ESP-r Selftest

Jon Hand

4 February 2025

Summary

One aspect of the evolution of simulation tools is the need to periodically test for unintended changes in predictions. Code submissions rely on such tests. Ideally tests should be relatively simple to carry out and:

- test a mix of simulation facilities and assessment domains
- test a matrix of project models focused on different assessment goals
- test simulation models under different boundary conditions and control regimes
- test a range of reporting types and performance topics
- clarify differences between a reference and test version of the simulation tool

Developers have adopted many approaches to this challenge. ESP-r has, for decades, included a *tester* facility originally developed by Canmet Energy in Ottawa. This was implemented as a ~4500 line Perl script which took command line directives (there are two dozen) as seen in Figue 1. It acted on a matrix of 155 simulation models held within a `test_suite` set of folders and generated an overall pass-fail report with various options to drill down into the statistics of any differences found. See Appendix 1 for a summary of the `tester.pl` facility.

```
./tester.pl -v --no_h3k --diff_data --diff_tool meld --databases /opt/esp-r/ /opt/esp-r/bin/bps /home/jon/esp-r_joe/bin/bps --ref_loc /opt/esp-r/bin --test_loc
/home/jon/esp-r_joe/bin --save_results

___________  August 2023

./tester.pl -v --no_h3k --diff_data --diff_tool meld --databases /opt/esp-r/ /opt/esp-r/bin/bps /home/jon/esp-r_v17/bin/bps --ref_loc /opt/esp-r/bin --test_loc
/home/jon/esp-r_v17/bin --save_results

./tester.pl -v --no_h3k --diff_data --diff_tool meld --databases /opt/esp-r/ /opt/esp-r/bin/bps /home/jon/esp-r_v17/bin/bps --ref_loc /opt/esp-r/bin --test_loc
/home/jon/esp-r_v17/bin --save_results --path ../test_suite/esru_benchmark_model

___________  Feb 2024
```

Figure 1: tester.pl command line examples

```
jon@jon-OptiPlex-3000:~/Src/esrusvn4/standard/tester$ ls
additional_tests  Readme  README  scripts  testing_results  test_suite
jon@jon-OptiPlex-3000:~/Src/esrusvn4/standard/tester$ ls scripts
ANALYSE                        bps_test_report_25March2021.txt  co-sim.pl
ANALYSE_osx                    bps_test_report_4april_2022.txt  esp-r.cnf
automated_tests.pl             bps_test_report_6aug_2018.txt    nightly_branch_tests.dat
bps_test_report_17aug_2023.txt bps_test_report_6may_2021.txt    nightly_branch_tests.pl
bps_test_report_19july_2021.txt bps_test_report_6oct_2021.txt   Simple.pm
bps_test_report_23feb_2018.txt bps_test_report_feb_2022.txt     tester_example_use.txt
bps_test_report_24jan_2023.txt bps_test_report_feb_2024.txt     tester.pl
jon@jon-OptiPlex-3000:~/Src/esrusvn4/standard/tester$ ls test_suite
alberta_infil_model      cetc_battery_model      plt_elec_net            plt_trnsys_wrapper
Annex42_fuel_cell        check_suites.sh         plt_electric_HWT        plt_zone_heat_gain_coupling
ascii_dbs                complex_fenestration    plt_lookup_table        pv_example
basesimp                 dhw_bcd                 plt_NCHE                shading
bld_ground_reflectivity  elec_gain_into_zone     plt_pre_A42_PEMFC_model sloped_obstr
bld_hc_ISO15099          h3kreports              plt_pre_A42_SOFC_model  Ventilation
bld_PV                   idealized_hvac          plt_radiant_floor       which_cfg
ccht_benchmark           Lion_battery            plt_SDHW                window_control
cellular_miso            multi_year_simulations  plt_solar_collector
cellular_offices         plt_boundary_conditions plt_stratifed_tank
jon@jon-OptiPlex-3000:~/Src/esrusvn4/standard/tester$
```

Figure 2: tester scripts and test_suite folder

Unfortunately, the Perl script is ancient and there are no longer the skills set in the ESP-r community to maintain it or evolve it to match the evolution of ESP-r.

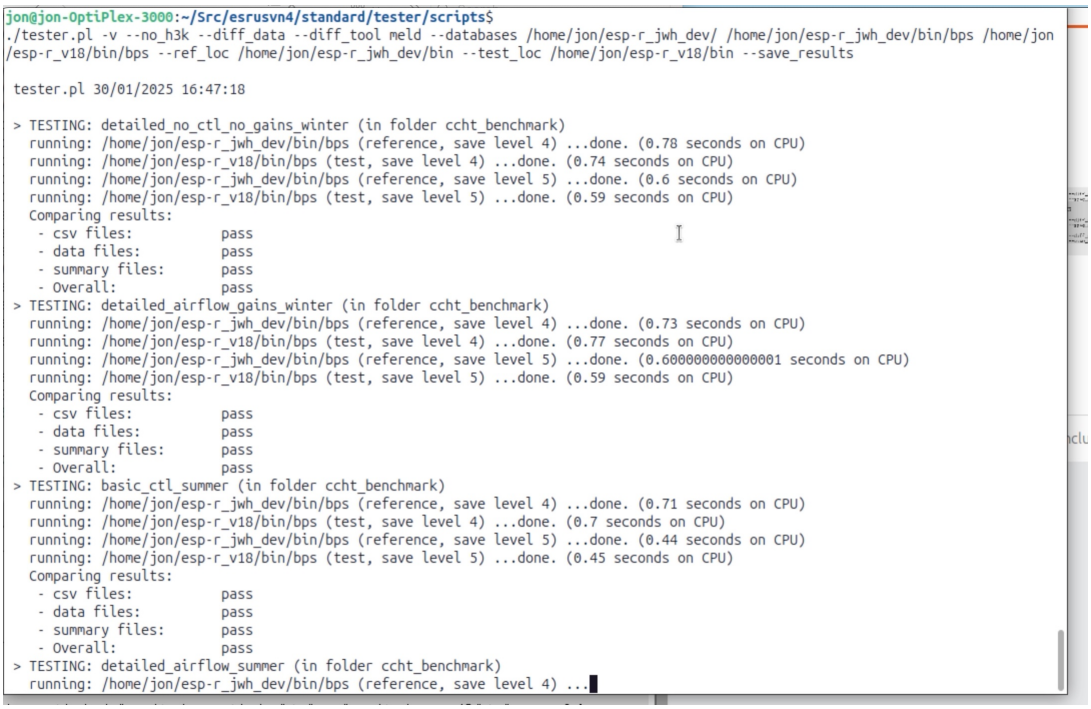


Figure 3: A tester.pl session traversing the matrix of tests

As an alternative to creating yet another complex edifice in a scripting language, facilities have been implemented within the ESP-r Project Manager (`prj`) source code. Logic supporting validation projects such as BESTEST already existed as well as facilities to encode data mining directives and invoke external software agents. These have been adapted and expanded to provide a similar testing regime to that provided by the Perl script.

The approach taken differs from tester.pl as follows:

- directives driving the tests are held in a text file rather than rely on command line options,
- the directives file is in tag-data format (see Appendix 3) and user editable so that new models and tests can be added as needed,
- data mining is carried out via directives in 'PIF' files which set the what/when/form of reporting (see Appendix 2),
- tester.pl required separate model cfg files for each assessment and edited the cfg files to impose different save levels. The selftest facility leverages simulation parameter sets (SPS) within model cfg files to support multiple timesteps, assessment periods and save levels (this reduces the test matrix from 155 to 118 model cfg files),
- the usr can interactively choose to test the full list, a specific group or individual models,
- rather than exist only in the source distribution, selftest directives and models are included in the ESP-r distribution in the models/validation/selftest folder

Approximately 1600 lines of Fortran implement a user interface (to select the tests), a parser for directives, invocation of the simulation engine, data mining, and reporting, and file management, as well as the capture of specific differences in performance. The directives file (see below and Appendix 3) includes:

- path to the reference and test versions of ESP-r,
- tolerance for determining whether performance differences should be reported.
- the list of models which may be assessed,
- the simulation parameter set (SPS) the simulation engine should use,
- the PIF file driving the data mining,
- names of the report files to be generated

```
*SELFTEST
*help  The models contained in the following sections demonstrate
*help  performance differences between two ESP-r versions.
*help
*help  It is possible to run individual tests or multiple selections.
*help   Please ensure relevant model and report folders exist!
*help   Edit entries to match ESP-r distribution locations.
*bin_path_std /home/jon/esp-r_std/bin/
*bin_path_tst /opt/esp-r/bin/
*model_path  /opt/esp-r/validation/selftest/
*report_path_std /opt/esp-r/validation/selftest/std_reports/
*report_path_tst /opt/esp-r/validation/selftest/tst_reports/
*tollerance 1.0
*verbose YES
*output FILE /opt/esp-r/validation/selftest/all_self.txt
*label  -- standard tester models --
*item
*group basic and detailed AIM
*help  A set of models demonstrating ESP-r's functionality.
*item
*name  basic_AIM_MAX
*cfg   basic_AIM_MAX.cfg
*root  alberta_infil/cfg/
*SPS    test_s4
*PIF    basic_AIM.pif
*res    results.bres
*rep    basic_AIM_MAX_test_s4.dat
*sum    basic_AIM_MAX_test_s4.summary
*item
*name  basic_AIM_MAX_S5
*cfg   basic_AIM_MAX.cfg
*root  alberta_infil/cfg/
*SPS    test_s5
*PIF    SL5
*res    results.bres
*sum    basic_AIM_MAX_test_s5.summary
*h3k    basic_AIM_MAX_test_s5.h3k
*item
*name  basic_AIM_MIN
*cfg   basic_AIM_MIN.cfg
. . .
```

Listing 1: The directives file format.

Users will typically replicate the 'models/validation/selftest' folder to a convenient location. The user would then edit the directives file (named `selftest_list` in the standard distribution) to point to the two ESP-r versions to be tested and adjust a few of the other attributes. The Project Manager is then invoked in `-mode text` so that the feedback as assessments, data mining and comparisons all happen in the same window. It is not critical which folder `prj` is invoked from but it might be convenient to do it within the replicated selftest folder. The user chooses 'self testing' from the menu and is then asked for the directives file and then selects the scope of the tests to be run.

The 'self testing' option scans the directives file name (supplied by the user) and presents a list of models that can be tested – these are in groups approximating those used by tester.pl.

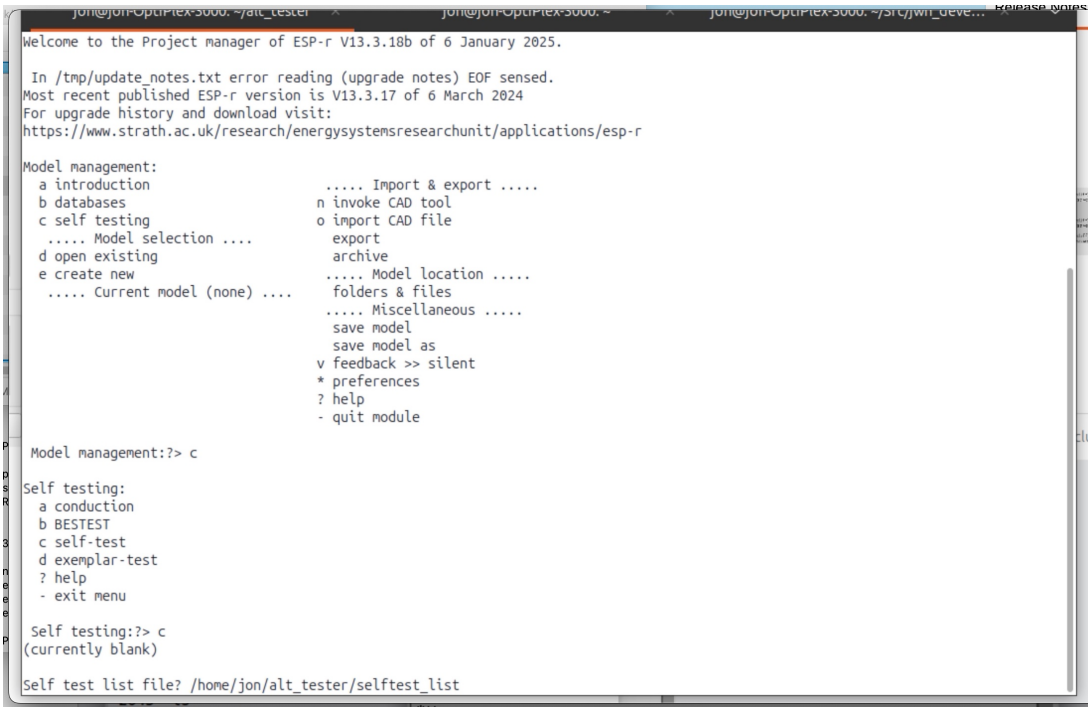


Figure 4: Invocation of prj, choose self testing and supply directives file name.

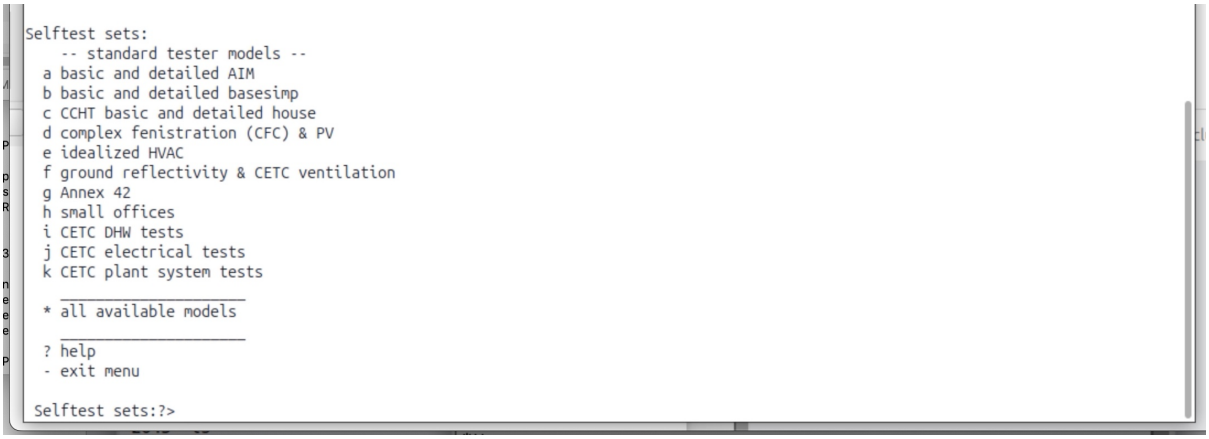


Figure 5: Named groups of tests

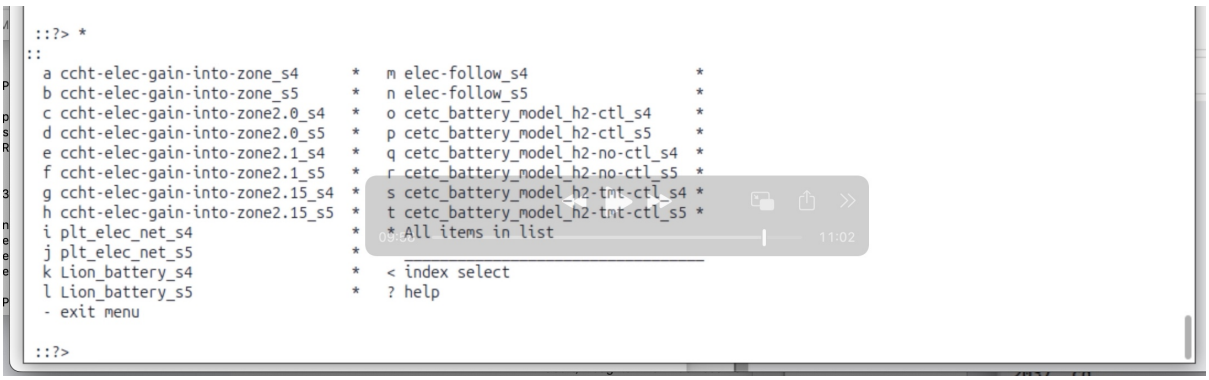


Figure 6: Models within a group (*) signals they have been selected

One can choose `*all available,` one of the named groups or specific tests within a group. In the Figure above the user selected all of a specific group to test. `Prj` then uses the directives for these specific models when invoking the assessments and doing the data mining. Figure 7 shows a typical session. In Figure 8 is a session running in 'verbose' model (useful to check that simulation facilities are being correctly invoked).

After running the assessment with the standard and test versions of the simulation engine the reports are scanned and if identical or within tolerance this is reported. In this case there are 5 lines different in the *.h3k file and 15 specific performance values outside tolerance in a .summary file.

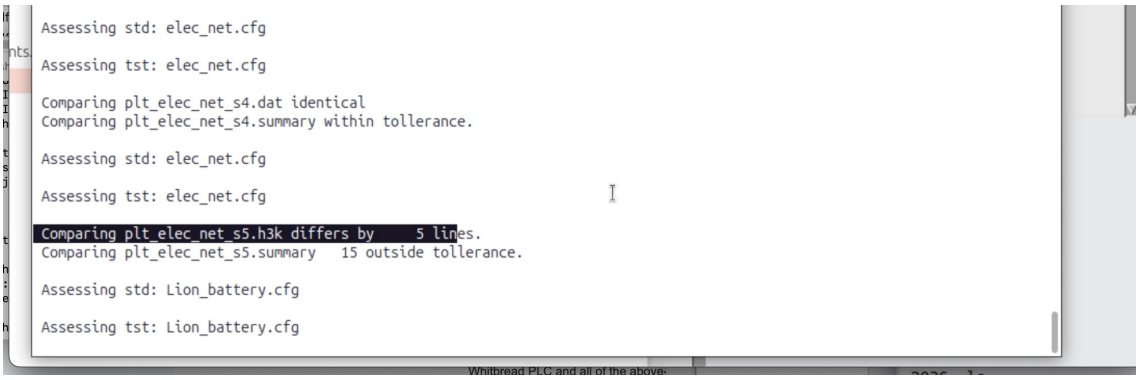


Figure 7: Feedback during the tests (in non-verbose mode)

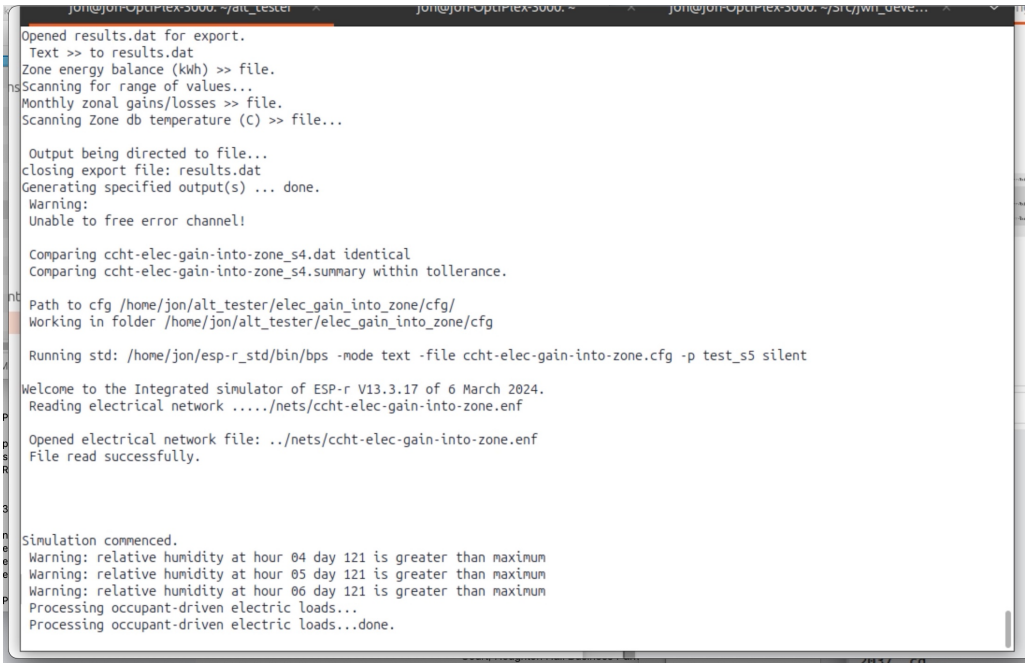


Figure 8: Selftest facility running in verbose mode.

At the end of the assessments and data mining the total number of ASCII line differences in `*.dat` files and `*.summary` lines outwith tolerance are reported.

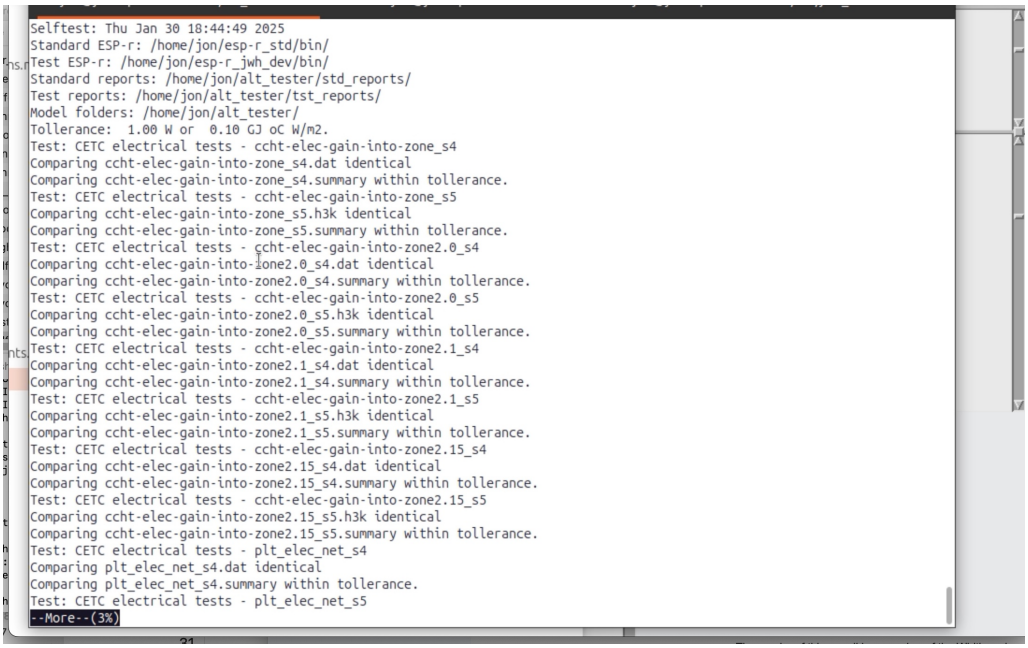


Figure 9: Selftest initial portion of summary report.

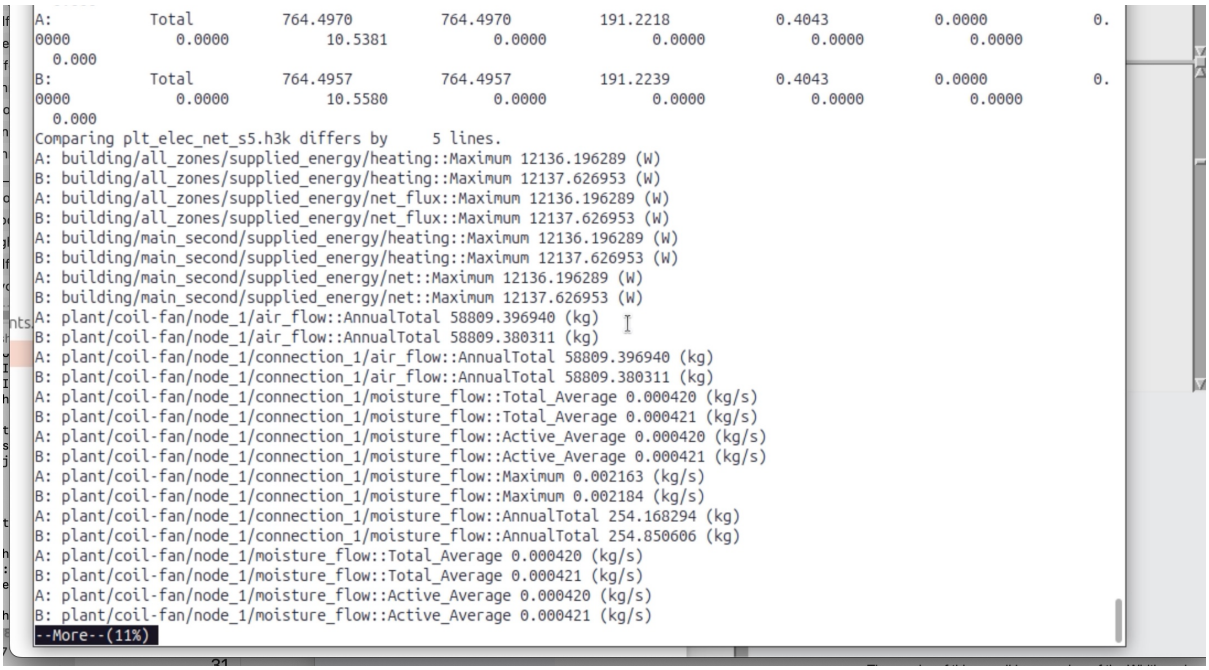


Figure 10: Specific differences found during tests.

The differences often are at the 3rd decimal place or a timing difference. As simulation is a numerical process such differences are often expected and accepted. The `*tollerance` directive can help with this.

At the end of the selftest process two folders are populated – one with reports generated by the standard simulation suite and the other with the test version. Users might use a visual difference tool such as `meld` on the two folders if a different kind of review is needed.

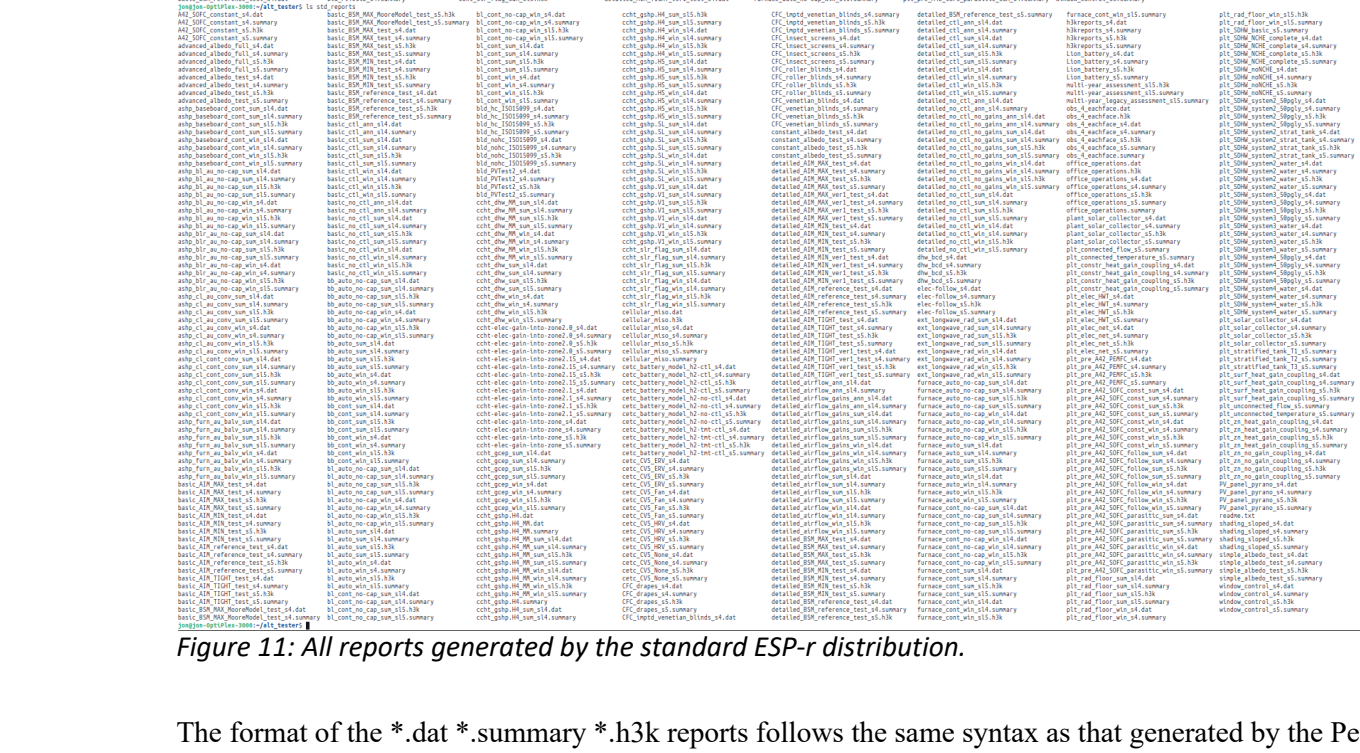


Figure 11: All reports generated by the standard ESP-r distribution.

The format of the *.dat *.summary *.h3k reports follows the same syntax as that generated by the Perl script. The primary difference is that data mining has not used a shell script to drive the extraction process.

All of the models found in the legacy tester test_suite have been converted to the current ESP-r file format, simulation parameter sets for the required assessment periods and save levels added and PIF files created to match these periods in order to drive subsequent data mining. The process allowed for some glitches in the original test_suite models to be corrected.

All models run – with the exception that if a legacy version of ESP-r is chosen it may not recognise the current ESP-r model file syntax. Also bugs that caused havoc in multi-year assessments have been corrected that it is now possible to run such models.

Adding further tests within the current matrix is straightforward once you figure out the patterns. For example to add an annual assessment to one of the models open it up in the Project Manager, add an additional simulation parameter set, ensure that the results files follow the results.bres results.pres etc. convention. Note the simulation parameter set name. Invoke the simulator and test that it will run that new assessment:
bps -mode text -file the_cfg_file -p the_sps_name silent

If that works, a PIF file is needed (to reflect the longer simulation period). There is a script named `create_selftest_pif.key` which will invoke `res` and traverse the various reports needed and generate a matching `test.pif` file. The first line of which needs to be edited to alter the file name to `results.dat` . What that done, replicate one of the directive `*item` sets for the model and edit it

to reflect the name of the simulation parameter set as well as the labels so the user selection makes sense. An example is shown below:

*item
```
*name  basic_ctl_summer
*cfg   basic_ctl.cfg
*root  ccht_benchmark/cfg/
*SPS    sum_sl4_ts1
*PIF    basic_sum_sl4_ts1.pif
*res    results.bres
*rep    basic_ctl_sum_sl4.dat
*sum    basic_ctl_sum_sl4.summary
*item
*name  basic_ctl_annual
*cfg   basic_ctl.cfg
*root  ccht_benchmark/cfg/
*SPS    ann_sl4_ts1
*PIF    basic_ann_sl4_ts1.pif
*res    results.bres
*rep    basic_ctl_ann_sl4.dat
*sum    basic_ctl_ann_sl4.summary
```

For more information on setting up new models for inclusion in the selftest facilities the following Appendices may help. Note, if an old version of ESP-r is to be tested then the syntax of the model files will need to be consistent with that older version.

Appendix 1: tester.pl

The Perl script tester.pl is in many respects over engineered to support large scale validation studies in which thousands of models are tested on a daily cycle. Within it's 164kB it includes scores of internal functions and although documented it was clearly written by someone who had extensive knowledge of large scale Perl deployments. Debugging of Perl is a niche skill.

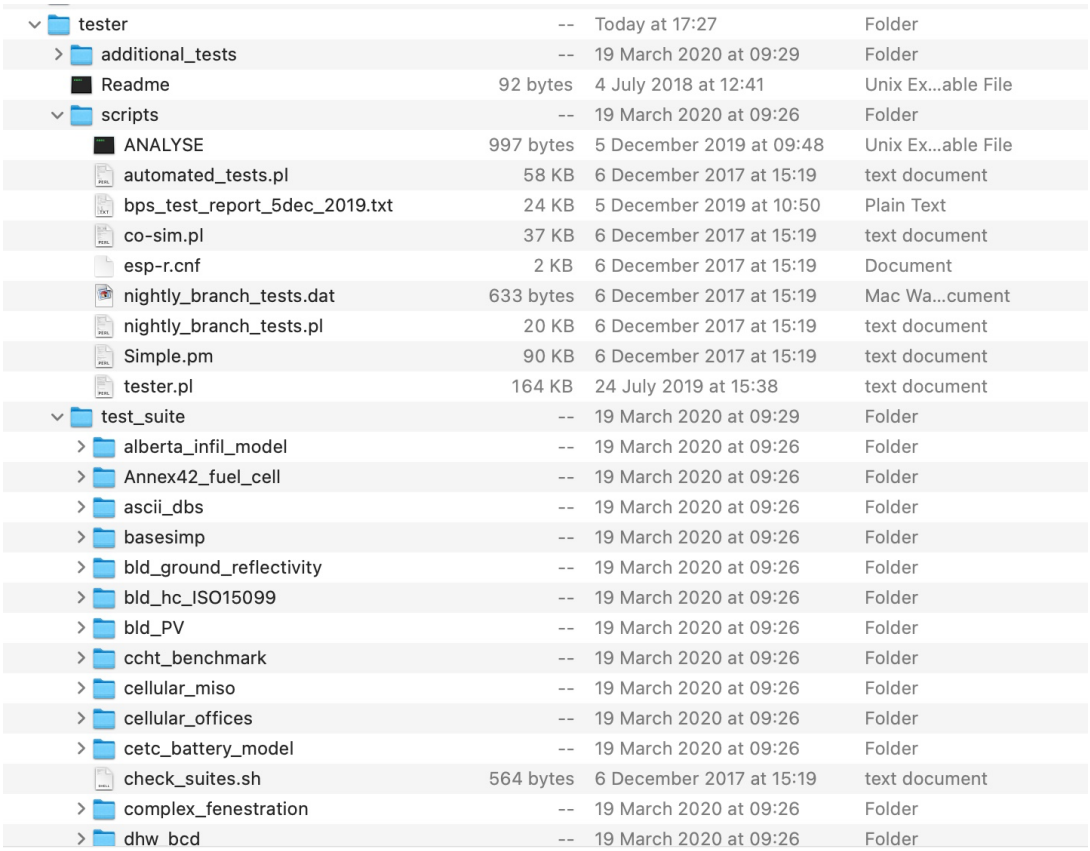
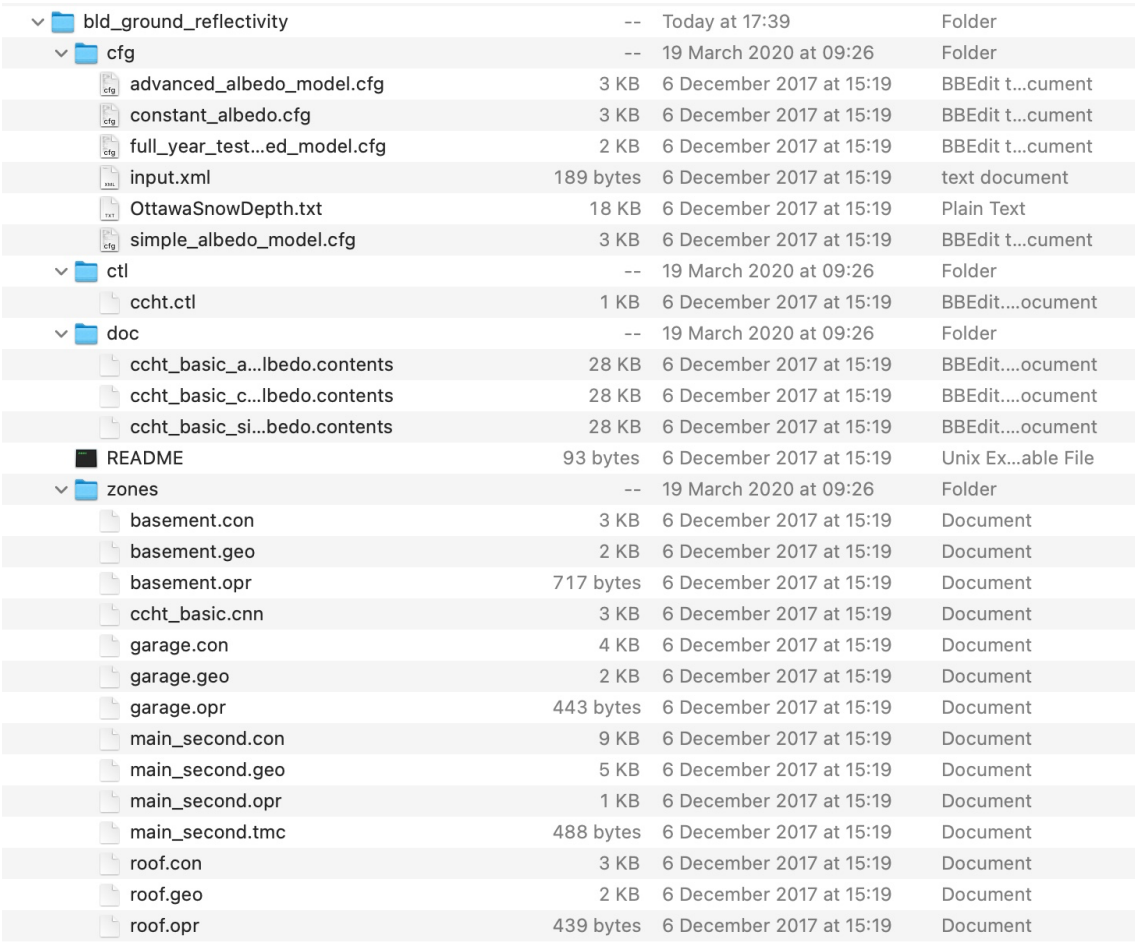


Figure 12: Layout of the tester folders in the source distribution of ESP-r.

The Perl script acts on a *test_suite* of ESP-r models which complied with a rule set e.g. what it expected to find within models and specific naming conventions. Although may of the models share attributes each has been adapted to focus assessments on a specific simulation feature e.g. an environmental control regime, layout of system or electrical components, or specific heat transfer path. For example to test facilities related to ground reflectance there are variant models looking at different computational approaches in Figure 7.

To ensure that computations were consistent across a range of boundary conditions some tests were carried out over different seasons. In Figure 8 there are winter and summer versions of each model, in other model folders there might be annual versions.



bld_ground_reflectivity	--	Today at 17:39	Folder
cfg	--	19 March 2020 at 09:26	Folder
advanced_albedo_model.cfg	3 KB	6 December 2017 at 15:19	BBEdit t...cument
constant_albedo.cfg	3 KB	6 December 2017 at 15:19	BBEdit t...cument
full_year_test...ed_model.cfg	2 KB	6 December 2017 at 15:19	BBEdit t...cument
input.xml	189 bytes	6 December 2017 at 15:19	text document
OttawaSnowDepth.txt	18 KB	6 December 2017 at 15:19	Plain Text
simple_albedo_model.cfg	3 KB	6 December 2017 at 15:19	BBEdit t...cument
ctl	--	19 March 2020 at 09:26	Folder
ccht.ctl	1 KB	6 December 2017 at 15:19	BBEdit....ocument
doc	--	19 March 2020 at 09:26	Folder
ccht_basic_a...lbedo.contents	28 KB	6 December 2017 at 15:19	BBEdit....ocument
ccht_basic_c...lbedo.contents	28 KB	6 December 2017 at 15:19	BBEdit....ocument
ccht_basic_si...bedo.contents	28 KB	6 December 2017 at 15:19	BBEdit....ocument
README	93 bytes	6 December 2017 at 15:19	Unix Ex...able File
zones	--	19 March 2020 at 09:26	Folder
basement.con	3 KB	6 December 2017 at 15:19	Document
basement.geo	2 KB	6 December 2017 at 15:19	Document
basement.opr	717 bytes	6 December 2017 at 15:19	Document
ccht_basic.cnn	3 KB	6 December 2017 at 15:19	Document
garage.con	4 KB	6 December 2017 at 15:19	Document
garage.geo	2 KB	6 December 2017 at 15:19	Document
garage.opr	443 bytes	6 December 2017 at 15:19	Document
main_second.con	9 KB	6 December 2017 at 15:19	Document
main_second.geo	5 KB	6 December 2017 at 15:19	Document
main_second.opr	1 KB	6 December 2017 at 15:19	Document
main_second.tmc	488 bytes	6 December 2017 at 15:19	Document
roof.con	3 KB	6 December 2017 at 15:19	Document
roof.geo	2 KB	6 December 2017 at 15:19	Document
roof.opr	439 bytes	6 December 2017 at 15:19	Document

Figure 13: Models focused on different ground reflectance approaches.

The script collected a master list of all of the ESP-r cfg files and then replicated each model, made a temporary copy of the model cfg file and edited it's attributes to support multiple assessments at different 'save levels'.

```
jon@jon-OptiPlex-3000:~/Src/esrusvn4/standard/tester/test_suite/idealized_hvac/cfg$ ls *.cfg
ashp_baseboard_cont.cfg              baseboard_cont_winter.cfg           ccht_gshp.HS_winter.cfg
ashp_baseboard_cont_summer.cfg       boiler_auto_no-cap_summer.cfg       ccht_gshp.SL_summer.cfg
ashp_baseboard_cont_winter.cfg       boiler_auto_no-cap_winter.cfg       ccht_gshp.SL_winter.cfg
ashp_boiler_auto_no-cap_summer.cfg   boiler_auto_summer.cfg              ccht_gshp.V1_summer.cfg
ashp_boiler_auto_no-cap_winter.cfg   boiler_auto_winter.cfg              ccht_gshp.V1_winter.cfg
ashp_cool_auto_conv_summer.cfg       boiler_cont_no-cap_summer.cfg       ccht_slr_flag_summer.cfg
ashp_cool_auto_conv_winter.cfg       boiler_cont_no-cap_winter.cfg       ccht_slr_flag_winter.cfg
ashp_cool_cont_conv_summer.cfg       boiler_cont_summer.cfg              ext_longwave_rad_summer.cfg
ashp_cool_cont_conv_winter.cfg       boiler_cont_winter.cfg              ext_longwave_rad_winter.cfg
ashp_furnace_auto_balv_summer.cfg    ccht_dhw_summer.cfg                 furnace_auto_no-cap_summer.cfg
ashp_furnace_auto_balv_winter.cfg    ccht_dhw_summer_MooreModel.cfg      furnace_auto_no-cap_winter.cfg
baseboard_auto_no-cap_summer.cfg     ccht_dhw_winter.cfg                 furnace_auto_summer.cfg
baseboard_auto_no-cap_winter.cfg     ccht_gcep_summer.cfg                furnace_auto_winter.cfg
baseboard_auto_summer.cfg            ccht_gcep_winter.cfg                furnace_cont_no-cap_summer.cfg
baseboard_auto_winter.cfg            ccht_gshp.H4_summer.cfg             furnace_cont_no-cap_winter.cfg
baseboard_cont_no-cap_summer.cfg     ccht_gshp.H4_summer_MooreModel.cfg  furnace_cont_summer.cfg
baseboard_cont_no-cap_winter.cfg     ccht_gshp.H4_winter.cfg             furnace_cont_winter.cfg
baseboard_cont_summer.cfg            ccht_gshp.HS_summer.cfg
jon@jon-OptiPlex-3000:~/Src/esrusvn4/standard/tester/test_suite/idealized_hvac/cfg$
```

Figure 14: Perl script requirement of multiple model cfg files.

As each assessment is run a suite of reports were generated. One (listing 2 below) is derived from invoking the ESP-r `res` module via a data mining script. Other reports (listings 3 & 4) were generate by the simulation engine based on the directives in an input.xml file: a so-called .summary file and a .h3k file which corresponded with the needs of a Canadian simulation-based incentive scheme.

```
# Results library: results.bres; (Results cellular_miso)
# Output period: 00:02 on 06/06/67 to 23:57 on 14/06/67  (STS=05m, OTS=05m)
# Sensible heating load (kW)
Description Max_value Max_occur Min_value Min_occur Ave_value Std_dev
manager_a               0.000  06-Jun@00h02       0.000  06-Jun@00h02       0.000     0.000
manager_b               0.000  06-Jun@00h02       0.000  06-Jun@00h02       0.000     0.000
coridor                 0.000  06-Jun@00h02       0.000  06-Jun@00h02       0.000     0.000
All                     0.000  06-Jun@00h00       0.000  06-Jun@00h00       --          --

# Sensible cooling load (kW)
Description Max_value Max_occur Min_value Min_occur Ave_value Std_dev
manager_a               0.000  06-Jun@00h02      -0.300  06-Jun@09h42      -0.108     0.133
manager_b               0.000  06-Jun@00h02      -0.300  06-Jun@09h47      -0.106     0.132
coridor                 0.000  06-Jun@01h27      -0.300  06-Jun@11h02      -0.128     0.126
All                     0.000  06-Jun@00h00      -0.900  06-Jun@11h02       --          --

# Zone db temperature (C)
Description Max_value Max_occur Min_value Min_occur Ave_value Std_dev
manager_a                30.8  12-Jun@14h17        20.1  09-Jun@04h42       25.02     2.417
manager_b                30.8  12-Jun@14h17        20.6  09-Jun@04h37       25.06     2.326
coridor                  30.0  11-Jun@10h52        23.0  08-Jun@05h02       25.24     1.580
All                      30.8     --               20.1     --              25.11     --
. . .


 Zone energy requirements summary
    Zone                Sensible heating           Sensible cooling         Humidification    Dehumidification
 id name                Energy       Hours         Energy       Hours       Energy   Hours    Energy  Hours
                     kWhrs  kWhrs/m2 required    kWhrs  kWhrs/m2 required    kWhrs  required   kWhrs  required
  1 manager_a         0.00     0.00     0.0     -23.42    -1.74   104.1       0.00     0.0       0.00     0.0
  2 manager_b         0.00     0.00     0.0     -22.84    -1.69   104.0       0.00     0.0       0.00     0.0
  3 coridor           0.00     0.00     0.0     -27.69    -3.29   149.6       0.00     0.0       0.00     0.0

    All               0.0      0.0      0.      -74.0     -2.1    358.        0.0      0.0       0.0      0.0

     0.0 hours when heating required in at least one zone.
   149.5 hours when cooling required in at least one zone.

 Causal energy breakdown (kWh) at air point for zone 1: manager_a

                                    Gain         Loss
 Infiltration air load             0.000     -13.157
 Ventilation air load              0.114      -0.010
 Casual Occupt                     5.670       0.000
 Casual Lights                     0.000       0.000
 Casual Equipt                     0.000       0.000
 Casual --                         0.000       0.000
 Controlled casual gain            0.000       0.000
 Thermal bridge (linear)           0.000       0.000
 Heat storage @ air point          0.951      -0.953
 Convection @ opaque surf: ext     2.926      -0.382
 Convection @ opaque surf: ptn    26.713      -1.185
 Convection @ transp surf: ext     2.797      -5.822
 Convection @ transp surf: ptn     2.006      -0.055
 Convection portion of plant       0.000     -23.423
 Totals                           41.177     -44.987
 . . .
```

Listing 2: fragments of a *.dat report via scripted invocation of ESP-r res module.

```
building/all_zones/energy_balance/net::Total_Average 602.755216 (W)
building/all_zones/energy_balance/net::Active_Average 602.755216 (W)
building/all_zones/energy_balance/net::Maximum 2756.129639 (W)
building/all_zones/energy_balance/net::Minimum -0.000244 (W)
building/all_zones/energy_balance/net::AnnualTotal 0.468702 (GJ)
building/all_zones/envelope/all_components/heat_loss::Total_Average 762.446896 (W)
building/all_zones/envelope/all_components/heat_loss::Active_Average 762.446896 (W)
building/all_zones/envelope/all_components/heat_loss::Maximum 4243.211914 (W)
building/all_zones/envelope/all_components/heat_loss::Minimum 0.000000 (W)
building/all_zones/envelope/all_components/heat_loss::AnnualTotal 0.592879 (GJ)
building/all_zones/envelope/all_components/net_flux::Total_Average 0.000000 (W)
building/all_zones/envelope/all_components/net_flux::Active_Average 0.000000 (W)
building/all_zones/envelope/all_components/net_flux::Maximum 0.000000 (W)
building/all_zones/envelope/all_components/net_flux::Minimum 0.000000 (W)
building/all_zones/envelope/all_components/net_flux::AnnualTotal 0.000000 (GJ)
building/all_zones/envelope/ceilings/heat_gain::Total_Average 0.080329 (W)
building/all_zones/envelope/ceilings/heat_gain::Active_Average 0.080329 (W)
building/all_zones/envelope/ceilings/heat_gain::Maximum 2.844705 (W)
building/all_zones/envelope/ceilings/heat_gain::Minimum 0.000000 (W)
```

```
building/all_zones/envelope/ceilings/heat_gain::AnnualTotal 0.000062 (GJ)
building/all_zones/envelope/ceilings/heat_loss::Total_Average 35.615911 (W)
building/all_zones/envelope/ceilings/heat_loss::Active_Average 35.615911 (W)
building/all_zones/envelope/ceilings/heat_loss::Maximum 96.418106 (W)
building/all_zones/envelope/ceilings/heat_loss::Minimum 0.000000 (W)
building/all_zones/envelope/ceilings/heat_loss::AnnualTotal 0.027695 (GJ)
building/all_zones/envelope/ceilings/net_flux::Total_Average 35.535582 (W)
building/all_zones/envelope/ceilings/net_flux::Active_Average 35.535582 (W)
```

Listing 3: fragment of a *.summary file via input.xml directives

```
Performance assessment report
Results library ./results.bres
Climate file clm67
Configuration file cellular_miso.cfg
Configuration descr MISO control model see log file for more details
Period Tue-06-Jun to Wed-14-Jun Year 1967

 Zone         max air T  (occurance)                          min air T  (occurance)
manager_a      30.82 Mon-12-Jun@14.29   20.14 Fri-09-Jun@ 4.71
manager_b      30.81 Mon-12-Jun@14.29   20.62 Fri-09-Jun@ 4.62
coridor        30.00 Sun-11-Jun@10.88   23.02 Thu-08-Jun@ 5.04

 Zone           max heat (occurance)      max cool (occurance)        heating     cooling
                  (kW)                      (kW)                      (MJ)        (MJ)
manager_a         0.00 Tue-06-Jun@ 0.04     -0.30 Tue-06-Jun@ 9.71        0.0      -96.2
manager_b         0.00 Tue-06-Jun@ 0.04     -0.30 Tue-06-Jun@ 9.79        0.0      -92.2
coridor          -0.01 Tue-06-Jun@ 0.04     -0.30 Tue-06-Jun@11.04       -0.0     -111.9

All zones:
 Max_Temp   30.8  in manager_a on Mon-12-Jun@14.29
 Min_Temp   20.1  in manager_a on Fri-09-Jun@ 4.71
 Max_Heat    0.0  in manager_a on Tue-06-Jun@ 0.04
 Max_Cool   -0.3  in manager_a on Tue-06-Jun@ 9.71

Total heating requirements       -0.0 (MJ)
Total cooling requirements    -300.24 (MJ)

Monthly metrics:
Month   Heating   Cooling
Month    (MJ)      (MJ)
Jun       -0.0     -300.2

********SYSTEMS INFORMATION*********

FAN, HRV, AND PUMP ELECTRIC ENERGY
          MONTH        FAN_ENERGY MJ     HRV ENERGY MJ      GSHP_PUMP MJ      GCEP_PUMP MJ
              JUN               0.0000            0.0000            0.0000            0.0000
   TOTAL ELEC ENERGY            0.0000            0.0000            0.0000            0.0000

**********ZONE INFORMATION*********

          Zone( 1) manager_a
          Month     Aver.Temp (oC)   Solar Extern(MJ)   Solar Intern(MJ)  Sol Abs Trans(MJ)   Sol Abs
Opq.(MJ)   Casual Rad. (MJ)  Casual Conv. (MJ)   Fndtn Losses(MJ)
            Jun           30.9668           267.6857            2.4810           19.2505
216.9950           20.4120           20.4120            0.0000

          Zone( 2) manager_b
          Month     Aver.Temp (oC)   Solar Extern(MJ)   Solar Intern(MJ)  Sol Abs Trans(MJ)   Sol Abs
Opq.(MJ)   Casual Rad. (MJ)  Casual Conv. (MJ)   Fndtn Losses(MJ)
            Jun           30.9360           267.6855            2.4810           19.2505
216.9949           20.4120           20.4120            0.0000

          Zone( 3) coridor
          Month     Aver.Temp (oC)   Solar Extern(MJ)   Solar Intern(MJ)  Sol Abs Trans(MJ)   Sol Abs
Opq.(MJ)   Casual Rad. (MJ)  Casual Conv. (MJ)   Fndtn Losses(MJ)
            Jun           31.2372            0.0000           38.2746            2.8161
30.4963           43.7667           43.7667            0.0000

SDHW Data
          Month   Tank Elec (kWh)   Tank Fuel (kWh)  Solar Gain (kWh)   Pump Elec (kWh)
            JUN            0.0000            0.0000            0.0000            0.0000
          Total            0.0000            0.0000            0.0000            0.0000
```

Listing 4: fragments of a *.h3k report file.

At the end of a tester.pl session a summary would be generated to show which tests were competed and which failed because of differences in the predictions.

```
tester.pl Test Report
Testing commenced on 27/02/2024 10:41:02


Test parameters:
 - Test suite path:        /home/jon/Src/esrusvn/jwh_alt_commons/tester/test_suite/
 - Abbreviated runs:       disabled

Test System Information:
 - Username:               jon
 - Host:                   jon-OptiPlex-3000
 - Platform:               x86_64
 - Operating system:       Linux:6.2.0-39-generic

bps binaries:
 - Path:                   (reference)  jon-OptiPlex-3000:/opt/esp-r/bin/bps
                           (test)       jon-OptiPlex-3000:/home/jon/esp-r_v17/bin/bps
 - SVN source:             (reference)
                           (test)
 - Compilers:              (reference)
                           (test)
 - Graphics library:       (reference)
                           (test)
 - XML support:            (reference)
                           (test)
 - Modifiation date:       (reference)  2024-02-27 10:39:55.178557416 +0000
                           (test)       2024-02-27 10:31:52.829774026 +0000
 - MD5 Checksum:           (reference)  6c0b6d03e36771e14cd5721e6a36007b
                           (test)       973598df7b7edf3c5aba6ab14576ffb5
                                        (files differ)

Compared output: .csv .data .summary 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                                        .summary  .xml .data  .csv  overall dt-CPU(%)
-------------------------------------------------------------------------------------------------------------------------------
Annex42_fuel_cell              SOFC_constant                                   .      .    -      .      -
Lion_battery                   Lion_battery                                    .      .    -      .      -
NCHE_basic                     NCHE_basic                                      .      .    .      .      X
NCHE_complete                  NCHE_complete                                   .      .    -      .      -
NCHE_complete_noNCHE           NCHE_complete                                   .      .    -      .      -
Ventilation                    CVS_ERV                                         .      .    -      .      -
Ventilation                    CVS_Fan                                         .      .    -      .      -
Ventilation                    CVS_HRV                                         .      .    -      .      -
Ventilation                    CVS_None                                        .      .    -      .      -
alberta_infil_model            basic_AIM_MAX                                   .      .    -      .      -
```

Figure 15: Summary report generated by tester.pl

This combination of reports allowed users a variety of views into building and system performance. It also ensured that any logic changes that impacted the production of performance reports were also evident. In general these report types should be produced in any replacement software testing regime.

The legacy state of the Perl script and depths of it's interaction with the simulation models is typified by it's method of adapting the model files so that specific assessments could be run. The Perl scrip assumed a model cfg file format which existed in ~2005 on which to modify the file attributes. For example it expected that simulation parameters were held in the following syntax:

```
*sps    1   1   4   1   5   0   # parameter sets, default startup, zone & plant ts, save level & frequency
   1   2   7   2  test          # startup, zone & plant ts, save level, @ ts, period start DM & end DM
*sblr results.bres
*splr results.pres
*selr results.eres
*end_set
*end_sps
```

More current versions of ESP-r assume the following syntax:

```
*sps    2  1  4  1  5  0         # parameter sets, default startup, zone & plant ts, save level & frequency
*set    1  4  1  4  0  1  2  7  2 test_s4  # startup, zone & plant ts, save level, @ ts, period start & end
*sblr ./results.bres
*splr ./results.pres
*selr ./results.eres
*set    1  4  1  5  0  1  2  7  2 test_s5
*sblr ./results.bres
*splr ./results.pres
*selr ./results.eres
```

The newer syntax allows each simulation parameter set to have a different save level and building and plant timesteps (obviating the need for editing of the cfg files). Newer models would have to be edited to conform to that 2005 syntax and the current file I/O code had to recognise and respect the archaic syntax.

As the models in the tester test_suite used a legacy format, the conversion to selftest required the following steps:
a) replicate the test_suite model folders (so as not to mess up the original models),
b) taking each test_suite model folders in turn, investigate the simulation periods implied,
c) if there were winter, summer and annual versions copy one cfg file into a new name e.g. basic_control_summer.cfg, basic_control_winter.cfg becomes basic_control.cfg with the following pattern:

```
*sps    5   3   1  10   5   0   # parameter sets, default startup, zone & plant ts, save level & frequency
*set    3   1  10   5   0   1   7   7   7  sum_sl5_ts1  # startup, zone & plant ts, SL, @ ts, period
*sblr ./results.bres
*set    3   1  10   5   0  15   1  21   1  win_sl5_ts1
*sblr ./results.bres
*set    3   1  10   4   0   1   7   7   7  sum_sl4_ts1
*sblr ./results.bres
*set    3   1  10   4   0  15   1  21   1  win_sl4_ts1
*sblr ./results.bres
*set    3   1  10   4   0   1   1  31  12  ann_sl4_ts1
*sblr ./results.bres
```

The names for parameter sets are then referenced in the selftest directives file. Also note that the names of the performance files created take the form `./results.bres` (the ./ ensures that the simulation engine writes the file in the same folder as the `*.cfg` file.

To save time later, invoke the simulator manually to see if the assessment can be run without faulting e.g. `bps -mode text -file basic_control.cfg -p win_sl5_ts1 silent`

If there were multiple *.cfg files in a folder determine if they were substantially the same model. If so the selftest facility needs only a PIF file for the winter, summer and/or annual assessments (see Appendix 2).

Appendix 2: the PIF file

Validation studies, such as BESTEST often stipulate specific performance attributes to be tracked and compared. One effort to limit user 'finger slips' or tool interface changes from breaking automation scripts and introducing errors into validation tasks and aid routine validation runs is to embed directives into the test models.

In ESP-r, this took the form of a so-called performance information file (PIF). Here is a brief summary of how it works. The ESP-r results analysis application interface reacts to user requests for reports on specific topics by instantiating a number of attributes held in memory about the what/where/when/form of reports to be generated. These attributes are then used by the data mining facilities when producing reports.

Of course, an automation script could be the source of 'user requests' but these are somewhat fragile so long ago the PIF concept, data structures and logic were evolved to support the specific data recovery tasks needed for validation. During an initial interactive data mining session the what/where/when/form attributes are written to a file. When invoked with a '-recovery' directive the data mining functions take their directives from the file rather than a users or automation script keystrokes.

The original PIF concept was focused on a specific sub-set of performance data. During 2023 and 2024 the code was extended to almost all reporting types and topics with the aim that any data recovery task that could normally be carried out by a user or an automation script could be handled by the '-recovery' facility. The selftest facility leverages this capability.

Below are fragments of a PIF file. They are somewhat human readable.

```
*Performance information file
1 results.dat # Output file
# Group name and description.
name_here description_here
  5 # statistics
# Start  day month  hour finish day month hour output-ts averaging
*period    15     6     1    22     6    24     1     0
# directives: occupancy rate-of-change anchors delimeter ctl-patterns units hours day-demarcation labels
*filters     0     0     0 - UNKNOWN     0     0     0     0
*metlist
# metric zone surf/type node/layer res-set line-type symbol axis label-width
*data      8     1     0     1     1    -3     6     2    12
*slab Sensible heating load (kW)
*glab Default
*end
# ---------------------------------
# Group name and description.
name_here description_here
  5 # statistics
# Start  day month  hour finish day month hour output-ts averaging
*period    15     6     1    22     6    24     1     0
# directives: occupancy rate-of-change anchors delimeter ctl-patterns units hours day-demarcation labels
*filters     0     0     0 - UNKNOWN     0     0     0     0
*metlist
# metric zone surf/type node/layer res-set line-type symbol axis label-width
*data      9     1     0     1     1    -3     6     2    12
*slab Sensible cooling load (kW)
*glab Default
*end
# ---------------------------------
 . . .
# Group name and description.
name_here description_here
  9 # zone energy balance
# Start  day month  hour finish day month hour output-ts averaging
*period    15     6     1    22     6    24     1     0
# directives: occupancy rate-of-change anchors delimeter ctl-patterns units hours day-demarcation labels
*filters     0     0     0 - UNKNOWN     0     0     0     0
*metlist
# metric zone surf/type node/layer res-set line-type symbol axis
*data     61     1     0     0     1    -3     6     4
*slab Zone energy balance (kWh)
*glab Default
*end
```

```
# ---------------------------------
# Group name and description.
name_here description_here
 13 # monthly reporting
# Start  day month  hour finish day month hour output-ts averaging
*period    15     6     1    22     6    24     1     0
# directives: occupancy rate-of-change anchors delimeter ctl-patterns units hours day-demarcation labels
*filters     0     0     0 - UNKNOWN     0     0     0     0
*metlist
# metric zone surf/type node/layer res-set line-type symbol axis monthly-column-choices
*data     64     1     0     1     1    -3     6     4     1     1     1     1     1     1
*slab Monthly zonal gains/losses
*glab Default
*end
# ---------------------------------
# Group name and description.
name_here description_here
  2 # histogram
# Start  day month  hour finish day month hour output-ts averaging
*period    15     6     1    22     6    24     1     0
# directives: occupancy rate-of-change anchors delimeter ctl-patterns units hours day-demarcation labels
*filters     0     0     0 - UNKNOWN     0     0     0     0
*metlist
# metric zone surf/type node/layer res-set line-type symbol axis
*data      1     1     0     0     1     1     6     1
*slab Zone db temperature (C)
*glab Default
*end
```

For example, the `*period` attributes set the start and end of the data recovery for the current topic. The `*filters` attributes define if data is filtered by occupied-unoccupied periods, the kind of delimeter to use in reports, etc. The `*metlist` attributes define which of the performance topic to recover, which portion of the simulation model to focus on and what symbols apply. There is also a label for this portion of the report.

Where separate winter, summer or annual assessments are required, separate PIF definitions are needed so that the `*period` attributes are correct. Where selftest is testing a performance issue with alternative approaches which imply different underlying zone and surface definitions there might need to be separate PIF files so that the `*data` attributes match.

For the selftest facility where the same report format was desired across the matrix of models it made sense to ease the creation of PIF files.

Assuming the model cfg files were updated (as mentioned in the prior Appendix1) a PIF file is generated by running a simulation and then invoking an option in the res module to dump the PIF attributes to a file.

Given that the same order and form of reporting was desired for the more than 100 models it made sense to capture the keystokes needed to generate a PIF file while using res. This sequence (distributed as `create_selftest.key`) creates a test.pif file.

a) choose a model cfg file and run one of simulation parameter sets for example:
`bps -mode text -file basic_control.cfg -p win_sl5_ts1 silent`

This will create a `results.bres` file which the `create_selftest.key` script will look for. Run it via `./create_selftest.key` which will create a `test.pif` file. The first line of the file needs to be edited to change the ??.dat to `results.dat`. If there were winter, summer and annual simulation parameter sets then for each, re-run bps and then the .key script and edit it's first line and change the pif file name i.e. winter.pif. These pif file names will be referenced in the selftest directives file.

Appendix 3: The selftest directives file

To support the selftest facility directives about which models, what periods, what data mining to undertake and what to name the performance report files generated as the matrix of tests progress are held in a text file. The file is user generated. It follows a tag data format and fragments of the file are included below to illustrate the tags and attributes.

The pattern of tests in the selftest models reflects those in the original tester test_suite:

a) For building only or building + plant models both save level 4 and save level 5 runs are made. Save level 4 data mining is based on the directives in PIF files and generate `*.dat` files. Save level 5 data mining is based on directives within an `input.xml` file found in each model cfg folder and generate `*.summary` and usually `*.h3k` report files.

b) For plant only models there is no save level 4 data mining and so directives in the selftest directives file specify only save level 5 runs. Also, the `*.h3k` reports cannot be generated for plant only models so those tokens are not included in the directives file.

c) The assessment periods e.g. winter, summer, annual or other periods are replicated.

Thus for winter and summer assessments at both save levels one might expect to find a sequence of four `*items` entries for the same model.

Although the key words are fixed the names and labels are up to the user. For the currently defined selftest models a number of naming conventions have been used although most of these are not set in stone with the exception that the simulator is expected to create results.bres / results.pres / results.eres depending on the domains included in the assessment.

What is important is that references within the directives file are consistent with the attributes within the ESP-r simulation models. For example, simulation parameter set names need to have an exact match or the simulation will fail. When introducing a new model or assessment, it generally saves time to add items to the directives file one at a time and then invoke the selftest facility to see if it runs complete correctly and the relevant report files have been generated.

The user edits needed to be applied to the distributed version of the directives file would be to adjust: `*bin_path_std,*bin_path_tst *model_path` to reflect locations on the current computer. Most users will want to replicate the whole of the selftest folder in the ESP-r distribution to a convenient location e.g.

```
rsync -av /home/ralph/src/esp-r_v13/models/validation/selftest/ /home/ralph/selftest/
```

Other common edits would identify where the reports generated should be placed. After review these might be archived or removed as required.

```
*report_path_std, *report_path_tst
```

Checks on *.dat files are for literal differences. Checks for .summary files can have a tolerance filter greater than 1.0 if lots of tiny differences are obscuring reports. And if the sefltest process is faulting it is useful to alter the *verbose from NO to YES.

```
*tollerance 1.0
*verbose YES
```

In the listing below comments are noted in red.

```
*SELFTEST      header tag
*help  The models contained in the following sections demonstrate
*help  performance differences between two ESP-r versions.
*help          highlevel documentation
```

```
*help  It is possible to run individual tests or multiple selections.
*help   Please ensure relevant model and report folders exist!
*help   Edit entries to match ESP-r distribution locations.
*bin_path_std /home/jon/esp-r_std/bin/   path to a standard set of executables
*bin_path_tst /opt/esp-r/bin/            path to the test set of executables
*model_path  /opt/esp-r/validation/selftest/    where the test models are located
where the *.dat, *.summary *.h3k files for the standard version are placed
*report_path_std /opt/esp-r/validation/selftest/std_reports/
where report files for the test version are placed
*report_path_tst /opt/esp-r/validation/selftest/tst_reports/
*tollerance 1.0      threshold for reporting differences e.g. 1 = 1W, 0.1GJ, 0.1 deg C
*verbose YES         see underlying chatter as agents are invoked and run
*output FILE /opt/esp-r/validation/selftest/all_self.txt     summary report
*label  -- standard tester models --     used by interface
*item         start of a group or a test
*group basic and detailed AIM            short description for interface
*help  A set of models demonstrating ESP-r's functionality.
*item                       marks start of a test (in this case save level 4)
*name  basic_AIM_MAX        identifier for logic as well as user
*cfg   basic_AIM_MAX.cfg    model cfg file
*root  alberta_infil/cfg/   local path (to append to *model_path above
*SPS    test_s4             simulation parameter set name
*PIF    basic_AIM.pif       data mining directives (for save level 4 runs)
*res    results.bres        zone results file for data mining to act on
*rep    basic_AIM_MAX_test_s4.dat        report generate by res
*sum    basic_AIM_MAX_test_s4.summary    simulator report based on input.xml directives
*item                       marks start of a test (in this case save level 5)
*name  basic_AIM_MAX_S5
*cfg   basic_AIM_MAX.cfg
*root  alberta_infil/cfg/
*SPS    test_s5
*PIF    SL5                 tag to signal only simulator reports expected
*res    results.bres        dummy file created by res (ignored)
*sum    basic_AIM_MAX_test_s5.summary    simulator report based on input.xml directives
*h3k    basic_AIM_MAX_test_s5.h3k        a report used for Canadian projects
*item                       marks start of next model and test
*name  basic_AIM_MIN
*cfg   basic_AIM_MIN.cfg
. . .
```

Listing 2: Annotated directives file format.