /usr/esru
sudo chown fred /usr/esru
sudo chgrp staff /usr/esru
cd Src/cvsdude
svn checkout https://espr.svn.cvsdude.com/esp-r/branches/development_branch
cd development_branch/src
./Install -d /usr/esru --gcc4 --reuse_ish_calcs

cd /home/fred/Src/cvsdude/development_branch/tester/scripts
./tester.pl -v --databases /usr/esru/esp-r /usr/esru/esp-r/bin/bps
  --ref_loc /usr/esru/esp-r/bin/ --test_loc /usr/esru/esp-r/bin/
  --create_historical_archive ESRU_dev_linux.tar.gz
```

To run tester to directly compare the standard version of ESP-r (in /usr/esru) and your version of ESP-r something like the following command (on one line) would be used:

```
cd /home/fred/Src/cvsdude/development_branch/tester/scripts
  ./tester.pl -v --databases /usr/esru/esp-r /usr/esru/esp-r/bin/bps
  /home/fred/esru_pre_merge/esp-r/bin/bps
  --ref_loc /usr/esru/esp-r/bin/ --test_loc /home/fred/esru_pre_merge/esp-r/bin/
```

To compare two archives the command sequence is:

```
cd /home/fred/Src/cvsdude/development_branch/tester/scripts
./tester.pl -v -a ESRU_dev_linux.tar.gz -a ESRU_pre_merge_linux.tar.gz
```

Lets assume that the performance predictions were *close enough* to be considered a match. We can now. To be polite, it is useful to check that what we propose to merge into the development_branch can be done without any conflicts. In preparation for the merge the local copy of the fred branch should be cleaned and it is also a good idea to make a backup of it in case the merge process fails (e.g. a power outage).

```
cd /home/fred/Src/cvsdude/fred_merge/src
make clean
cd /home/fred/Src/cvsdude
tar cf fred_prior_to_merge.tar fred_merge
```

Lets say that that your revisions to be taken are r7888:7895 and r7998:8012. Check out a fresh development_branch to test your contribution on:

```
cd
cd Src/cvsdude
svn checkout https://espr.svn.cvsdude.com/esp-r/branches/development_branch dev_test
cd dev_test
svn merge -r7888:7895 https://espr.svn.cvsdude.com/esp-r/branches/fred
svn merge -r7998:8012 https://espr.svn.cvsdude.com/esp-r/branches/fred
```

If this merge happens without any conflicts then you are good to go! Your next step is to send a message to the archivist with the summary of the changes and testing carried out as well as specific instructions about which revisions in your branch to take.

8.5 Finding differences with other branches

If you want to clearly see the differences between your branch and the current development_branch **before you do the merge** you could use a script to compare the two distributions. In the source distribution bin folder is a script named shortdiff. Copy it into your $HOME/bin folder so you can run it easily. It is given two paths:

```
./shortdiff /home/jon/Src/cvsdude/development_branch/src
 /home/jon/Src/cvsdude/jwh/src
```

The result of running that script will be a file named *mydifferences*. It will include information on which files have differences. You can then use a file difference viewing tool on the two files to check the details of differences.

```
currently comparing /home/jon/Src/cvsdude/Jon_Hand/src and ../../jwh_for_merge/src

cetc
looking at cetc ../../jwh_for_merge/src/cetc
diff cetc/ashp_cooling.F ../../jwh_for_merge/src/cetc/ashp_cooling.F
diff cetc/chemical_properties.F ../../jwh_for_merge/src/cetc/chemical_properties.F
diff cetc/DHW_module.F ../../jwh_for_merge/src/cetc/DHW_module.F
diff cetc/h3k_report_data.F ../../jwh_for_merge/src/cetc/h3k_report_data.F
diff cetc/RE-H2-ctl.F ../../jwh_for_merge/src/cetc/RE-H2-ctl.F
cetc/h3kreports
looking at cetc/h3kreports ../../jwh_for_merge/src/cetc/h3kreports
diff cetc/h3kreports/TReportsManager.cpp ../../jwh_for_merge/src/cetc/h3kreports/TReportsManager.cpp
diff cetc/h3kreports/TVariableData.cpp ../../jwh_for_merge/src/cetc/h3kreports/TVariableData.cpp
diff cetc/h3kreports/TVariableData.h ../../jwh_for_merge/src/cetc/h3kreports/TVariableData.h
esrubld
looking at esrubld ../../jwh_for_merge/src/esrubld
diff esrubld/casual.F ../../jwh_for_merge/src/esrubld/casual.F
diff esrubld/complex_fenestration.F ../../jwh_for_merge/src/esrubld/complex_fenestration.F
diff esrubld/solar.F ../../jwh_for_merge/src/esrubld/solar.F
diff esrubld/spmatl.F ../../jwh_for_merge/src/esrubld/spmatl.F
 . . .
```

8.6 Adding a new subroutine to ESP-r

To clarify the proceeding discussions, lets follow the sequence of tasks involved in introducing a new feature into ESP-r. Below is a fragment of code from esrucom/edatabase.F that checks if three real numbers are all very close to zero in order to determine if data scanned from a file is correct.

```
C If all values are still 0.0 then not an actual element.
        CALL ECLOSE(DBCON,0.0,0.001,CLOSE1)
        CALL ECLOSE(DBDEN,0.0,0.001,CLOSE2)
        CALL ECLOSE(DBSHT,0.0,0.001,CLOSE3)
        if(CLOSE1.and.CLOSE2.and.CLOSE3)then
          write(outs,'(A,I3,A,2i4)') ' Material db reference ',IEL,
     &                          ' has no data, or all zero...',IR,IFMAT
          call usrmsg(outs,' Please check your selection! ','W')
          ier=1
          return
        endif
```

The subroutine eclose is used hundreds of times to test if numbers are close to a specified value. There are many places where a vector e.g. XYZ requires testing to see if all are within a tolerance. The project is thus to create a subroutine that tests one vector against another for a given tolerance and return a single logical indicator. This new subroutine call would look like:

```
C If all values are still 0.0 then not an actual element.
        call eclose3(DBCON,DBDEN,DBSHT,0.0,0.0,0.0,0.001,CLOSE4)
        if(CLOSE4)then
          write(outs,'(A,I3,A,2i4)') ' Material db reference ',IEL,
     &                          ' has no data, or all zero...',IR,IFMAT
          call usrmsg(outs,' Please check your selection! ','W')
          ier=1
          return
        endif
```

The implementation of the subroutine is as follows:

```
C ******************* ECLOSE3
C ECLOSE3 allows two real vectors R1 & R2  & R3 to be checked for closeness
C to a given tolerance TOL with X1 X2 & X3 and returns CLOSE = .TRUE. or .FALSE.
      SUBROUTINE ECLOSE3(R1,R2,R3,X1,X2,X3,TOL,CLOSE)
      LOGICAL CLOSE,CLOSEA,CLOSEB,CLOSEC
      real R1,R2,R3  ! the vector to test
      real X1,X2,X3  ! the vector to compare against
      real TOL       ! how close
```

```
      call eclose(R1,X1,TOL,CLOSEA)   ! test first pair
      call eclose(R2,X2,TOL,CLOSEB)   ! test second pair
      call eclose(R3,X3,TOL,CLOSEC)   ! test third pair
      if(CLOSEA.and.CLOSEB.and.CLOSEC)then
        CLOSE=.true.  ! all are close
      else
        CLOSE=.false. ! at least one is different
      endif

      RETURN
      END
```

Note that each of the parameters of the subroutine is typed and documented as is the logic. As this subroutine would be useful in many ESP-r modules and is of the same type of utility as the subroutine eclose the logical place for it is within lib/esru_lib.F next to the subroutine eclose. And we should also provide a synopsis at the top of esru_lib.F

After editing the file esru_lib.F we can use a svn command **svn status** to see what files have changed and **svn diff** to show changes svn has noticed:

```
svn status lib
M      lib/esru_lib.F

svn diff lib/esru_lib.F
Index: lib/esru_lib.F
===================================================================
--- lib/esru_lib.F   (revision 4490)
+++ lib/esru_lib.F   (working copy)
@@ -83,6 +83,7 @@
 C  DOT3(a,b,product) Return dot product of two vectors a & b.
 C  ZEROS:   Clear a 4x4 array prior to doing vieweing transforms.
 C  ECLOSE:  Checks tolerance between two real numbers.
+C  ECLOSE3: Checks tolerance between two real vectors (3 numbers).
 C  ESIND:   Function returning SIN of angle where angle is given in degrees.
 C  ECOSD:   Function returning COS of angle where angle is given in degrees.
 C  ETAND:   Function returning TAN of angle where angle is given in degrees.
@@ -2279,6 +2280,27 @@
       RETURN
       END

+C ******************* ECLOSE3
+C ECLOSE3 allows two real vectors R1 & R2  & R3 to be checked for closeness
+C to a given tolerance TOL with X1 X2 & X3 and returns CLOSE = .TRUE. or .FALSE.
+      SUBROUTINE ECLOSE3(R1,R2,R3,X1,X2,X3,TOL,CLOSE)
+      LOGICAL CLOSE,CLOSEA,CLOSEB,CLOSEC
+      real R1,R2,R3  ! the vector to test
+      real X1,X2,X3  ! the vector to compare against
+      real TOL       ! how close
+
+      call eclose(R1,X1,TOL,CLOSEA)   ! test first pair
+      call eclose(R2,X2,TOL,CLOSEB)   ! test second pair
+      call eclose(R3,X3,TOL,CLOSEC)   ! test third pair
+      if(CLOSEA.and.CLOSEB.and.CLOSEC)then
+        CLOSE=.true.  ! all are close
+      else
+        CLOSE=.false. ! at least one is different
+      endif
+
+      RETURN
+      END
+
 C ******************* ESIND
       FUNCTION ESIND (DEG)
 C ESIND: Returns SIN of angle where angle is given in degrees.
```

The lines with a + at the start have been added. The same procedure can be carried out for changes in esru-com/edatabase.F

```
svn status esrucom
?      esrucom/edatabase.F-
M      esrucom/edatabase.F

svn diff esrucom/edatabase.F
Index: esrucom/edatabase.F
```

```
===================================================================
--- esrucom/edatabase.F   (revision 4439)
+++ esrucom/edatabase.F   (working copy)
@@ -891,12 +891,13 @@
      &                  DRV,TITL,PNAM)

       CHARACTER PNAM*72,TITL*72,outs*124
-      logical close1,close2,close3
+      logical close1,close2,close3,close4

       IER=0
       close1=.false.
       close2=.false.
       close3=.false.
+      close4=.false.

 C The record in the material db is IEL + 1 UNLESS the
 C reference is to material db 0 (air).
@@ -913,9 +914,14 @@
      &       DRV,PNAM

 C If all values are still 0.0 then not an actual element.
+
+C test new code
+        call eclose3(DBCON,DBDEN,DBSHT,0.0,0.0,0.0,0.001,CLOSE4)
+    write(6,*) 'close4 test is ',close4
         CALL ECLOSE(DBCON,0.0,0.001,CLOSE1)
         CALL ECLOSE(DBDEN,0.0,0.001,CLOSE2)
         CALL ECLOSE(DBSHT,0.0,0.001,CLOSE3)
+    write(6,*) 'separate eclose calls ',close1,close2,close3
         if(CLOSE1.and.CLOSE2.and.CLOSE3)then
           write(outs,'(A,I3,A,2i4)') ' Material db reference ',IEL,
      &                           ' has no data, or all zero...',IR,IFMAT
```

The status command indicates a ? prior to a file named edatabase.F-. Subversion does not know about that file because the developer has not added it to the repository. This is for a good reason - the developer has made a backup of the file edatabase.F prior to editing it. Pedantic? Not really.

What is actually seen in the diff is a version of edatabase.F which is being used for testing - it contains both the new call and the previous logic and some write statements to indicate the values during testing. Of course one could avoid the use of write statements via the use of a debugger, setting break points and using the debugger's ability to print out variables. The requirement for testing is in no way pedantic. Even experienced coders will make typographic errors which are not detected by compilers.

Once interactive testing of the new facility has been undertaken users can either comment out debug statements or remove them. And a subsequent svn diff shows:

```
svn diff edatabase.F
Index: edatabase.F
===================================================================
--- edatabase.F (revision 4439)
+++ edatabase.F (working copy)
@@ -891,12 +891,13 @@
      &                  DRV,TITL,PNAM)

       CHARACTER PNAM*72,TITL*72,outs*124
-      logical close1,close2,close3
+      logical close1,close2,close3,close4

       IER=0
       close1=.false.
       close2=.false.
       close3=.false.
+      close4=.false.

 C The record in the material db is IEL + 1 UNLESS the
 C reference is to material db 0 (air).
@@ -913,10 +914,8 @@
      &       DRV,PNAM

 C If all values are still 0.0 then not an actual element.
-        CALL ECLOSE(DBCON,0.0,0.001,CLOSE1)
-        CALL ECLOSE(DBDEN,0.0,0.001,CLOSE2)
-        CALL ECLOSE(DBSHT,0.0,0.001,CLOSE3)
-        if(CLOSE1.and.CLOSE2.and.CLOSE3)then
```

```
+        call eclose3(DBCON,DBDEN,DBSHT,0.0,0.0,0.0,0.001,CLOSE4)
+        if(CLOSE4)then
           write(outs,'(A,I3,A,2i4)') ' Material db reference ',IEL,
      &                          ' has no data, or all zero...',IR,IFMAT
           call usrmsg(outs,' Please check your selection! ','W')
```

The dash in the first column signals that lines have been removed. Is there anything else required? The way to find out if there are additional coding tasks is to compile the prj module and run a syntax check on the code.

```
all_fckX11
  . . .
      - program unit: EPKMLC

      - program unit: ERPCDB
  CLOSE1
(file: edatabase.F, line:     894)
**[323 I] variable unreferenced
  CLOSE2
(file: edatabase.F, line:     894)
**[323 I] variable unreferenced
  CLOSE3
(file: edatabase.F, line:     894)
**[323 I] variable unreferenced

      - program unit: EMKAMLD

      - program unit: EROPTDB
 . . .
```

The all_fckX11 script (found in the source distribution bin folder) runs the syntax checking tool for the code distribution. And in the section related to edatabase.F there are three unreferenced variables. The revised logic does not use CLOSE1, CLOSE2 or CLOSE3 so we can limit future confusion by removing these prior to commiting the code. After saving the file svn indicates:

```
svn diff  edatabase.F
Index: edatabase.F
===================================================================
--- edatabase.F (revision 4439)
+++ edatabase.F (working copy)
@@ -891,12 +891,10 @@
      &                  DRV,TITL,PNAM)

       CHARACTER PNAM*72,TITL*72,outs*124
-      logical close1,close2,close3
+      logical close4

       IER=0
-      close1=.false.
-      close2=.false.
-      close3=.false.
+      close4=.false.

 C The record in the material db is IEL + 1 UNLESS the
 C reference is to material db 0 (air).
@@ -913,10 +911,8 @@
      &       DRV,PNAM

 C If all values are still 0.0 then not an actual element.
-        CALL ECLOSE(DBCON,0.0,0.001,CLOSE1)
-        CALL ECLOSE(DBDEN,0.0,0.001,CLOSE2)
-        CALL ECLOSE(DBSHT,0.0,0.001,CLOSE3)
-        if(CLOSE1.and.CLOSE2.and.CLOSE3)then
+        call eclose3(DBCON,DBDEN,DBSHT,0.0,0.0,0.0,0.001,CLOSE4)
+        if(CLOSE4)then
           write(outs,'(A,I3,A,2i4)') ' Material db reference ',IEL,
      &                          ' has no data, or all zero...',IR,IFMAT
           call usrmsg(outs,' Please check your selection! ','W')
```

The steps above confirm that coding adheres to the ESP-r coding guide. The nature of the change does not alter predictions made by the simulator. The interactive test indicated that the logic was equivalent. Running the tester.pl script would, in this case, be pedantic if the developer was planning on making additional changes prior to asking the archivist to update the development_branch. If, however, this was the last change to be included in

an update to the development_branch the archivist would require the tester.pl script to be run.

Depending on the computer the standard *tester.pl* tests in the *tester* folder can take several hours to run. Those who need a quicker test to use as coding progresses can also use the test scripts in the *validation/benchmark/QA/model/cfg* and *validation/benchmark/QA/model1.1/cfg* folders. This step is represented by '3b' in Figure 4. An example fragment of the report generated is shown below:

```
tester.pl Test Report
Testing commenced on 21/01/2009 18:56:47
After grand merge of dev into Jon_hand and re-merge of pending changes.

Test parameters:
 - Test suite path:        /home/jon/Src/cvsdude/development_branch/tester/test_suite/
 - Abbreviated runs:       disabled

Test System Information:
 - Username:               jon
 - Host:                   osiris
 - Platform:               i686
 - Operating system:       Linux:2.6.24-23-generic

bps binaries:
 - Path:                   (reference)  osiris:/home/jon/esru_dev/esp-r/bin/bps
                           (test)       osiris:/home/jon/esru_jwh_tm/esp-r/bin/bps
 - SVN source:             (reference)  development_branch@r3801 (locally modified)
                           (test)       Jon_Hand@r3794 (locally modified)
 - Compilers:              (reference)  gcc-3.4/g++-3.4/g77-3.4
                           (test)       gcc-3.4/g++-3.4/g77-3.4
 - Graphics library:       (reference)  X11
                           (test)       X11
 - XML support:            (reference)  Supported
                           (test)
 - Modification date:       (reference)  2009-01-21 18:36:04.000000000 +0000
                           (test)       2009-01-21 17:57:14.000000000 +0000
 - MD5 Checksum:           (reference)  94a6931eb3497c4dc16eff74fc33384d
                           (test)       762a08f684ba9913d698bff39423ab75
                                        (files differ)

Compared output: .csv .data .h3k .xml files
Overall result: Pass.

Summary of test results:
  - '-' indicates test case passes
  - 'X' indicates test case fails
  - '.' indicates files were not produced, or were not compared
-------------------------------------------------------------------------------------------------
Folder                        Model                          .xml  .data .csv  .h3k  overall dt-CPU(%)
-------------------------------------------------------------------------------------------------
Annex42_fuel_cell             SOFC_constant                   -     -     -     -      -     -3.3
alberta_infil_model           basic_AIM_MAX                   -     -     -     -      -      0.23
alberta_infil_model           basic_AIM_MIN                   -     -     -     -      -      0.92
alberta_infil_model           basic_AIM_TIGHT                 -     -     -     -      -     -0.68
alberta_infil_model           basic_AIM_reference             -     -     -     -      -     -8.4e-11
alberta_infil_model           detailed_AIM_MAX                -     -     -     -      -      0.57
alberta_infil_model           detailed_AIM_MAX_ver1           -     -     -     -      -     -1.7
alberta_infil_model           detailed_AIM_MIN                -     -     -     -      -      0.14
alberta_infil_model           detailed_AIM_MIN_ver1           -     -     -     -      -     -1.3
alberta_infil_model           detailed_AIM_TIGHT              -     -     -     -      -     -0.28
alberta_infil_model           detailed_AIM_TIGHT_ver1         -     -     -     -      -      0.28
alberta_infil_model           detailed_AIM_reference          -     -     -     -      -      0.42
basesimp                      basic_BSM_MAX                   -     -     -     -      -     -0.66
basesimp                      basic_BSM_MAX_MooreModel        -     -     -     -      -     -0.22
basesimp                      basic_BSM_MIN                   -     -     -     -      -      2.5
. . .
plt_zone_heat_gain_coupling   plt_multizone_zone_gain_test    -     -     -     -      -      0.37
pv_example                    pv_2000Glo                      -     -     -     -      -      0.53
therm_man_test                h2-ctrl                         -     -     -     -      -      1.4
type-999                      gc80                            -     -     -     -      -     -0.68
-------------------------------------------------------------------------------------------------

Parameter dt-CPU describes the percent change in simulation CPU
runtime between the reference and test versions of bps.
  - When different versions of bps are exercised on the same
    machine, dt-CPU is a measure of the relative efficieny of
    the ESP-r source code.
  - When the same version of bps is exercised on different
    machines, dt-CPU is a measure of the comparative performance
```

```
    of ESP-r on different hardware and operating systems.

No differences were found in XML output. Detailed report unnecessary.
```

If the report generated by the testing script indicates no differences with the *development_branch* then the new contributions are ready for the next step. If there are differences reported then discussions must be held with the Archivist to determine whether the differences are expected or have identified an error. A fragment from a report with differences is shown below:

```
. . .
basesimp                        detailed_BSM_MIN             X     X     X     X     X       1.9
basesimp                        detailed_BSM_reference       -     X     X     X     X       2.5
. . .
------------------------------------------------------------

TEST CASE detailed_BSM_MIN (basesimp)
 - Folder:          basesimp
 - Model:           detailed_BSM_MIN.cfg
 - MAX error (W)    1.8057  W  (0.1756 %) - observed in: building:all_zones:thermal_ld:net:month_01 (min)
 - MAX error (oC)   0.14092 oC (0.69135%) - observed in: building:zone_05:surf_06:temp:month_01 (max)
-------------------------------------------------------------------------------------------------------
Elements exhibiting differences                              Units |Relative   Absolute   Reference  Test
                                                                   |Diff (%)   Diff       Value      Value
-------------------------------------------------------------------------------------------------------
building:all_zones:supplied_energy:heating:annual (min)        W |  0.45335   -1.8056    398.28   400.09
building:all_zones:supplied_energy:heating:month_01 (min)      W |  0.45335   -1.8056    398.28   400.09
building:all_zones:supplied_energy:net_flux:annual (min)       W |  0.45335   -1.8056    398.28   400.09
building:all_zones:supplied_energy:net_flux:month_01 (min)     W |  0.45335   -1.8056    398.28   400.09
building:all_zones:thermal_loads:heating:total:annual (min)    W |   0.1756   -1.8057    1028.3   1030.1
building:all_zones:thermal_loads:heating:total:month_01 (min)  W |   0.1756   -1.8057    1028.3   1030.1
building:all_zones:thermal_loads:net:annual (min)              W |   0.1756   -1.8057    1028.3   1030.1
building:all_zones:thermal_loads:net:month_01 (min)            W |   0.1756   -1.8057    1028.3   1030.1
building:zone_05:surface_06:temperature:annual (max)           oC|  0.69135   -0.14092   20.383   20.524
building:zone_05:surface_06:temperature:month_01 (max)         oC|  0.69135   -0.14092   20.383   20.524
-------------------------------------------------------------------------------------------------------
```

The tester.pl script logic and reporting facilities identify which performance data is beyond the tolerance of the test as well as the location within the model. This is a substantial aide in the identification of offending code as well as faults in test models.

If changes in the source code introduce a new facility consider how this might be tested. Inclusion of new models in the tester/test_suite can ensure that future changes do not have an untended on the new facility.

Models for inclusion in test/test_suite need to be configured in specific ways: first, references to standard databases should be in the form of /usr/esru/esp-r/databases or should point to model specific databases. Model configuration files should include a simulation preset so that it can be run without user intervention. The name of the preset must be *test*. The tester process includes the creation of XML output files and this requires the inclusion of a file named *input_xml* in the test model cfg folder. Look in other test_suite models for examples of this xml file.

Once the tests are passed the developer would send an email to the Archivist with a summary of the changes to be taken into the development branch and a list of the revisions to take and the name of the branch. This summary will form part of the eventual release notes and should be formatted and checked for spelling.

Ideally, code submissions should be related to one concept. For example changes which implement a new interface to an ideal zone control should be made separately from code that extends a statistical report in the results analysis module. A change in a data structure should be made to all associated code blocks in one commit if possible so that the ESP-r distribution is consistent. If there are multiple issues in the commit then each should be noted separately.

The documentation sent to the archivist is a *summary of the contribution* which will form part of the release note which will be published when ESP-r versions change. Readers will be looking for the significance and applicability of the changes. Avoid detail, as interested parties can look at the detailed documentation in your branch. Be sure to include the following in your summary:

- the nature of the change
- whether it is new functionality for the user, repairs a bug, tidies code, or is a work in progress
- if new functionality is provided for the user, then indicate whether this affects the Simulator, Project Manager, Results Analyzer, etc.

- what users might notice
- what developers might notice
- if a new method has been implemented provide a citation to a thesis or a paper that describes its theoretical basis
- if a bug has been repaired, then indicate which application this affects and the nature of the change
- if a bug fix alters simulation results describe what the changes are and what types of models would be impacted
- if this change tidies code or alters a data structure but does not alter functionality indicate this
- if the submission is work in progress indicate whether it is safe for others to use
- IMPORTANT: Use proper grammar and spelling!

An example of such a please-take-this message is shown below:

```
Summary of changes:
- A sequence of commits which merge the functionality of the tdf
  module into the project manager. This allows closer ties between
  information about a model and sets of temporal data. The first
  benefit is in support of the UK NCM activity data.
- After testing shifted source code in esrutdf to esrucom and/or
  esruprj and updated Install script and Makefile.

  Uses will notice fewer keystrokes are required to manage temporal
  data and there is somewhat more space for the menu display although
  there is a bit less space for graphic display.

- Added lighting requirements to UK NCM description.

- Added constructions needed for UK NCM models.

Testing summary:
- Tester.pl temporal data not used in tester.pl models. Testing of the
  validation/QA/benchmark model indicates identical predictions.

- pre-commit and post-commit syntax checks on full source of all modules
- Compile Solaris F90 X11
- Compile Linux GCC 3.4 X11
- Compile OSX GCC 3.3 X11
- Single step debug sessions reading existing temporal data and editing
  within the project manager followed by interactive tests of editing
  data and displaying it.
- Interactive tests of new UK NCM facilities and databases.

Revisions to take from prj_dev branch:
R2504:2515
R2519:2530
R2533:2535
```

The Archivist may merge these commits directly into the development branch (this is represented by ’3c’ in Figure 4. When the commits have been merged into the development branch an email (based on the summary sent to the Archivist) will be sent to the development community advising others to merge the newly updated development branch into the sub-branches (the cycle begins again). Note that the developer who supplied the changes should not merge those same changes back into their own branch.

8.7 Additional steps for complex developments

Where the number of changes are substantial, or there is a risk of a conflict the developer may be asked to perform additional steps to ease the work of the Archivists. An example of this was discussed in the section Getting your changes into the development_branch.

This is represented by ’1c’ in Figure 4. If this test merge is successful the Archivist is notified with the standard summary and instructions about what to take into the development branch. The Archivist will then merge these changes into the development branch (’1d’ in Figure 4) and notify the development community.

8.7.1 The Nuke & Pave approach

If the test merge results in conflicts then an alternative procedure is required which involves forcing the fred branch to be exactly like the development_branch and then re-introducing each of the commits that have been pending, running the tests and then doing a single commit which includes all changes and the resolution of any commits. In the slang of subversion this is a 'nuke and pave over' option. It places an extreme burden on the contributor and there are several points where difficulties may arise.

Before starting this process it is necessary that no other changes be made to the development_branch. This involves freezing the development_branch for a brief period, performing the 'nuke and pave' followed by merging pending commits within the users branch and resolving conflicts and committing all changes and resolves as a single commit which the Archivist takes.

The steps shown below are extracts from an actual 'nuke and pave' and you will need to alter the details to fit your branch and the details of the commits that you are attempting to unify. The process begins with checking out a fresh version of the developers branch into a new sandbox jwh_for_merge and then backing it up and performing the grand merge (nuke and pave). If the grand merge fails remove the jwh_for_merge folder and extract the tar file.

```
svn checkout https://espr.svn.cvsdude.com/esp-r/branches/Jon_Hand jwh_for_merge
tar cf jwh_before_merge.tar jwh_for_merge
cd jwh_for_merge
svn merge https://espr.svn.cvsdude.com/esp-r/branches/Jon_Hand
  https://espr.svn.cvsdude.com/esp-r/branches/development_branch ./
....time passes....
svn commit
```

Depending on network speed this kind of merge command can take several hours to complete and often there is little or no feedback while svn is processing the command. If there are network faults the process may abort and essentially what you need to do is remove the local sandbox and restore from the before-merge tar file and try again. It is unlikely that there will be conflicts (if so they will need to be resolved and documented). At the end of the process commit the merge with an explanitory note.

The next steps are to merge back in the separate commits that were pending. Hopefully you would have been taking notes as the development work progressed and the log kept by subversion about your branch will provide the details. Lets say you wanted a log for recent commits to a particular branch (from revision 4320 to revision 4340) - the command would be:

```
svn log -v -r 4320:4340 https://espr.svn.cvsdude.com/esp-r/branches/Jon_Hand > 4340a.txt
```

Double check the revision numbers prior to giving each of the merge-back commands. Again it can be helpful to make a tar file of the sandbox at the start of the process. The following are an extract from notes taken as the pending commits were merged back into the branch.

```
Re-establish earlier code:
svn merge -r 4126:4127 https://espr.svn.cvsdude.com/esp-r/branches/Jon_Hand
no conflicts

svn merge -r 4134:4135 https://espr.svn.cvsdude.com/esp-r/branches/Jon_Hand
no conflicts

svn merge -r 4135:4136 https://espr.svn.cvsdude.com/esp-r/branches/Jon_Hand
no conflicts

svn merge -r 4136:4137 https://espr.svn.cvsdude.com/esp-r/branches/Jon_Hand
no conflicts

svn merge -r 4151:4152 https://espr.svn.cvsdude.com/esp-r/branches/Jon_Hand
C    src/esrue2r/radcmds.F - minor edit
 svn resolved radcmds.F
C    src/esruish/graph.F  - minor edit
svn resolved graph.F  - minor edit
C    src/esrudfs/visvec.F
svn resolved visvec.F
 . . .
```

Note that there were a number of ranges of commits that were required. After each merge it was noted whether the merge was clean or the nature of the conflict. The command 'svn resolved' are necessary after you have corrected the conflict to inform svn of the changes. Such notes may seem pedantic, however, they become your record of actions so that you do not later attempt to merge the same changes a second time.

To clarify whether merges have worked some developers do a visual comparison between the code and an untouched source code sandbox (hence the recommendation to checkout a fresh source code sandbox so that earlier sandboxes are available for comparison).

Remember, good practice is for the development community to update their own branches as soon as practical with changes in the *development_branch*. Failure to update can cause problems when it comes to preparing code for submission to the Archivist. Good practice also includes making a backup copy of local sand boxes prior to doing any major merges.

9 New release testing

The preparation for creating a new official distribution often requires a sequence of commits from several developers who are involved in carrying out the tests. For this task a temporary branch is often created for use by the testing team ('4a'). Issues identified and corrected by the testers will be merged into the temporary branch ('4b'). Once the Archivist is satisfied with the coding contributions, and the quality assurance tests are run on the temporary branch the code is merged into the development branch ('4c') and a new distribution is created ('2a') and announced.

The time delays inherent in the process can impact development teams who need to rapidly progress work. One approach is to create a branch which several people have access to so that each member of the team can commit and update from changes made within the team. Interactions with the Archivist and other members of the development community follow the standard pattern.

Occasionally a change made in one developers branch is of immediate interest to another. While is possible to merge such a change between branches, cross-branch merges should be clearly documented to ensure that only the originator of the change commits it to the development branch.

9.1 Audit Trail

The audit trail built into SVN become, in effect, a blog for the community as contributions are committed to the repository. Each commit includes a message identifying what changed, what the impact on users and developers is and how the change was tested. The commit message for the addition of subroutine eclose3 provided as part of the commit is shown below:

```
- Introduce a library subroutine to test closeness
  of to vectors (3 reals) returning a logical value.
  Update logic in esrucom/edatabases.F to use this
  new facility.  Users will not notice this change.

Testing
- Compile Linux GCC 4.1 X11
- Interactive test followed by removal of debug statements
- Pre-commit syntax checks
- Should not alter any predictions

--This line, and those below, will be ignored--

M    lib/esru_lib.F
M    esrucom/edatabase.F


svn commit lib esrucom
Sending        esrucom/edatabase.F
Sending        lib/esru_lib.F
Transmitting file data ..
Committed revision 4533.
```

The revision number returned after the commit command is unique and should be referenced when the developer is requesting that the development_branch be updated.

Each of the branch owners are notified of changes as they occur. The message composed during the commit forms one part of the notification. The notification also includes web links so that the differences in the code can be viewed.

9.2 Documentation for users

Viewed from the outside, the provision of documentation for users of ESP-r is of variable quality. Before interfaces offered contextual help, user manuals were the primary point of reference. The advent of contextual help provides an alternative to reference manuals but also competes for scarce resources in order to populate the hundreds of dialogues and scores of menus within ESP-r. One task that many developers forget is to update the contextual help messages to reflect the current facilities offered by the interface.

The above examples of the use of subversion have been taken from those in the development community who are already adept at doing-the-dance. For those who are joining the ESP-r community there is a need for additional guidance. The following discussion of svn commands may be helpful.

10 Subversion Guide

The following section covers many of the tasks that developers are expected to accomplish via the source code control environment subversion (abbreviated svn).

Subversion maintains a record of changes to the source code and provides a means for dealing with concurrent changes to source files. It also allows developers to contribute to the ESP-r code base. This section describes how to use Subversion to checkout a working copy, to add and remove files, and to commit files back to the ESP-r code base. It is also useful to read the other documents in the *archive* folder which complement this document.

10.1 Revision history

This document is under versioning control, and suggestions and contributions are strongly encouraged. The troff-formatted source file for the latest version is found in the folder src/archive/ESP-r_developers_doc_text.txt

To generate an A4 postscript document from this file via the groff suite of tools (available for many operating systems) issue the following command:

```
cat ESP-r_developers_doc_text.txt | eqn | tbl | groff -mms -dpaper=a4 -P-pa4 ESP-r_developers_doc.ps
```

10.2 What is a Subversion Repository?

A Subversion repository is both a storage area for project source code and a tracking system for source code changes. It keeps track of the history of changes to every file and directory contained within it. The ESP-r Central Subversion repository ensures that all developers and users have the most up-to-date version of the ESP-r source code. Subversion also allows for developers to commit their bug-fixes and/or enhancements to the repository.

For a complete description please refer to Chapter 2 of Versioning control with Subversion, by Collins-Sussman, Fitzpatrick and Pilato:

http://svnbook.red-bean.com/en/1.1/ch02.html

10.3 Obtaining a Subversion Client

Command line clients for various operating systems can be downloaded from the subversion website:

http://subversion.tigris.org/project_packages.html

There are also a number of graphical user interfaces that can be used to access a Subversion repository. Due to variations in their use, they are not documented here. Most of these tools also provide a command line mode, and this documentation is still applicable.

10.4 Anonymous access for Non-Developers

If you do not plan to modify ESP-r source code, you may download a working copy using anonymous access. This is convenient for students and professionals who simply wish to download and compile the latest version of ESP-r or make minor modifications such as increasing the maximum geometric complexity by adjusting parameters in the source code header (also known as include) files.

The following command will download ESP-r's source code from the ESP-r Central repository to the current working directory of your local computer:

```
svn checkout  https://espr.svn.cvsdude.com/esp-r/branches/development_branch
```

With this working directory, you will be able to compile ESP-r on your own computer. Though you will also be able to alter the source code, you, will be unable to make changes to the ESP-r Central repository.

If you're contemplating modifying ESP-r, are strongly encouraged to obtain a developer's account. The GNU public license includes a provision that changes you make to ESP-r should be shared with the community and such contributions are made via subversion commands described in this document and the other documents in the *archive'* folder.

10.5 ESP-r Development with Subversion.

If you wish to make changes to the ESP-r Central repository, you must become familiar with the concepts of working on ''branches'', merging your changes back into your branch in the repository and documenting your work so that others in the community can take advantage of the changes that you contribute. These concepts are not trivial, and it is very important that you become comfortable with them.

For a complete description of branches and merging, please refer to chapter 4 of Versioning control with Subversion, by Collins-Sussman, Fitzpatrick and Pilato:

http://svnbook.red-bean.com/en/1.1/ch04.html

ESP-r development occurs on separate ''sub-branches'' that are assigned to individual contributors or teams of contributors. Contributors modify and commit their code on to these sub-branches, where they can be inspected by others. After completing a rigorous testing process, the ESP-r archivist merges contributions from a sub-branch on to the main development branch, where they can be accessed by all developers.

A complete description of the ESP-r Subversion branch structure is available in the document ''Structure of the ESP-r source code archive'' which also includes further suggestions for how development work is managed within the ESP-r development community.

10.6 Obtaining a Developer's Account and Sub-branch

In order to work on a sub-branch, you must first have an ESP-r developer's account and a sub-branch name assigned to you. To obtain an account and sub-branch, contact Alex Ferguson (aferguso@nrcan.gc.ca).

10.7 Checking out a Sub-branch

You must perform a repository ''checkout'' to obtain a working copy of your sub-branch. A ''checkout'' will download the module into your current working directory, where you can compile, alter the source code, and ''commit'' your changes back to your developer-specific sub-branch for others to view. To perform a checkout, use the following command:

```
svn checkout https://espr.svn.cvsdude.com/esp-r/branches/<sub-branch_name>
```

Provide your developer's account name and password when prompted.

If you are working on a computer with Cygwin or the Native Windows development environment (MSYS and MinGW) you can also use an extension to Windows Explorer called Tortoisesvn from <http://tortoisesvn.tigris.org>. Tortiosesvn allows you to right click on folders in Windows Explorer to accomplish many svn commands. For the compile cycle you will also need a command line svn enviroment - this is available under Cygwin but you will have to acquire a command line Subversion client such as is available from CollabNet <http://www.open.collab.net/downloads/subversion/>.

Note that changes that you make in your working directory are NOT recorded in your branch of the repository until you issue a relevant subversion command.

10.8 Common Subversion Commands

There are many commands available in Subversion; the following are the most common commands you will use. For a more extensive list, please refer to Chapter 3 of the book Versioning control with Subversion:

http://svnbook.red-bean.com/en/1.1/ch03s05.html

None of these commands will affect other developer-specific sub-branches, they only affect the sub-branch you have been assigned to work with.

ADD and DELETE files

Use these commands to schedule adding/removing files or directories to/from the sub-branch you are working on. Additions and deletions will only take effect in the repository once you perform a ''commit'' command.

```
svn add <path_to_file_to_be_added>
svn delete <path_to_file_to_be_deleted>
```

Check the STATUS of your workspace.

This command will list all the files you have changed relative to the sub-branch you are working on, which is handy to use before a ‘‘commit’’ or an ‘‘update’’.

```
svn status <directory_or_filename>
```

If the status list is long you may wish to re-direct the output to a file named current_status.txt use the following command:

```
svn status <directory_or_filename> >current_status.txt
```

If you see a file marked with a ’?’ in the status list this signals that it is not known within the repository. If you want the file to be known then issue the following sequence of commands:

```
svn add <path_to_file_or_folder_to_be_added>
svn commit <path_to_file_or_folder_to_be_added>
```

If the file or files to be added are the only pending tasks then you could issue a more general command:

```
svn add <path_to_file_or_folder_to_be_added>
svn commit
```

Remember that files within the working directory which are not part of the repository risk being lost. Some files should not be included in the repository - for example, object files and executables created during the compile process are not part of the repository. Typically only the ASCII version of databases are included in the repository (binary versions are created during the Install process).

10.8.1 Update files/directories of your workspace.

While you’ll initially work on your own personal branch, you may also be involved in collaborative projects requiring several developers to share the same branch. In these projects, you will need to periodically update your working copy with changes other developers have committed to the project branch. The ‘‘update’’ command will update your local copy with any changes that other developers have committed to the project branch since your last checkout/update. *But be careful!* It automatically merges code into your files, so inspect all updated files and ensure your code still works correctly. Some developers create a local archive or backup copies of files which are work-in-progress prior to issuing ‘‘update’’ commands.

```
svn update  <directory_or_file_to_be_updated>
```

A *conflict* may occur if changes in the repository affect the same files you’ve modified in your local copy. Conflicts are discussed in detail below.

COMMIT your changes to the repository.

This command will commit all file changes, as well as Adds and Removes, to the branch you are working on. See the repository document for further advise on how to plan your commits. Only valid ESP-r developer account holders can use this command to update their branch of the repository or joint branches which they may be working on. Prior to doing a commit set an environment variable to tell svn the editor you prefer to use to document your commit.

```
SVN_EDITOR=nedit
export SVN_EDITOR
svn commit <directory_or_filename>
```

A text editor will be opened after you issue the above command. Enter a detailed message that elaborates the reasons for your coding change/addition, the intent of your code, and the testing that you have conducted. You should indicate in detail what impact this change has upon ESP-r functionality, and in particular, the impact it has upon calculation results. This message will be permanently recorded in ESP-r Central’s repository log and will act as a reference for other developers and for yourself in the future. Use proper sentence structure and grammar to effectively communicate this critical information to your colleagues.

Within 24 hours of committing your changes, you will receive an automatically-generated test report comparing the new version you’ve submitted with the previous version on your sub-branch. This test report will tell you if your new version compiles correctly in various configurations, and will also highlight any questionable syntax and potentially erroneous code introduced by your commit. Note that this report is based on differences

between your current commit and the previous state of your branch. To review the full syntax report you will have to run the syntax checking software yourself.

10.8.2 Conflicts

Conflicts arise when changes received from another developer, during an update or merge, overlap with local changes that you have in your working copy. You must resolve these conflicts before committing your changes to the repository. Subversion will flag files in conflict with a "C" after an update or merge:

```
svn update
---------------------- (OUTPUT) ---------------------------------------

U  Install              <-- U indicates the file Install updated
C  esrubps/bps.F        <-- C indicates conflicts exist in esrubps/bps.F
Updated to revision 3.  <-- Notification of update to revision number
```

Subversion will not allow you to commit any files until the conflict is manually resolved. A full discussion on resolving conflicts can be found in Chapter 3 of Versioning control with Subversion, by Collins-Sussman, Fitzpatrick and Pilato:

http://svnbook.red-bean.com/en/1.1/ch03s05.html#svn-ch-3-sect-5.4

10.8.3 Checking recent changes in development_branch log

If you have not been paying close attention to recent changes in the development_branch then you can use svn commands to find out what is the most recent revision to the development_branch and recover the development_branch log to see details of changes:

```
svn info https://espr.svn.cvsdude.com/esp-r/branches/development_branch
---------------------- (OUTPUT) ---------------------------------------
Path: development_branch
URL: https://espr.svn.cvsdude.com/esp-r/branches/development_branch
Repository Root: https://espr.svn.cvsdude.com/esp-r
Repository UUID: 7d53e970-de11-0410-8a54-3d01b9da36cf
Revision: 385
Node Kind: directory
Last Changed Author: ibeausol
Last Changed Rev: 355
Last Changed Date: 2006-07-28 08:03:06 -0400 (Fri, 28 Jul 2006)
```

Note the current revision number (385) and the development_branch was last revised at 355.

To find out what has changed in the development_branch ask subversion for the log:

```
svn log -v https://espr.svn.cvsdude.com/esp-r/branches/development_branch >> dev_log_355.txt
```

This creates a file dev_log_355.txt which you can open in a text editor. Giving the file a name which includes the revision number is a good way of keeping track of your work.

If you notice there were changes between revision 200 and 385 that have not been included in your branch then you will want to issue a svn merge command. Probably best to backup your sandbox before you do merges. Or you could checkout a fresh sandbox of your branch and do the merge into that. If horrible things happen you can toss away the fresh sandbox and start again.

```
cd <your_sandbox>
svn merge -r 200:385 https://espr.svn.cvsdude.com/esp-r/branches/development_branch
---------------------- (OUTPUT) ---------------------------------------
U integer.c
U button.c
U Makefile
```

6. Check to see if there are any conflicts and check the changes that have been merged.

10.8.4 Resolving conflicts

In the case that there are conflicts between the local changes and those on the development branch subversion will create a *left* and *right* version of the source file. To see the differences use a visual comparison tool with the

left and *right* versions. The source file itself will have embeded markings <<<<<<< or >>>>>>> indicating where the conflict is located. Manually edit **the source file** (not the left or right version files) and if you are happy with the result issue a "svn resolved" command with the source file name. If you want to take the development branch version execute an "svn revert".

REFERENCES

For more information on Subversion Merging, read Chapter 4: Branching and Merging of Versioning control with Subversion, by Collins-Sussman, Fitzpatrick and Pilato:

http://svnbook.red-bean.com/en/1.1/svn-book.html#svn-ch-4

Hand, J., 2011. The ESP-r Cookbook. University of Strathclyde, Glasgow, Scotland. http://www.esru.strath.ac.uk.

Hand, J., 2009. Documentation of Open-source Simulation - Addressing Multiple Points of Interest. IBPSA BS 09 Conference proceedings, Glasgow, Scotland 2009.

Svn repository is at https://espr.svn.cvsdude.com/esp-r/branches/development_branch