SCIENTIFIC NOTEBOOK

612-12E

by

Zbigniew Wojcik

Southwest Research Institute
Center for Nuclear Waste Regulatory Analyses
San Antonio, Texas

INITIAL ENTRIES

Scientific Notebook: #612E

Issued to: R. Janetzke

Issue Date: September 17, 2003

Account Number: 20.06002.01.113

Title: TPA 5.0 Code Development

Participants: R. Janetzke
S. Mohanty
R. Rice
C. Scherer
O. Pensado
R. Benke
P. LaPlante
G. Adams
B. Winfrey
M. Smith
Z. Wojcik

Objective:

This scientific notebook will document the work performed in the development of the TPA code.

May 14, 2004

TOPIC: Updates to the TPA interface. Updates to Fortran files ebsfail.f and tpa.inp

Received the task from Ron and Osvaldo to modify the code making the interface with the ebsfail.f module.
Traced the TPA Fortran code and studied the interface.
Prepared the design for the updates to the Fortran file ebsfail.f based on the list of changes and the list of required outputs.
Discussed the requirements and the TPA code with the Ron.
The requirements for the interface were composed of:
* A list of parameters to add or remove: type (e.g. constant, loguniform), name, and value;
* Fragments of ebsfail() subroutine to follow reading and writing these parameters;
* Default input file ebsfail.def for ebsfail()
Studied carefully the Fortran subroutine ebsfail(), the common blocks and other modules of the TPA Fortran Code.
Prepared the proposals for the updates.

May 21, 2004.

TOPIC: Updates to the TPA interface. Modules ebsfail.f and tpa.inp - continuation

Traced consistency of Fortran code when reading the required sample input data by the subroutine Input(). The data resides in the file tpa.inp.
Made short meetings with Ron and Osvaldo to discuss the updates to ebsfail.f.
Implemented the updates. Made sure that parameters not definable by the user are read from the file ebsfail.def, and all parameters definable by the user from the file tpa.inp. Made sure that new constants definable by the user are in the key loop (to be inputted only at the first realization).
Compiled the code by using the command:
make tpa
Tested the code in Fortran: the names of parameters were inputted from in the file tpa.inp together with their type (e.g. constant, triangular, normal, loguniform) and default values. When testing, either the defaults or values corrected by the user were read into the system by ebsfail.f into a look-up table.
Contacted Ron and Carol regarding the issues of handling the TPA code on the Unix .
Contacted Ron to of handle the TPA code on the PC.
Completed the task.

May 28, 2004

TOPIC: Implementation of major changes in the near field environment.

Received the task from Osvaldo and Ron to change the logics of the nfenv() which up to now was directly based on the temperature. From now on, the logics should be based directly on the environmental chemistry, and the corrosion of the drip shield and waste package should be considered separately.
Started major updates to the Fortran file nfenv.f.
Discussed the requirements with the colleagues, Ron and Osvaldo. The requirements were given to me as:
1. The list of tasks, telling of what to introduce and implement.
Corrosion inhibitors had to be introduced to the TPA code, to compute the effective inhibitor and to output it to the file ebstrhc.inp. New parameters Seepage Threshold and Flag Seepage had to be introduced and used in TPA code

based on the explanations. Near-field chemistry had to be changed significantly in the TPA code based on the explanations;
2. Short description of the corrosion equations;
3. List of parameters already handled by the interface in the previous task listed in the' Updates to TPA interface';
4. Explanations of the Seepage Threshold and Seepage Flag to be implemented;
5. Explanations of the chemistry logics to be implemented;
6. List of the new parameters to be used by the chemistry logics in the code;
7. New format of the file ebstrhc.inp to be provided.

Prepared the logics of changes and made the design of the updates.
Verified the logics and the expected changes in the code with Ron and Osvaldo.
(a) Divided of the procedure nfenv() into two subroutines: nfenvFl() and nfenv() with the major change of the logics. After the split the computation in the new nfenvFl() and nfenv() is no longer directly based on the temperature, but on the environmental chemistry. The old logics used the concept of Epoch depending on the location of the temperature peak. Implemented the new logics into two separate subroutines which is based now on the concentrations of the chemicals: Fl, Cl, CO3, NO3, SO4, delta_ECritical and pH. The logics of the new nfenvFl() which I implemented is based on the corrosion of the drip shield by Fl. The logics of the new nfenv() which I implemented is based now on the corrosion of the WP associated with Cl, CO3, NO3, SO4, delta_ECritical and pH. Traced the data flow in the Fortran code by hand.
(b) Divided the subroutine assignConcentrations() into assignConcentrationsFl() and assignConcentrations(). The assignConcentrationsFl() which I implemented assigns concentrations of Fl. The new assignConcentrations(), which I implemented, determines concentrations of Cl, CO3, NO3, SO4, and also assigns delta_ECritical and pH. Traced the data flow by hand.

June 4, 2004

TOPIC: Implementation of major changes in the near field environment - continuation.

Searched for places to add new inhibitors no4, so4.
Studying and clarifying the processing of effective inhibitor. Discussed the issue with Osvaldo.
Computed the effective inhibitor inh in the ebsfail() in file ebsfail.f:

inh = no3(i) + wnwc * co3(i) + wnws * so4(i)

where no3(i), co3(i), so4(i) are concentrations of the inhibitors as a function of time period i, and wnwc, wnws are the ratios of the inhibiting constants:

wnwc = InnerInhibitingCarbonateToCl / InnerInhibitingCarbonateToCl
: wnws = InnerInhibitingCarbonateToCl / InnerInhibitingSulphateToCl

These constants are the input parameters.
Found place to add the effective inhibitor inh to the file 'ebstrhc.inp'. Added the effective inhibitor to the file 'ebstrhc.inp' in ebsfail().
Made corrections to the new nfenvFl() and nfenv(). I placed both codes in the file nfenv.f.
Made corrections to assignConcentrationsFl() and assignConcentrations().
Developed a new module setting qm3peryrperwpinsahitwp() to 0 when temprep() exceeds ThresholdSeepage, and inserted it at the end of nfenvFl(). In addition, qm3peryrperwpinsamisswp() takes the value of qm3peryrperwpinsahitwp() just before qm3peryrperwpinsahitwp() becomes 0.
Studied the TPA code to do the insertion of nfenvFl() and nfenv(). Discussed and received suggestions from Ron.
Made the insertion of nfenvFl() and nfenv() by calling them both from exec.f in the right places.
Received requirement from Ron to pass parameters by the header whenever possible.
Added declarations of the arrays no3(ntim), so4(ntim) with concentrations of corrosion inhibitors in the exec.f and as the header parameters of ebsfail(), where ntim is the number of time periods over the whole time of simulation.

Added new parameters: RewettingHumidity[] (threshold for relative humidity), SeepageThreshold[T] (seepage does not reach drip shield when temperature exceeds it), FlagSeepage[] (activating the SeepageThreshold[T]),

and all the new chemicals concentration parameters, incl. inhibitors for the three environments, as required. Finished development of the Fortran codes to read and use these new parameters.
Compiled and debugged the tpa code with the changes by using:
make tpa
Executed the tpa code with the new near field environment by using:
tpa.e
Started to test: looked at the output file totdose.res before and after changing the model
to see the differences, as suggested by Ron. The results were obtained for initial concentrations of chemicals of the environments I, II and II set to 0. No other concentrations were tested yet to see correctness of the changes to the model.
Tested correctness of the new module modifying qm3peryrperwpinsahitwp() by printing the temprep() and qm3peryrperwpinsahitwp() before and after modifying qm3peryrperwpinsahitwp() by checking visually if the values of qm3peryrperwpinsahitwp() are set to 0 as expected, i.e. when temprep exceeds SeepageThreshold[T].

Received next partial requirements for subsequent testing from Osvaldo and Ron:
* Set initial concentrations of chemicals (incl. inhibitors) of the environments I, II and II set to values above 0 and observe the behavior of Cl and Fl as a function of time periods;
* Observe the temperature at the repository when FlagSeepage[] =1 and then when FlagSeepage[] =0.

June 11, 2004

TOPIC: Implementation of major changes in the near field environment - continuation.

Deleted the input files multibe.dat and multiaf.dat as irelevant. Deleted also statements using the data associated with these files deleted. Deleted also parameters that were no longer in use, as requested by Osvaldo.

Found place to add the effective inhibitor inh to the file 'nfenv.rlt'. Added the effective inhibitor to the file 'nfenv.rlt' in exec.f (just after call to ebsfail()) by extracting inh first from the file 'ebstrhc.inp'. The file nfenv.rlt was used for testing by graphing outcomes of the new environmental chemistry.
Tested the logics:
1. Set SeepageThreshold[T] to 100 to get reasonable number of records with qm3peryrperwpinsahitwp() equal to 0 when Flag Seepage was set to 1. Tested that the values of the array qm3peryrperwpinsahitwp() are set to 0 for the right inputs.
2. Selected values of parameters of RewettingHimidity, SeepageThreshold[T] to receive outcomes for all Environments: Dry, EnvironmentI, EnvironmentII, EnvironmentIII. When testing the code, the RewettingHumidity was set to 0.982. Smaller values (e.g. 0.98) resulted in too small number of the records in Environment II. SeepageThreshold[T] was set to 100, and Flag SeepageThreshold() to 1.0. The values of concentrations of chemicals for Dry period were all set to 0 for the testing. For EnvironmentI they were as follow: EnvI_Cl=0.6[mol/L], EnvI_Fl=0.5[mol/L], EnvI_CO3=0.1[mol/L], EnvI_NO3=0.1[mol/L], EnvI_SO4=0.1[mol/L]. For EnvironmentII they were as follow: EnvII_Cl=0.9[mol/L], EnvII_Fl=0.8[mol/L], EnvII_CO3=0.2[mol/L], EnvII_NO3=0.2[mol/L], EnvII_SO4=0.2[mol/L]. For EnvironmentIII they were as follow: EnvIII_Cl=0.7[mol/L], EnvIII_Fl=0.6[mol/L], EnvIII_CO3=0.05[mol/L], EnvIII_NO3=0.05[mol/L], EnvIII_SO4=0.05[mol/L]. The correct final values of these parameters were not known to me, the above values were set only for testing logics during one realization. These parameters were not distributions but they were constants for the same reasons.
Tested and made sure that the split into the four environments was correct. Tested the logic by printing the values of variables controlling the logic. Observed results for all 201 time periods together with the parameters of the conditional statements setting the logics. The following information was printed for each time step inside subroutines assignConcentrations() and assignConcentrationsFl(): a) Step number; (b) Name of environment (Dry, EnvI, EnvII, EnvIII); concentration of a selected chemical (e.g. of Cl or of Fl); c) Relative Humidity; and d) qm3peryrperwpinsahitwp(). The following constants were printed in front of all the data changeable through the

time steps: Rewetting Humidiry, CriticalRHAq = Critical Relative Humidity Agueous Corrosion. Compared the results printed with the results expected. The results agreed.

Implemented the logics involving the drip shield failure into the nfenv() and assignConcentrations(). The nfenv() and assignConcentrations() deal with environment of waste package. The assignConcentrations() assigns the outcomes to the EnvI, EnvII and EnvIII if drip shield was intact after the dry period. Looked for the information on drip shield failure because of the Fl corrosion. Found this information in dsfailt.dat file. Parsed the file produced after processing each Area to extract the flag reporting on the drip shield (ds) failure and the time period of the ds failure. Installed this parsing of dsfailt.dat file in exec.f. Tested the code parsing the dsfailt.dat. Tested the whole subroutine assignConcentrations() implementing the logic with the chemistry with the drip shield failure. Made graphs in Excell with a few variables as the evidence of the correctness of the logics. Looked for values of parameters of the environment and for the constraints to get all range of outcomes of interest for test, especially after a ds failure.
Made graphs of the variables controlling the logic over the 201 time periods. The graphs were made from the file nfenv.rlt outputted by the tpa.e. The following variables were graphed: temprep() = temperature of repository; clwp() = concentration of chloride; gm3hit() = qm3peryrperwpinsahitwp(); flds = concentration of fluoride; inh = concentration of inhibitors; ThrSee=SeepageThreshold[T] (a constant); RewettingHimidity (a constant); dsFault = drip shield fault time period.
The task was finished with tests using constant concentrations for all the Environments I, II and II. The graphs indicated correct results from processing, because: a) qm3peryrperwpinsahitwp() was set to 0 whenever temprep() exceeds ThrSee=SeepageThreshold[T]; b) when the drip shield fails during environmentII, the concentration of Fl stays at its level, whereas concentrations of Cl and inhibitors jumps to their EnvironmentII concentrations, as expected. The results of the tests were approved by Osvaldo, the field expert.
To eliminate a possibility of a mechanical failure different than because of corrosion, the Seismic Disruptive Scenario Flag was set to 0 in tpa.inp. All tests and graphs were done for NumberOfRealizations set to 1 in tpa.inp.

Started to work on the right sequence of the calls to subroutines in exec.f. For testing, used the sequence:
. . . nfenvFl(), ... dsfault(), ... nfenv() . . .
Started to work on the piece-wise distributions of the inhibitors for the nfenv().

June 19, 2004

TOPIC: Implementation of major changes in the near field environment - continuation.

Finished work on the piece-wise distributions of the inhibitors for the nfenv(). Implemented the read-out of piece-wise distributions of Fl in nfenvFl() by computing the indexes of EnvironmentI_Fl[mol/L], EnvironmentII_Fl[mol/L], EnvironmentIII_Fl[mol/L] inside the ikey IF block, and then executing the valuesp() for these indexes outside the ikey IF block. This way these parameters (incl. piece-wise one) will be read under any distribution into the simulation at each new realization, and these distributions will be modifiable by the user in the file tpa.inp. Implemented the read-out of piece-wise distributions of Cl, CO3, NO3, SO4, DeltaECrit and pH for all three environments in nfenv() in a similar way. Made sure that DeltaECrit is passed to ebsfail() through an array after computing concentrations in subroutine assignConcentraions().

TOPIC: Test of totdose.res data file with major outcomes

The problem was fixed with a rapid jump of total dose at the last time step. The issue is now to test the outcomes after and before the fix. May be, more than one bug was in the simulation program.
The executable before the fix is on /home/janetzke/tpa50t/tpa.e
The executable after the fix is on /home/janetzke/tpa50u/tpa.e
Parameters residing in tpa.inp must be the same.
I copied them to my subdirectory:

cp ../janetzke/tpa50t/tpa.inp .
cp ../janetzke/tpa50u/tpa.inp tpa_u.inp
The parameters were compared by using the diff UNIX command, they happened to be identical.
I worked from my subdirectory by using the tpa50t testing environment after typing:
setenv TPA_DATA /net/spock/home/janetzke/tpa50t/
setenv TPA_TEST /net/spock/home/janetzke/tpa50t/
I have set the NumberOfRealizations to 50 in tpa.inp file for initial tests.
To have some outcomes of execution, I copied tpa.e to my subdirectory:
cp ../janetzke/tpa50t/tpa.e .
and then typed
tpa.e
There were not too many outliers, I could hardly spotted 1 outlier out of 50 realizations.
Then I have set the NumberOfRealizations to 300 in tpa.inp.
This time I could see many outliers.

I developed a Fortran code to detect the outliers. A combined criterium was used, involving average, weighted average and maximum. Average difference la is computed between total dose of the last time step and ct previous time steps. Also, average difference da is computed between the second to the last time step and its ct previous time steps. Similarly, weighted averages wla & wda are computed over the last ct time steps. Maximum difference dm between subsequent 2 time steps is used as well, playing very important role. Its significance stems from the fact, that ct can be a number between 1 and 8. For small ct, e.g. ct=3, which is more appropriate than ct greater than 5, there is no sufficient information about the regular changes in the total dose. The last 2 or 3 time steps do not involve reliable information about the range of the changes. Therefore, more subsequent time steps are used to have more reliable knowledge about regular changes in the total dose. I used 9 last time steps to acquire this knowledge. On the other hand, the total dose is smooth frequently over only 3 subsequent time steps, i.e. non-decreasing or non-increasing. So, adding positive and negative differences over 8 last time steps to compute average would not be correct. I have found the compromise: I took numbers (averages) over 3 last time steps, and I confronted them with the knowledge of the typical changes in the total dose taken from the last 9 time steps. Most reliable knowledge coming from the maximum difference dm between 2 subsequent time steps in the last 9 periods is used as the threshold dmt for average over last ct=3 periods:

$$dmt = w * ct * dm \qquad (1)$$

where w is the weight. I used the weight w=1.2. Obviously, neither the average difference la nor the weighted difference wla is allowed to exceed ct * dm, what is guaranteed by w slightly above 1, if full knowledge exists in the total doses of the last 9 time steps. The above expression for dmt is hardly applicable to hundreds of records with data by hand, therefore tests of results before and after fixing the problem were done by the new Fortran program that I developed.
In addition to the threshold by dmt, also thresholds by da and wda were incorporated:

$$dat = W * da; \qquad wdat = W * wda \qquad (2.a,b)$$

with weight W playing the same role as the weight w. I set W to 1.5.

The outlier is detected, if da > dmt & da > dat & wda > wdat (3)

I developed the testing Fortran program dose_er.f reading the parameters and total dose file, then making the decision and printing the results of detecting the outliers. The parameters are: file name with total doses, e.g. totdose.res, then ct, the weights for the weighted average, and the weights w and W. The output is the file dose_er.out making a table with the fields: rNr, la,wla, da, wda,dm, outlier, where rNr is the time step, and the field outlier has the value 'outlier' or dot if there was no problem for the current realization.
Tested the program dose_er.f.
Run 300 realizations with the problem not fixed (output written to file totdose300_t.res) and 297 realizations with the problem fixed (output written to file totdose300_u.res).
Executed the testing program dose_er.f on the file totdose300_t.res (outcomes with problem not fixed).
Received 195 correct cased and 115 outliers. Tested 20 outliers and 20 correct cases visually comparing results printed by dose_er.f with totdose300_t.res. All cases indicated as outliers by dose_er.f were real outliers in totdose300_t.res. All cases tested and indicated as correct by dose_er.f were correct cases in totdose300_t.res.

Then executed the testing program dose_er.f on the file totdose300_u.res (outcomes with problem fixed). Received 295 correct cased and 2 outliers. Tested these 2 outliers and 20 correct cases visually comparing results printed by dose_er.f with totdose300_u.res. All cases tested and indicated as correct cases by dose_er.f were correct cases when looking at numbers in totdose300_t.res. All these 2 cases indicated as outliers by dose_er.f were real outliers at numbers in totdose300_u.res. One of these two outliers had the maximum difference between two subsequent time steps equal to 0.1207e-7 with the total dose at the last time step equal to 3.9661e-7, and at the previous to the last equal to 3.3919e-7. The second outlier had the maximum difference between two subsequent time steps equal to 0.491e-9 with the total dose at the last time steps equal to 3.6643e-8, and at the previous to the last equal to 3.2408e-8. These two outliers detected indicated that further testing was needed, e.g. proving that higher jumps in total dose in subsequent time steps are likely to occur. Made plots of the two outliers to see if jumps of the magnitude detected by dose_er.f happen in totdose300_u.res in the same realization. They do happen. The plot showed, that: (a) for same realization nr. 48 the jump of about 3 times higher took place about time step 155; (b) inside realization nr. 132 the jump of about 5 times higher took place about time step 175.
As the final conclusion, my tests show that the problem was fixed.

June 25, 2004

TOPIC: Test of totdose.res data file

Completed testing the fix to the totdose.res data file: finished documentation of the Fortran program, and made sure of the conclusion of correctness of fix.

TOPIC: Test of modification of the code reader.f, subroutine writemeanvalues().

Subroutine writemeanvalues() was modified to create new files: tpamin.out and tpamax.out. My task was to test the modifications. Looked at the file reader.f to see the modifications to make the new outputs: they were similar to the making the output to the file tpameans.out, the primary goal of the subroutine writemeanvalues() up to now.
Prepared the testbed: I already had the newest version o tpa code, the tpa50u as subdirectory: ../zwojcik/tpa50u/
The ../zwojcik/tpa50u/ was treated as the reference before the changes.
Made copy of ../zwojcik/tpa50u/ into ../zwojcik/tpa50u_m/:
zwojcik# cp -r tpa50u tpa50u_m
Copied the modified reader.f into ../zwojcik/tpa50u_m/ overwriting previous reader.f. Then installed and executed the pass / fail criteria comparing the ../zwojcik/tpa50u/ with the modified ../zwojcik/tpa50u_m/:
1. Compiled ../zwojcik/tpa50u_m/ by
make
Executed ../zwojcik/tpa50u_m by
tpa.e
The code executed successfully one realization to completion. The outcomes were normal.
2. Compared the output files lhs.out and sp.tpa of interest from the execution of ../zwojcik/tpa50u/ and the modified ../zwojcik/tpa50u_m/. The task was done as follow:
zwojcik# cp ../zwojcik/tpa50u_m/lhs.out lhs_470.out
zwojcik# mv lhs_470.out ../zwojcik/tpa50u/ .
zwojcik# diff -iw lhs.out lhs_470.out
The outcomes lhs.out and lhs_470.out were identical, there were no time stamps in the files.
zwojcik# cp ../zwojcik/tpa50u_m/sp.tpa sp_470.tpa
zwojcik# mv sp_470.tpa t ../zwojcik/tpa50u/ .
zwojcik# diff -i sp.tpa sp_470.tpa
The outcomes sp.tpa sp_470.tpa were the same, except for time stamps in the files in line 3.
3. Compared the output files tpameans.out from the execution of ../zwojcik/tpa50u/ and the modified ../zwojcik/tpa50u_m/. The task was done as follow:
zwojcik# cp ../zwojcik/tpa50u_m/ tpameans.out tpameans_470.out

zwojcik# mv tpameans_470.out ../zwojcik/tpa50u/ .
zwojcik# diff -i tpameans.out tpameans_470.out
The outcomes tpameans.out and tpameans_470.out were the same, except time stamps in the files in line 7.
4. The files tpamin.out and tpamax.out were created successfully when running tpa.e in ../zwojcik/tpa50u_m/. Both tpamin.out and tpamax.out were compared visually on the screen with tpa.inp in in ../zwojcik/tpa50u_m/:
(a) All types constants, iconstants, iFlag were the same in tpamin.out, tpamax.out and tpa.inp;
(b) Types uniform, loguniform, normal, lognormal in tpa.inp were mapped into type constant. The minimum value of two was taken into tpamin.out, and the maximum of two was taken into tpamax.out. The second value is defined as max, and the first value is defined as minimum.
(c) For triangular and logtriangular, the third value (defined as max) was taken into tpamax.out and the first value (defined as minimum) was taken into tpamin.out.
(d) Types beta, finiteexponentail were mapped into type constant. The second value (defined as max) was taken into tpamax.out and the minimum value listed before this maximum was taken into tpamin.out.
(e) Type hazardcurve was mapped into tpamin.out & tpamax.out without a change.
(f) Type correlateinputs was commented out in tpamin.out & tpamax.out, because after converting distributions into constants correlations make no sense: probabilities disappear.
(g) Type exponential was mapped into type constant. The -log(0.0001) / lambda was taken into tpamin.out, where lambda is the value existing in tpa.inp. The -log(0.9999) / lambda was taken into tpamax.out.

The visual testing was simplified: I was carefully looking for about subsequent 3 occurrences of each new type in tpa.inp testing correctness of its mapping into tpamin.out and tpamax.out. I could carefully examine all data specified in tpa.inp, but I understood that all mappings were done by the computer subroutine writemeanvalues() doing everything in the same way. Once one type was mapped, all other occurrences of the same type were mapped in the same way. Ron accepted this simplification of the visual test, in addition, however, that the simplified visual testing would be done the same way also from the end of the files. I did the visual inspection also from the ends of the files. In addition, the tests would fail when using tpamin.out as tpa.inp and independently when using tpamax.out as tpa.inp when executing tpa.e(see points (5) and (6)).
5. The tpamin.out was used as tpa.inp and then tpa.e was executed successfully to completion in ../zwojcik/tpa50u_m/.
6. The tpamax.out was used as tpa.inp and then tpa.e was executed successfully to completion in ../zwojcik/tpa50u_m/.
All tests passed successfully.

TOPIC: Test of ds failure model

Started to test the implementation of drip shield failure. Osvaldo wanted to know how the earliest failure time of ds should be implemented in the new corrosion chemistry model based on the three environments.
Traced that dsfail.f calls standalone program dsfailt.f and that dsfailt.f reads the file 'dsfailt.inp' listing file name fluoride.dat as its first record. File fluoride.dat should list F concentrations over time. Subroutine dsfailttime computes the ds failure time based on the thickness of ds. The conclusion is that the corrosion by F is controlled by the contents of the file fluoride.dat whose format remains the same for the new corrosion chemistry model.

July 2, 2004

TOPIC: Test SCR-485 of modification of the Fortran codes releaset.f and exec.f to eliminate NaN output values when executing on minimum values of tpa.inp.

Designed, developed and made test for TPA SCR-485. Compared TPA code results before and after modifications on the system level. Two System Level (SL) tests were made, both in the run directory /home/zwojcik/scr485/. Both tests compared versions to see if problems disappeared. Both were done for NumberOfRealizations =1. Assumptions made for the two tests: 1. The system TPA is complete in the run directory /home/zwojcik/scr485/; and

2. Files exec_scr485.f and releaset_scr485.f are the modifications of the files exec.f and releaset.f prepared to fix the problem.

The **first SL test** was done to show that the old code produces the NaN values, that modified code runs to completion and that the outputs to the file cpwpfilstats.out are without values NaN. (i.e. that the problem was fixed). Special input file was used: tpamin.out (i.e. minimum values of input parameters tpa.inp) as tpa.inp.
The file wpfillstats.out generated by original tpa.e was compared to output file wpfillstats.out generated by modified tpa.e when using tpamin.out as tpa.inp.
The pass / fail criteria were developed. The test passes if: 1. The modified TPA runs successfully to completion; 2. Screen outputs of the TPA of before and after the modifications are the same when using regular tpa.inp as input (except of the dates of Job Started lines); and 3. The output file wpfillstats.out generated by the modified code is free from values NaN when using tpamin.out as tpa.inp.
During the step by step first SL test procedure:
1. Compiled and executed the code before modifications on basic inputs:
make
tpa.e > tpa_orig.out
2. Saved files in question in the run directory as before modification:
cp tpa.e tpa_orig.e
cp tpa.inp tpa_orig.inp
cp tpamin.out tpamin_orig.out
cp tpamax.out tpamax_orig.out
cp wpfillstats.out wpfillstats_orig.out
The wpfillstats.out had no values NaN (because of not minimum inputs only).
3. Executed the code before modifications when switching tpa input to tpamin.out. Copied the initial input to the tpa50v version to make sure that the inputs are the same:
cp tpamin.out tpa.inp
tpa.e > tpa_min_orig.out
cp tpa_orig.inp . /tpa50v/tpa.inp
4. Executed the code before modifications when switching input to tpamax.out. Copied the screen output to tpa50v version for System Test II:
cp tpamax.out tpa.inp
tpa.e > tpamax_orig.out
cp tpamax_orig.out . /tpa50v/ .
5. Saved tested results wpfillstats.out to see if they have error values NaN :
cp wpfillstats.out wpfillstats_min.out
The results wpfillstats_min.out had error values NaN (as expected before the modifications).
Then, tested the modified code:
6. Got the new system tpa5.0v in subdirectory /home/zwojcik/scr485/tpa50v:
~janetz/gettpa
make
7. Executed the modified code on the regular inputs and on the minimum inputs in /home/zwojcik/scr485/tpa50v:
tpa.e > tpa_485.out
cp tpamin.inp tpa.inp
tpa.e > tpa_min_485.out
8. Observed, that the results of execution before and after modifications were the same (except the dates in Job Started lines):
diff tpa_485.out tpa_orig.out
So, test PASSED.
9. Saved test results wpfillstats.out and saw they had no error values NaN in /home/zwojcik/scr485/tpa50v:
cp wpfillstats.out wpfillstats_min.out
Test PASSED, because of no error values NaN in output wpfillstats_min.out after modifications.

The **second System Level (SL) test** was shorter. I compared versions to see if any side effects exist in the outcomes. I used special input file: tpamax.out as tpa.inp. The objective was to show that modified code runs to completion on some different inputs and the outputs to the file wpfillstats.out are still without any values NaN (i.e. to show that no unexpected side effect exist). The output files to compare were: wpfillstats.out generated by original tpa.e with wpfillstats.out generated by the modified tpa.e when using tpamax.out as tpa.inp. To compare also were the TPA results printed to the screen for the maximum input values before and after modifications. I set the test criteria. The test passes if:
1. The modified TPA runs successfully to completion; 2. The output file wpfillstats.out generated by the modified code is free from NaN values when using tpamax.out as tpa.inp; and 3. The screen output files from the original code and from the modified code are the same when both execute on the same maximum values of input (except for time/date stamps).
During the step by step second SL test procedure in /home/zwojcik/scr485/tpa50v:
1. Executed the code after the modifications when switching input to maximum values tpamax.out:
cp tpamax_orig.out tpa.inp
tpa.e > tpa_max.out
2. Saved tested results wpfillstats.out to see for any error values NaN :
cp wpfillstats.out wpfillstats_max.out
Test PASSED, because the results wpfillstats_max.out did not have error values NaN (because input values were at the maximum).
3. Compared the screen outcomes of the original code with those generated by the modified code:
diff tpa_max.out tpamax_orig.out
The results were the same (except the dates in Job Started lines).
Test PASSED.

Overall test status: **PASS.**

July 9, 2004

TOPIC: Test SCR-487 comparing TPA code results before and after fixing problems caused by the lack of bypassing gapfraction correction for the glass model.

I did the test on the system level.
During the System Level test I compared results from versions before and after modifications and see if problem with not bypassing correction for the glass model disappeared. The path of run directory if initial code is:
/home/zwojcik/scr487/tpa50u/
The path of run directory if modified code is:
/home/zwojcik/scr487/tpa50v/
Program modes used: IModel = 5; OutputMode = 2; SelectAppendFile = 9; NumberOfRealizations = 1.
I set the objective of the test: Show that the problem with not bypassing correction for the glass model was fixed in the modified code. Specifically, I wanted to show that the modified code runs to completion, and I planned to detect small increases or decreases in the output szft.rlt after the modifications, as reported by R. Codell in the change request.
I made the assumptions: 1. The systems tpa50u and tpa50v are complete in the run directory /home/zwojcik/scr487/; and 2. On /home/zwojcik/scr487/, the system tpa50v comprises the modifications with the problem fixed, and tpa50u is the initial version with the problem.
I planned to compare: 1. Output file szft.rlt generated by tpa.e after the modifications in tpa50v, with szft.rlt generated by tpa.e before the modifications in tpa50u; 2. Screen outputs sent to files, before and after the modifications.
I set the pass /fail criteria: The test passes if: 1. The modified TPA runs successfully to completion and their screen outputs are the same, except time stamps; 2. (a) The output szft.rlt generated by tpa.e after the modifications shows small differences as reported by R. Codell in the change request, both increases and decreases; and (b) the outputs

displayed on the screen show differences associated with bypassing for the glass model; 3. There are no values NaN in any output when running the version after the modifications.
I devised, developed and implemented the following step by step **SL1** test procedure:
1. Copy the newest version tpa50v of the TPA system into /home/zwojcik/scr487/tpa50v. It comprises the modifications to the problems:
ssh zwojcik@spock
cd scr487
cd tpa50v
~rjanetz/gettpa
2. Copy the version tpa50u before the improvements into /home/zwojcik/scr487/tpa50u:
cp -Rv ./tpa50u/* ~/scr487/tpa50u
3. Set the parameters in tpa.inp at which the problem occurred, in the version /home/zwojcik/scr487/tpa50u before the modifications:
cp tpa.inp tpa_ini.inp
OutputMode = 2
SelectAppendFile = 9
IModel = 5
cp tpa.inp tpa_bypass.inp
4. Set the same input parameters in tpa.inp in tpa50v. Make sure that the input tpa.inp is the same in tpa50v and in tpa50u. In tpa50v:
cp ../tpa50u/ tpa.inp .
5. Execute tpa in tpa50u:
tpa.e > tpa_no_bypass.out
6. Execute tpa in tpa50v:
tpa.e > tpa_bypass.out
7. Compare their outputs to szft.rlt in tpa50v and save the results of the comparison:
cp ../tpa50u/szft.rlt szft_no_bypass.rlt
diff szft_no_bypass.rlt szft.rlt > diff_bypass
cp szft.rlt szft_bypass.rlt
Observed differences indicated by the change *c* character, both increases and decreases. Most differences are at the first place after the decimal point, and there are practically changes in almost each non-zero number.
Test PASSED.
8. Compare their outputs to the screen in tpa50v and save the differences:
cp /home/tpa50u/ tpa_no_bypass.out .
diff tpa_bypass.out tpa_no_bypass.out > diff_bypass_screen
The outputs were different, as a result of the bypassing when using IModel = 5. The differences were related to the bypassing when using IModel = 5 and were complementary to the differences detected in the files szft.rlt. The differences were indicated by the change *c* character.
Test PASSED.
9. Save the input files both in tpa50u and tpa50v:
cp tpa.inp tpa_bypass.inp
10. See if there are error values NaN in any file in tpa50v:
grep NaN *
No NaN values detected. Also the screen did not show any error values NaN. So, neither any file nor screen output generated NaN values.
Overall result: all the tests PASSED.

July 16, 2004

TOPIC: Next test SCR-487 comparing TPA code results before and after fixing problems caused by division by 0.

I verified that the original code produces the error. I compared results from versions before and after modifications and see if problems with division by 0 disappeared.

The path of run directory if initial code is: /home/zwojcik/scr487/tpa50u/
The path of run directory if modified code is: /home/zwojcik/scr487/tpa50v/
I set in tpa.inp special parameters at which the errors happened: VolcanismDisruptiveScenarioFlag =1; DirectReleaseOnlyFlag =1; NumberOfRealizations =1000; StartAtRealization =668; StopAtRealization =668;
I set the objective: Show that the problem was fixed in the modified code.
I made the assumptions: 1. The systems tpa50u and tpa50v are complete in the run directory /home/zwojcik/scr487/; and 2. The system tpa50v comprises the modifications with the problem fixed, and tpa50u is the initial version with the problem.
I selected the output files to compare: totdose.res generated by original tpa.e in tpa50u, and totdose.res generated by modified tpa.e in tpa50v.
I established the pass /fail criteria: The test passes if: 1. The modified TPA runs successfully to completion; 2. totdose.res generated by modified tpa.e has non-empty dose peak; 3. There are no values NaN in any output when running the version after the modifications.
I designed and implemented the step by step test procedure as follow:
1. The newest version of the TPA system with the modifications to the problems is already in /home/zwojcik/scr487/tpa50v.
The version tpa_50u before the improvements is already in /home/zwojcik/scr487/tpa50u.
2. Set the parameters in tpa.inp listed below at which the problem occurred, in the version /home/zwojcik/scr487/tpa50u before the code modifications:
cp tpa_ini.inp tpa.inp
Then using vi-editor, set these parameters in the file tpa.inp:
VolcanismDisruptiveScenarioFlag =1
DirectReleaseOnlyFlag =1
NumberOfRealizations=1000
StartAtRealization=668
StopAtRealization=668
3. Set the same input parameters in tpa.inp in tpa50v. Make sure that the input file tpa.inp is the same in tpa50v and in tpa50u. In tpa50v:
cp /home/tpa50u/ tpa.inp .
4. Execute the code, both in the /home/zwojcik/scr487/tpa50u before the modifications and in the /home/zwojcik/scr487/tpa50v after the modifications:
tpa.e > & tpa_487.out
The /home/zwojcik/scr487/tpa50v after the modifications executed successfully to completion.
Test PASSED.
5. Compare screen outputs in tpa50v, observe if the errors disappeared in the modified version, and save the results of the comparison:
cp /home/tpa50u/tpa_487.out tpa_487_div0.out
diff tpa_487_div0.out tpa_487.out > tpa_487_diff
Error messages 'Invalid Operation" disappeared in the new version in tpa50v.
The new version (in tpa50v) has a non-zero Peak Mean Dose in non-zero year, whereas the pertinent initial version (in tpa50u) has a zero Peak Mean Dose in zero year.
Test PASSED.
6. Observe the results totdose.res both in tpa50u and in tpa50v:
Save totdose.res resulting from initial version in tpa50v.
In the initial version totdose.res has all zero values in the tede column except for three NaN error values.
The modified version has three non-zero values instead of NaN error values.
cp totdose.res totdose_470.res
Test PASSED.
7. See if there are error values NaN in any file in tpa50v:
grep NaN *
The screen did not show any error values NaN in the modified version.
Test PASSED.
Overall test status: PASS.

TOPIC: Integrating the new near field environment model into TPA 50v.

Copied modified modules / files: nfenv.f, exec.f, ebsfail.f, tpa.inp, failt.f,
ebsfail.def, ebsfail.h, into folder /home/zwojcik/checkin/ for testing by another person.
Run directory: /home/zwojcik/tpa50/
Made a new folder /home/zwojcik/tpa50/ on SUN for the integrated TPA code of the
new near field environment model with the newest version tpa50v,
and started to integrate:
cd tpa50
~rjanetz/gettpa
cd ..
cp nfenv.f tpa50
cp exec.f tpa50
cp ebsfail.f tpa50
cp tpa.inp tpa50
cp ebsfail.def tpa50
cp ebsfail.h tpa50
cp ./codes/failt.f .
cp failt.f tpa50
Compiled and started to test the integrated TPA code in the /home/zwojcik/tpa50/ on SUN.
Started to discuss the issue of testing vs. comparing two models of nuclear waste repository.
Two models will provide different results for sure, and comparing outcomes of simulation from two models is an issue of research study. One way of the study suggested by Ron is to set values of selected or key parameters in one of models to such values that outcomes of the two models will be the same.
Other parameters would be equal, e.g. at mean, median and maximum level, and in general should be at some pre-defined levels. This comparison would provide measures of the differences in terms of changes in selected input parameters.

TOPIC: Generating new file with kd and rd coefficients.

Studied the concepts of fracture kd and fracture rd vs matrix kd and matrix rd. The distribution (or partition) coefficient kd is a ratio (fraction) of a contaminant concentration in soil to that in water. The retardation coefficient rd is the ratio of the velocity of water to that of a contaminant. The array kds(i) is computed by subroutine calc_kd() for each rock layer separately, where i is the contaminant Am, Np, Pn, Th, U. Subroutine calc_rd takes kds(i) and computes the array of retardation coefficients rds(i). Some of Rds are simply read from TPA.INP file.
The specific surface area ssArea parameter is computed for each rock layer and also needs to be collected with all kds(i) and rds(i).
Run directory: /home/zwojcik/kdrd/
Devised and implemented step by step procedure to write surface area, kd and rd to a new file actinide_kdrd.out:

1. Created run directory:
mkdir kdrd
cd kdrd
~janetz/gettpa
2. Studied the TPA code modules pertinent to the problem, understood the details and
found the places of TPA code of modifications in the uzft.f and in szft.f of
saving RealizationNumber, Subarea ID and saturated / unsaturated zone, then
Layer of Rock and SpecificSurfaceArea, then
header listing columns: Nuclide, matrix_KD, matrix_RD, fracture_KD, fracture_RD.
3. Studied the format of the new file actinide_kdrd.out to keep the required parameters.
4. Designed the initial modifications of uzft.f and szft.f to save all the required
parameters in new file actinide_kdrd.out.
5. Modified uzft.f to save the parameters in the new file actinide_kdrd.out.
6. Compiled, debugged and executed the tpa code in the run directory.

July 23, 2004

TOPIC: Integrating new near field environment into TPA50v

Created c:\swri\tpa50w on PC with PC version tpa50v. Copied modified files nfenv.f, exec.f, ebsfail.f, tpa.inp on the c:\swri\tpa50w, failt.f into c:\swri\tpa50w\codes, and ebsfail.def into the c:\swri\tpa50w\data.
Solving a problem with a run time error printed when executing subroutine ebsfail(). Traced the problem: the error message was printed when executing standalone program failt.f called from subroutine ebsfail(). Traced the error in the program failt.f. The error message was printed when executing subroutine input() which reads data from file ebsfail.inp. Traced the read statements causing the error. Located the read statement that fails. The file ebsfail.inp is created by subroutine ebsfail() based on the file ebsfail.def. Compared the file ebsfail.inp with the file ebsfail.def. Found TAB spaces in many places between numerical data and comments following numbers. Removed the TAB spaces from the file ebsfail.def. Executed the TPA again: the problem was FIXED, there were no run time error from the standalone program failt.f. The TAB spaces in the data file were the issue.

CONCLUSION from integrating the new near field environment: To avoid typing parameters into data files with arbitrary characters, using a data interface may be appropriate.

TOPIC: Testing the new near field environment

Collected and discussed with Osvaldo new near field environment parameters needed for the new concept. Received suggestions regarding all values of the parameters to be used for the testing. These suggestions are as follow:
**
constant
EnvironmentI_Fl[mol/L]
1.0e-5
**
constant
EnvironmentI_Cl[mol/L]
1.0
**
constant
EnvironmentI_pH[]
7.0
**
constant
EnvironmentI_NO3[mol/L]
1.0
**
constant
EnvironmentI_CO3[mol/L]
0.0
**
constant
EnvironmentI_SO4[mol/L]
0.0
**
constant
EnvironmentI_Wastepackage_DeltaECrit[VSHE]
0.0
**
constant
EnvironmentII_Fl[mol/L]

1.0e-5
**
usersuppliedpwisecdf
EnvironmentII_Cl[mol/L]
10
0.0, 0.0
0.45, 0.1620
0.95, 0.3175
1.84, 0.3721
1.98, 0.4891
1.99, 0.7042
2.23, 0.9757
2.570.9836
3.1, 0.9978
3.82, 1.0
**
usersuppliedpwisecdf
EnvironmentII_pH[]
6
5.902817, 0.0
6.05, 0.0022
7.0, 0.0164
7.23, 0.0243
8.83, 0.0789
8.84, 1.0
**
usersuppliedpwisecdf
EnvironmentII_NO3[mol/L]
10
0.0, 0.0
0.03, 0.162
0.06, 0.2166
0.07, 0.4881
0.14, 0.6051
0.18, 0.613
0.19, 0.8281
0.21, 0.8303
0.39, 0.8445
0.41, 1.0
**
constant
EnvironmentII_CO3[mol/L]
0.0
**
constant
EnvironmentII_SO4[mol/L]
0.0
**
constant
EnvironmentII_Wastepackage_DeltaECrit[VSHE]
0.0
**
constant
EnvironmentIII_Fl[mol/L]

4.08e-4
**
constant
EnvironmentIII_Cl[mol/L]
6.65e-3
**
constant
EnvironmentIII_pH[]
8.37e0
**
constant
EnvironmentIII_NO3[mol/L]
6.65e-3
**
constant
EnvironmentIII_CO3[mol/L]
2.11e-3
**
constant
EnvironmentIII_SO4[mol/L]
0.0
**
constant
EnvironmentIII_Wastepackage_DeltaECrit[VSHE]
0.0
**
constant
RelativeHumidityForVentilatedAir[]
0.3
**
** ZW SCR478 05-25-04 New to End New ZW
constant
SeepageThresholdT[C]
105
**
constant
FlagSeepageThreshold[]
1
**
constant
RewettingHumidity[]
0.98

TOPIC: Generating new file with kd and rd coefficients.

1. Tested uzft.f with an initial modification intended to save the parameters in the new file actinide_kdrd.out.
Tested a problem: The sub uzft() computes matrix Kds for all layers but does not compute Rds. Why?
Figured out that Kds are computed only as input parameters for Rds. So, Rds following Kds must be computed somewhere else. Found that location of computing Rds in the subroutine prenefmksa(). Realized, that prenefmksa() is also the last place of computing Rds: Rds are not re-computed for unsaturated zone.
(i) Studied, if the equation computing Rds in prenefmksa() is the same as that in subroutine uzft(). The equation specification did not change. Test PASSED.

(ii) Studied if the value of the parameter Porosity does not change. Traced the program by hand and found out that Porosity was the same in prenefmksa() and in uzft() because of using the following statements:

In uzft():

fname='Matrix Porosity_'//salayernam(ilayer)(1:4)

imporosity(ilayer)=ispquery(fname)

porosity=valuesp(imporosity(ilayer))

In prenefmksa():

spname='Matrix Porosity_'//salayernam n(ilayer)(1:4)

por(i,1)=valuesp(spquery(spname))

In prenefmksa() the por(i,1) is the porosity for layer i, and 1 stands for the matrix. So, computing is the same by readout of a parameter under the same name 'Matrix Porosity_'//salaerman(ilayer)(1:4) from tpa.inp file: test Porosity value PASSED.

Then executed the TPA code to see the values of Porosity on the screen, both in prenefmksa() is the same as that in uzft(). In prenefmksa() they were the same as in uzft(). Test PASSED.

(ii) In a similar way, tested the values of the parameter Density. The values were the same in prenefmksa() and in uzft(). Test PASSED.

2. Modified uzft() to save the parameters in the new file actinide_kdrd.out: a. Declared the names of the rock layers specific to unsaturated zone; b. Designed with details and implemented the transfer of the new file actinide_kdrd.out for the append operations between calls of subsequent realizations; c) Designed, implemented and tested the mechanism of temporal saving parameters of matrix kds() from the call to calc_kd() (computing the matrix Kd equations) to save them in file actinide_kdrd.out together with the fracture Kd computed later. The loop through rock layers was over the matrix and fracture Kd an Rd computations; d) Done the same for the Rd parameters, however, added new call to calc_rd() in uzft() to get all matrix Rd; e) Followed the required format for the new file actinide_kdrd.out.

3. Tested the modifications made to uzft(): compared the matrix Rds computed in uzft() as a result of the modification with matrix Rds computed in prenefmksa(). Made the comparison at the corresponding layer rocks and elements. The calc_kd() computes Kds for the nuclides in the sequence: Am, Np, Pu, Th, U. The prenefmksa() extracts these matrix Kds, associates them with the elements Am, Np, Pu, Th, U and then computes matrix Rds. Tested this association, which was correct. Test PASSED.

The Rds computed in prenefmksa() were associated with the right elements Am, Np, Pu, Th, U as in uzft() and were of the same values. So, the Rds computed for file actinide_kdrd.out in uzft() were correct. Test PASSED.

4. Made another, parallel test of correctness of passing matrix Kds from uzft() to prenefmksa() to compute RDs. Studied the call (in uzft.f) to sub setconsmv() saving results of the first call to calc_kd as matrix kd. The values of matrix kd are saved in datc() by layer rock (see common / mvb4 /, include 'mve.i'). The indexes to Kds corresponding to Am, Np, Pu, Th, U extracted by invquery() were the same both in uzft() and in prenefmksa(). So, the indexes to Kds (and Rds) whose values were saved in file actinide_kdrd.out were associated with the right elements Am, Np, Pu, Th, U and with their right values of Kds and Rds existing in prenefmksa(). The second test PASSED.

6. Tested if the values under fields Nuclide, matrixRD, matrixKD, fractureKD, fractureRD computed in uzft.f were written correctly to the file actinide_kdrd.out by comparing the values printed to the screen during execution with the contents of the file actinide_kdrd.out. They were the same. Test PASSED.

July 30, 2004

TOPIC: Generating new file with kd and rd coefficients.

1. Studied, designed and modified szft.f to save the RD parameters computed by calc_kd() and calc_rd()) into the new file actinide_kdrd.out: a. Declared the names of the rock layers specific to saturated zone; b. Implemented the transfer of the new file actinide_kdrd.out for the append operations between calls of subsequent realizations; c) Designed, implemented and tested the mechanism of transfer of parameters kds() from the call to calc_kd() computing the matrix KD equations to save these matrix parameters in new file actinide_kdrd.out after calling calc_kd() to compute the fracture KD equations. The differences with uzft were: (i) the loop through rock layers was only over the matrix KR an RD computations; (ii) the call to make matrix RDs() parameters were not made from

another module; (iii) the fracture parameters were only for the layer 2 of rock, so the matrix parameters had to be transferred to the file actinide_kdrd.out for the layer 2 of rock only. Studied computation of the parameters Kds in calc_kd() based on the file coefkdeq.dat (inputted by subroutine uzft() into 3D array params()).

2. Studied, designed and modified szft.f to save the RD parameters taken from TPA.inp file (not computed by calc_kd() and calc_rd()) into the new file actinide_kdrd.out. Studied two loop structures over which all the elements are scanned, then transfer of these nuclide through rock layers, incl. all controls and save operations inside the body of the loop structures. Studied and utilized the differences between the number and names of the nuclides handled when computing Rds by using calc_RD() and when reading RD coefficients from tpa.inp: only 5 nuclides are processed by using calc_kd() and calc_rd(): Am, Np, Pu, Th, U, whereas all nuclides are processed when reading Rds from tpa.inp. The array radelems() keeps the names Am, Np, Pu, Th, U, whereas all full names of nuclides are kept by the array namelj(). Identified and saved temporarily the required RDs inside the body of the loop structures. Declared new arrays to collect temporarily the mobile and immobile nuclides. Resolved the differences between the number and names of the nuclides handled when computing Rds by using calc_RD() and when reading RD coefficients from tpa.inp: only 5 nuclides are processed by using calc_kd() and calc_rd(): Am, Np, Pu, Th, U, whereas all nuclides are processed when reading Rds from tpa.inp.

3. Tested. Compared the matrix Rds for the elements Am, Np, Pu, Th, U computed by calc_rd() in szft() with those collected from tpa.inp for two layers: STFF and SAV. They were the same.
The value of fracture RD computed by calc_rd() was always equal to 1. This is because the value frac_forcefac is always equal to 0, because that frac_forcefac = valuesp(ifforcefac), and that ifforcefac = ispquery(UZFractureForceFactorForKdToRd). UZFractureForceFactorForKdToRd = 0 in tpa.inp. So, the values of RDs were as expected.
Tested, if the values RD, Nuclide, SpecificSurfaceArea, Realization Number and Subarea, printed to the file actinide_kdrd.out, are as printed to the screen. All they were the same. Test PASSED.
Got another problem when testing: the outputs of Kds computed by calc_rd() were for only the first area of the first realization. Tested, why. Found the condition in the code:

```
IF (ir .ne. ir_now) THEN
   Making actinide_kdrd.out
  END IF
```

where ir is the current realization number. Found out, that this condition is true only for the first area of the first realization, and this explains, why. Test PASSED, because the outcomes follow the logics as intended.
Verified, that fracture Rds are not collected from tpa.inp, and so they are not printed out.
4. Made meeting with Paul Bertetti and Ron. Paul requested that only one RD value per element per layer is needed.

August 6, 2004

TOPIC: Generating new file with kd and rd coefficients.

Made meeting with Paul and discussed the Kds and Rds needed for his testing.
To the general requirements to provide the Rds in both unsaturated and saturated zones, Paul requested in addition also the code outputting only the Rds and Kds in the saturated zone. Made updates on the copy of the TPA code described above: commented-out all write statements resulting Kds and Rds from the unsaturated zone, and modified the szft.f to output only the Kds and Rds for the first area only on each realization.
Delivarables:
subdirectory kdrd - tpa version providing Kds and Rds for the saturated zone only;
subdirectory kdrd_all - tpa version providing Kds and Rds for both the unsaturated and saturated zones.
Made next meeting with Paul and discussed the Kds and Rds. Gave the files uzft.f, szft.f, exec.f to Paul together with the file actinide_kdrd.out with Kds and Rds for the saturated zone. Paul was satisfied with the deliverable.

TOPIC: Fixing the problem with truncation of lines in ebsfail.inp.

Verified, that lines are in fact truncated in ebsfail.inp, by visual inspection of the file ebsfail.inp in the run subdirectory.
Noticed, that a variable 'aline' was declared in ebsfail.f as:
character *60 aline
Re-defined it to:
character *256 aline
Noticed also, that the format for the file operations was:
100 Format(a60)
Changed this format to:
100 Format(a256)
File affected: ebsfail.f.
Inserted statement:
aline=trim(aline)
at each write to the file ebsfail.inp to prevent keeping blanks at the end of each line.
Tested the correctness of these modifications by comparing ebsfail.inp with ebsfail.def. Now the lines in ebsfail.inp were without truncations. Test passed.

TOPIC: Fixing the problem with excessive allocation of resources by igetunitnumber() in exec.f when opening file ebsthrc.inp.

File affected: exec.f.
Noticed, that igetunitnumber() was called in the main loop. Moved the call to igetunitnumber() for file ebsthrc.inp in front of the main loop.

TOPIC: Fixing another problem with excessive allocation of resources by igetunitnumber() in nfenv.f.

File affected: nfenv.f.
Noticed, that igetunitnumber() was called in subroutines reflux3(), tabulartepmrh(), assignConcentrations() each time a procedure is called.
Implemented the following structure allowing to call igetunitnumber() only once for all realizations, even if a procedure is called in each realization:

```
Integer iunit, key_sub
Common / subx_com / iunit, key_sub
. . . . .
If (key_sub .ne. 8492746 ) then
   iunit=igetunitnumber()
   key_sub = 8492746
   end if
 Open(iunit, file=....)
```

TOPIC: Fixing the problem with using Open(12, file=....) instead of Open(iunit, file=....) in subroutine reflux2() in nfenv.f.

Implemented the structure as specified above in subroutine reflux2() preventing possible conflict with a file unit = 12 allocated somewhere by igetunitnumber().

TOPIC: Integrating projects on Kd and Rd coefficients.

Integrated my kdrd_all TPA code extracting values of Kds and Rds into file acticide_krds.out (files szft.f, uzft.f, exec.f) with Carol's work on Kd and Rd coefficients (files szft.f, uzft.f, tpa.inp, mod_rdrdi.h, mod_kd.h, nefmks.f). Resulting TPA code is in the subdiretory kdrs_integr.
Steps of integration:
1. Copied the kdrd_all into kdrs_integr
2. Saved szft.f, uzft.f tpa.inp as szft_1.f, uzft_1.f, tpa_1.inp respectively in kdrs_integr

3. Overwrote szft.f, uzft.f by Carol's szft.f, uzft.f in kdrs_integr
4. Saved nefmks.f as nefmks_1.f in kdrs_integr\codes
5. Overwrote nefmks.f by Carol's nefmks.f in kdrs_integr\codes
6. Opened uzft_1.f and uzft.f and copied one by one all modifications of uzft_1.f marked with ZW into uzft.f (in kdrs_integr).
7. Opened szft_1.f and szft.f and copied one by one all modifications of szft_1.f marked with ZW into szft.f (in kdrs_integr).
8. Compiled and executed.
9. Tested the integration regarding correctness of the file actinide_kdrd.out. The format and quantities were correct, though particular values of Kds and Rds were different, what was expected. Compared results of extracting KDs and RDs of version before integration with that after integration. Results were the same. Test PASSED.

TOPIC: Solving problem with NaN values in ebstrhc.inp file for 1024 realizations in the tpa version comprising the new near field environment.

1. Made a test bed in the subdirectory tpa50_nf to verify and get ready to fix the problem. Found the location of the write into the file ebstrhc.inp in the ebsfail.f. Added statements to the ebsfail.f code printing to a special test file ebs_tst.txt all data that are written to the ebstrhc.inp file.
2. Set NumberOfRealizations to 1024 in tpa.inp file.
3. Compiled, executed.
4. Searched for NaN value in file ebs_tst.txt. The file ebs_tst.txt is long, about 150MB.

No NaN value found. Test PASSED. Perhaps the NaN values exist for special tpa.inp data, I only set NumberOfRealizations to 1024. Will ask for parameters in tpa.inp. A likely explanation is, that NaN values disappeared after fixing the problem with excessive allocation of resources by igetunitnumber() in exec.f when opening file ebsthrc.inp.
Result:
ebs_tst.txt file in tpa50_nf\run.

TOPIC: TEST by duplicating results of old version on modified code using old burnup and nuclide data, SCR-483

Performed the tests under SCR-483 on PC. Used the baseline version: TPA 5.0v, and test Version: TPA 5.0w.
All modified / new source code files, affected old version (before modifications), and all affected input / output files are kept on the disk titled "SCR#483. Test Results. Z. Wojcik". A separate subdirectory 'test_results' exists keeping only the results of tests. The names of the files keeping the results of tests of the old version have the 3 characters '_v_', and the names of the files keeping the results of tests of the new version are without these 3 characters.
Performed several tests on the PROCESS LEVEL (PL).
Path of run directory if initial code for all tests: /swri/tpa50v/.
Path of run directory if modified code for all tests: /swri/tpa50w/.
Used NumberOfRealizations =1 in tpa.inp for all tests.

The test PL1 (PROCESS LEVEL 1) was to examine logics of invent.f reading data file burnup.dat in new version TPA50w.
Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\
Path of results: \scr483\test_results\
Special input files or modifications to input files: burnup.dat
Program modes to be used: NumberOfRealizations =1.
Description of PL1 Test Plan:
- Objectives: Test if the logics of the new program handles correctly the modifications in the input file burnup.dat

- Assumptions: All modification are marked in the new file invent.f by scr 483.
- Input file to test: burnup.dat.
- Fortran file to test: invent.f
- Pass / fail criteria: The test passes if the new program reads the modified file burnup.dat correctly, and the modifications in the Fortran code accommodate to the changes in the input files in question. The correctness of the read-out is tested by visual comparison of the printout of the input file burnup.dat with the print of the contents of the arrays into which burnup.dat is read in the program.

PL1 Test Procedure:
Tested the logics of readout of the file burnup.dat by comparing with that of the former version.
The format of the new file burnup.dat was slightly different. Checked, that the new file burnup.dat is read by the new invent.f. Tests passed.
(a) The line starting with word TITLE is skipped as needed; (b) The comments starting with ** are skipped as needed; (c) the yearofburnupdata (time 0) is read as needed; (d) bwrblend (boil water reactor blend) and pwrblend (pressure water reactor blend) are read as needed; (e) header (text) line is skipped; (f) data values (Time, bwr, pwr) are read in a loop until the end of the file. Test PASSED.
Noticed that there are no sufficient checks in the code of invent.f detecting errors in input data file burnup.dat. But burnup.dat changes, for instance the values of bwr, pwr of the new version are different compared to the old version, and may change in the future.
New modifications made: added the following checks: (a) check in the number of all data records read, and (b) check whether the Time is increasing and each of subsequent values of bwr, pwr are decreasing.
PL1 test **Results:**
Saved the contents of the arrays bwr() and pwr() in the file scr483_r when burnup.dat is correct. All saved results are in the subdirectory test_results.

PL2:
Name: See, how input file burnup.dat is read when its data is malformed by removal of one data value.
Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\
Path of results: \scr483\test_results\
Special input files or modifications to input files: burnup.dat malformed by removal of one data value.
Program modes to be used: NumberOfRealizations =1
Description of PL2 Test Plan:
- Objectives: See, how input file burnup.dat is read when data in it is malformed by removal of one data value.
- Assumptions: All modification are marked in the new file invent.f by scr 483.
- Input file to test: burnup.dat.
- Fortran file to test: invent.f
- Pass / fail criteria: The test passes if the read-out is robust to data malformed in burnup.dat by removal of one data value.

PL2 Test Procedure and Results:
Tested to see how the data are read when the new input file burnup.dat is formed correctly vs. data in burnup.dat is malformed by removal of one data value.
Made a test bed and compared the print of the tested input file burnup.dat with the print of the contents of the arrays bwr() and pwr() comprising data read from the burnup.dat.
Data file was malformed by removal of one data value from burnup.dat.
Tested outcomes of reading the file burnup.dat when no data was missing vs. one number was erased in the file burnup.dat. When data in the file burnup.dat was corrupted (one value was missing), there was no error message printed to the screen and data in the arrays bwr() and pwr() was incorrect (the number of items was less by 1 for 1 value missing and data were shifted down by 1 from the location of the missing value).
Saved the test bed in invent_r.f. Saved the contents of the arrays bwr() and pwr() in the file scr483_er when burnup.dat is corrupted (one data value was missing). Saved the contents of the arrays bwr() and pwr() in the file scr483_r when burnup.dat is correct. All saved results are in the subdirectory test_results.
Fixed the problem. After fixing, error message is printed and the execution is stopped. Test passed.

PL3:

Name: See, how input file burnup.dat is read when its data is malformed by inserting extra dot.
Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\
Path of results: \scr483\test_results\
Special input files or modifications to input files: burnup.dat with extra dot inserted.
Program modes to be used: NumberOfRealizations =1
Description of PL3 Test Plan:
- Objectives: See, how input file burnup.dat is read when data in it is malformed by extra dot.
- Assumptions: All modification are marked in the new file invent.f by scr 483.
- Input file to test: burnup.dat.
- Fortran file to test: invent.f
- Pass / fail criteria: The test passes if the read-out of burnup.dat is robust to data malformed by extra dot.

PL3 Test Procedure and Results:
Tested to see how the data are read when the new input file burnup.dat is formed correctly vs. data in burnup.dat is malformed.
Used the same test bed and compared visually the print of the tested input file burnup.dat with the print of the contents of the arrays bwr() and pwr() comprising data read from the burnup.dat.
Data file was malformed by inserting extra dot in the middle between data values in burnup.dat.
Program read the dot as value 0.0, subsequent values were shifted right by one column, and the last value in the record was lost. The read out was incorrect, no error message was printed, but the error did not affect the read out of other records. Test failed.
Fixed the problem. After fixing, error message is printed and the execution was stopped. Test passed.
The test bed is in file invent_r.f in \scr483\test_results\. Saved the burnup.dat malformed by extra dot as burnup.dat.
The scr_483_dot.tst file keeps the contents of the arrays bwr() and pwr() with the values read.

TOPIC: Fixing problems with non-robust read-out of file burnup.dat

The work started when testing the readout of a malformed text file when reading it in Fortran.
To test formatted and unformatted Fortran read statements, a few small deletions and insertions were made to deform a file.
Designed, developed and tested the module detecting error in burnup.dat.
a. Data file was malformed by removal of one data value from a text file.

Added the count ir of all records read from the file as strings, and then compared this count with the ia = the count of actual number of records read from burnup.dat with the aid of formats for the expected numerical data. When the data is malformed then the readout of the record using numerical formats is skipped, and then the count ia is reduced by 1. Implemented the module in the subroutine newinventdb().
Tested the new feature by executing the tpa code on the correct burnup.dat and on the burnup.dat with one missing value. When one data was missing, the program printed the error message and stopped. When all data was correct, the program executed normally without error the message.

Used a typical unformatted Fortran file read statement when one number was erased in a text file. When data in the file was corrupted (one value was missing), there was no error message printed to the screen and numbers in the arrays storing the data were incorrect. The number of items was less by 1 for one value missing, and data were shifted down by 1 from the location of the missing value. All data read may be affected by the error without any error message printed when using unformatted Fortran file read statement.

Formatted read did not print and error message and took subsequent file value under the variable to read (i.e. with error). No other data records were affected by the error when using formatted Fortran file read statement.

Making read-out robust by counting data records when using unformatted read statement:
Fortran ignores (skips) malformed data record when a value is missing when using unformatted Fortran file read statement. This way, because a missing value reduces the number of data records read by 1, the idea is to first read all records as text only (in format 'a') to count the number of data records, and then to read the file again to read data by using specific formats incorporating data types.
To fix the problem, developed an application program that makes the pilot read record by record into textLine character string to the file end, and then rewinds the file:

```
      ir = 0
      do while (.true.)
          read(ireadfile,'(a)',end=14) textLine
          ir = ir + 1
         enddo
14    rewind ireadfile
```

Then the application program reads the columns of data into arrays, and counts, using the count ia, the records not skipped because of missing values:

```
      i = 0
      do while (.true.)
          i = i + 1
            read(ireadburnupfile,*,end=40) times(i),bwr(i),pwr(i)
            ia = ia + 1
            enddo
 40       continue
```

Finally, the application program compares the number ia of data records read with the number ir of records expected to be correct. Error message is printed and the execution is stopped:

```
      if (ir .ne. ia) then
          print *, ' **>> Error reading data file <<** '
          print *, ' Possible reasons: missing data value'
          STOP
          endif
```

Developed the above module and installed it in the invent.f. Tested it by using the burnup_r.dat file. Test passed. Saved the test bed in invent_r.f. Saved the contents of the arrays bwr() and pwr() in the file scr483_er when burnup.dat is corrupted (one data value was missing). Saved the contents of the arrays bwr() and pwr() in the file scr483_r when burnup.dat is correct. All saved results are in the subdirectory test_results.

Then devised, implemented and tested a module checking, if all values of bwr and pwr are in the decreasing order as expected. The algorithm was simply checks if the new value read from burnup.dat is smaller compared to respective value read in the previous iteration. Tested the module. Used burnup.dat with one value too high making that not all values were in the decreasing order. Saved this malformed file as burnup_or.dat. When executing on this malformed file, the error message was printed that the order was not decreasing and the program stopped. When data was correct, there was no error message and the program executed normally.

Similarly, devised, implemented and tested a module checking, if all values of times are in the increasing order. Tested the module. Used burnup.dat with one value of time too high making that not all values were in the increasing order. Saved this malformed file as burnup_ti.dat. When executing on this malformed file, the error message was printed that the order was not increasing and the program stopped. When data was correct, there was no error message and the program executed normally.

b. Data file was malformed by inserting extra dot in the middle between data values.

Typical unformatted Fortran file read statement takes the dot as value 0.0, subsequent values were shifted right by one column (the effect of extra 'digit'), and the last value in the record was lost. The read out was incorrect, no error message was printed, but the error did not affect the read out of other records. With a formatted read the system prints an error message and the execution is stopped.

Making read-out robust by testing expected order of values (e.g. ascending)

When data is expected in a specific order, the values read can be tested if they are in that order:

```
      do 17 i=1,itimes
         if (i.gt.1) then
           if ( it .ge. times(i) ) then
             print *, ' **>> Error <<** '
             print *, ' times() must be in increasing order'
             STOP
             end if
           end if
         it=times(i)
```

17 continue
The burnup.dat data is sorted in columns. Dot that is treated as 0.0 changes this expected order, what allows to detect dot in unformatted read Fortan statement.
Developed the above module and installed it in the invent.f. Tested it. Saved the test bed in invent_r.f. burnup_r.dat file. Test passed.

August 13, 2004

TOPIC: TEST by duplicating results of old version on modified code using old burnup and nuclide data, SCR-483. Continuation.

The test PL4 (PROCESS LEVEL 4) was to see, how input file burnup.dat is read when its data is malformed by inserting an extra non-numerical character.
Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\
Path of results: \scr483\test_results\
Special input files or modifications to input files: burnup.dat malformed by an extra non-numerical character.
Program modes to be used: NumberOfRealizations =1
Description of PL4 Test Plan:
- Objectives: See, how input file burnup.dat is read when data in it is malformed by inserting an extra non-numerical character.
- Assumptions: All modification are marked in the new file invent.f by scr 483.
- Input file to test: burnup.dat.
- Fortran file to test: invent.f
- Pass / fail criteria: The test passes if the read-out is robust to data malformed in burnup.dat by inserting an extra non-numerical character.
PL4 Test Procedure and Results:
Tested to see how the data are read when the new input file burnup.dat is formed correctly vs. data in burnup.dat is malformed.
Used the same test bed and compared visually the print of the tested input file burnup.dat with the print of the contents of the arrays bwr() and pwr() comprising data read from the burnup.dat.
Data file was malformed by inserting an extra non-numerical character in the middle between data values in burnup.dat. System printed error message, execution was stopped. Test passed. No fix was needed.
The test bed is in file invent_r.f in \scr483\test_results\.

PL5:
Name: See, how input file burnup.dat is read when its data is malformed by inserting one extra digit.
Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\
Path of results: \scr483\test_results\
Special input files or modifications to input files: burnup.dat
Program modes to be used: NumberOfRealizations =1
Description of PL5 Test Plan:
- Objectives: See, how input file burnup.dat is read when data in it is malformed by inserting one extra digit.
- Assumptions: All modification are marked in the new file invent.f by scr 483.
- Input file to test: burnup.dat.
- Fortran file to test: invent.f
- Pass / fail criteria: The test passes if the read-out is robust to data malformed in burnup.dat by inserting one extra digit
PL5 Test Procedure and Results:
Tested to see how the data are read when the new input file burnup.dat is formed correctly vs. data in burnup.dat is malformed.

Used the same test bed and compared visually the print of the tested input file burnup.dat with the print of the contents of the arrays bwr() and pwr() comprising data read from the burnup.dat.
Data file was malformed by inserting one extra digit in the middle between data values in burnup.dat. Program read the extra number, subsequent values were shifted right by one column, and the last value in the record was lost. The read out was incorrect, no error message was printed, but the error did not affect the read out of other records. Test failed.
Fixed the problem. After fixing, error message is printed and the execution is stopped. Test passed.

Data formed correctly are read from the files without problems. Any value in a file can be incorrect, what is harder to detect than spotting a missing value. Data files must be carefully tested. It can be seen how the minor tests for robustness failed for the initial code. The corrected readout was free from the problems.
The readout method and the structure of burnup.dat in the new version is the same as in the old version, probably robustness of reading data from burnup.dat was not tested before.
To fix the problems:
1. Added a test for the number of values recorded in the arrays from the file, because one missing value may change all other input values, vs. one incorrect (e.g. miss-typed) value does not change other input values.
2. Tested in the program, if the time values read are increasing and all the subsequent values of arrays bwr() and pwr() are decreasing. Each problem stopped the program execution with a burnup.dat read error message. Tests passed.
The test bed is in file invent_r.f in \scr483\test_results\.

PL6:
Name: Test logics of invent.f when reading data file nuclides.dat in new version TPA50w
Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\
Path of results: \scr483\test_results\
Special input files or modifications to input file: nuclides.dat
Program modes to be used: NumberOfRealizations =1
Description of PL6 Test Plan:
- Objectives: (a) See if the logics of the new program handles correctly the modifications in the input file nuclides.dat;
(b) Check logics of the key changes in the invent.f.
- Assumptions: All modification are marked in the new file invent.f by scr 483.
- Input file to test: nuclides.dat.
- Fortran file to test: invent.f
- Pass / fail criteria: The test passes if:
(a) The new program reads the modified file nuclides.dat correctly, and the modifications in the Fortran code accommodate to the changes in the input files in question. The correctness of the read-out s tested by visual comparison of the printout of the input file nuclides.dat with the print of the contents of the arrays into which the read is made in the program; (b) Logics of the key modifications in the program is correct.
PL6 Test Procedure:
1. Tested the logics of inputting the modified file nuclides.dat. Compared with the previous file nuclides.dat and noticed differences. A new parameter exists:
yearofinventorydata=2033.
Practically all values in the fields CPMTU and CPMGLASS are now different because of the decay process. The number of nuclides iNuclides=57 is inputted first and then iNuclides records of data are read. The logic of inputting data from nuclides.dat is correct. Test PASSED
2. Tested the logic of the new three key statements:
storetimeatemplacement = yearofemplacement - yearofinventorydata (1)
time(2) = storetimeatemplacement (2)
tyr = time + yearofemplacement - yearofburnupdata (3)
The yearofemplacement is inputted in invent.f from tpa.inp from under the name CalendarYearOfEmplacement[A.D.]=2034.
The yearofinventorydata=2033 is inputted from the file nuclides.dat.

The time(2) is the storage time at emplacement, now data is different and so time(2) no longer adjusts to the data (by subtracting 10). This adjustment is no longer valid in tpa50w and so is no longer used.
A new valuable check exists printing an error message when yearofburnupdata + time(1) > yearofemplacement (see the next two tests).
Tests PASSED.
PL6 test Results:
Saved the test bed in invent_r.f (together with the test bed for testing the readout of burnup.dat).
All saved results are in the subdirectory test_results.

PL7:
Name: Test visually and by hand a new module printing an error message when
yearofburnupdata + time(1) > yearofemplacement
in new version TPA50w.
Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\
Path of results: \scr483\test_results\
Special input files or modifications to input file: nuclides.dat
Program modes to be used: NumberOfRealizations =1
Description of PL7 Test Plan:
- Objectives: Test printing an error message when yearofburnupdata + time(1) > yearofemplacement in the invent.f in new version TPA50w
- Assumptions: All modification are marked in the new file invent.f by scr 483.
- Input file to test: nuclides.dat.
- Fortran file to test: invent.f
- Pass / fail criteria: The test passes if:
(a) Logics of the statement yearofburnupdata + time(1) > yearofemplacement is correct; (b) A test by hand proves that the error message is printed when yearofburnupdata + time(1) > yearofemplacement.
PL7 Test procedure:
1. Checked visually the logics of the module

```
if (yearofburnupdata + times(1) .gt. yearofemplacement) then
      print *, ' ***>>> Error in newinventdb <<<*** '
      print *, ' First entry in burnup.dat applies to a time'
      print *, ' greater than the emplacement time. '
      End if
```

The yearofburnupdata + time(1) > yearofemplacement is the correction for the difference between the date the waste is placed into repository and the date of the first burn-up data in the file burnup.dat.
The yearofemplacement comes from tpa.inp, see the constant name CalendarYearOfEmplacement[A.D.] set to 2034.
The yearofburnupdata=2033 is the first number read from the file burnup.dat. The time(1)=1 is the first time [year] read from the file burnup.dat.
2. Tested the statement
yearofburnupdata + time(1) > yearofemplacement by hand,
where: yearofburnupdata=2033, time(1)=1, yearofemplacement=2034, getting:
i.e., 2033 + 1 >2034 to print the error message for this data. Test Passed.
PL7 test Results:
Inequality
yearofburnupdata + time(1) > yearofemplacement
for: yearofburnupdata=2033, time(1)=1, yearofemplacement=2034, gives:
2033 + 1 >2034 to print the error message for the current setting of data.

PL8:
Name: Test of a new module printing an error message when
yearofburnupdata + time(1) > yearofemplacement in new version TPA50w
Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\

Path of results: \scr483\test_results\
Special input files or modifications to input file: nuclides.dat
Program modes to be used: NumberOfRealizations =1
Description of PL8 Test Plan:
- Objectives: Test printing an error message when yearofemplacement - yearofburnupdata -time(1) <0 in the invent.f in new version TPA50w
- Assumptions: All modification are marked in the new file invent.f by scr 483.
- Input file to test: nuclides.dat.
- Fortran file to test: invent.f
- Pass / fail criteria: The test passes if
the error message is printed only when yearofburnupdata + time(1) > yearofemplacement.
PL8 Test Procedure:
Made a test bed by changing the CalendarYearOfEmplacement[A.D.] in tpa.inp to 2033.
Got the error message and the program execution stopped as expected :
'First entry in burnup.dat applies to a time grater than the emplacement time.
. . . '
With the correct values of the arguments to yearofburnupdata + time(1) > yearofemplacement no error message is printed, as required. Test passed.
PL8 test Results:
Saved the test bed tpa.inp as tpa_er.inp. All saved results are in the subdirectory test_results.

PL9:
Name: Tests of robustness of invent.f when data file nuclides.dat is malformed by deleting one value.
Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\
Path of results: \scr483\test_results\
Special input files or modifications to input file: file nuclides.dat, one data value deleted.
Program modes to be used: NumberOfRealizations =1
Description of PL9 Test Plan:
- Objectives: See if invent.f is robust when reading data from malformed file nuclides.dat.
- Assumptions: All modification are marked in the new file invent.f by scr 483.
- Input file to test: nuclides.dat.
- Fortran file to test: invent.f
- Pass / fail criteria: The tests pass if the readout is robust to data malformed by deleting one value.
PL9 Test Procedure and Results:
Tested to see how the data is read when the new input file nuclides.dat is malformed by deleting of one value under column epalim(iNucIndex) in record 4. The malformed file nuclides.dat was saved as nuclides_nr.dat.
Made a test bed (see invent_nr.f) and compared the print of the correct input file nuclides.dat with the print of the index i=iNucIndex and the contents of the arrays epalim(iNucIndex), cipermtuat10yr(iNucIndex), cipermtuat10yrglass(iNucIndex), idecaytype(iNucIndex) comprising the last 4 columns of the data read from the corrupted file nuclides_nr.dat.
No error message was printed and the results of reading were corrupted (see file scr483_nr.tst). Test failed.
Formatted Fortran read statement took digits only from the strictly predefined fields ignoring even a piece of a number if it belonged to the blank field nx. Positive result was of not corrupting any other records (compare records 5 up to end of the file scr483_nr.tst with the last 4 columns of the initial file nuclides.dat).
Fixed the problem by applying unformatted read (see file invent_y.f). With this fix the program does not read correctly, but detects the problem, prints the error message and stops execution. Test passed on corrupted input file nuclides_nr.dat.
Tested also the read of the correct file nuclides.dat using the fixed code. Saved the result of reading in file scr483_n.tst. Test passed: the read was correct (compared the output scr483_n.tst with the initial input file nuclides.dat: the data agreed).
All saved results are in the subdirectory test_results.

PL10:

Name: Tests of robustness of invent.f when data file nuclides.dat was malformed by inserting extra dot.
Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\
Path of results: \scr483\test_results\
Special input files or modifications to input file: nuclides.dat with extra dot inserted.
Program modes to be used: NumberOfRealizations =1
Description of PL10 Test Plan:
- Objectives: See if invent.f is robust when reading data from malformed file nuclides.dat with extra dot inserted.
- Assumptions: All code modification are marked in the new file invent.f by scr 483.
- Input file to test: nuclides.dat.
- Fortran file to test: invent.f
- Pass / fail criteria: The tests pass if the readout is robust to data malformed by inserting extra dot.
PL10 Test Procedure and **Results**: Data file was malformed by inserting extra dot in the middle between data values in nuclides.dat. The malformed file nuclides.dat was saved nuclides_dot.dat.
The initial program did not read the dot, error message was printed, test passed.
Also the modified program did not read the dot, error message was printed, test passed.

PL11:
Name: Tests of robustness of invent.f when reading data from file nuclides.dat malformed by shifting one number one space left.
Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\
Path of results: \scr483\test_results\
Special input files or modifications to input file: nuclides.dat
Program modes to be used: NumberOfRealizations =1
Description of PL11 Test Plan:
- Objectives: See if invent.f is robust when reading data from malformed file nuclides.dat when shifting one number one space left.
- Assumptions: All modification are marked in the new file invent.f by scr 483.
- Input file to test: nuclides.dat.
- Fortran file to test: invent.f
- Pass / fail criteria: The tests pass if the readout is robust to data malformed by shifting one number one space left.
PL11 Test Procedure and Results:
Data file was malformed by shifting one number left by one space in nuclides.dat (see file nuclides_f.dat, data record 4). The initial program (with formatted read statement, see file invent_0.f) has read the extra numbers. The read out was incorrect (see file scr483_f, record i=4), no error message was printed, but the error did not affect the read out of other records. Test failed.
The modified program (with unformatted read statement, see file invent_y.f) has read the shifted number correctly. The read out was all correct, no error message was printed. Test passed.

PL12:
Name: Tests of robustness of invent.f when reading data from file nuclides.dat malformed by by inserting extra numbers.
Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\
Path of results: \scr483\test_results\
Special input files or modifications to input file: nuclides.dat malformed by by inserting extra numbers.
Program modes to be used: NumberOfRealizations =1
Description of PL12 Test Plan:
- Objectives: See if invent.f is robust when reading data from malformed file nuclides.dat by inserting extra numbers.
- Assumptions: All modification are marked in the new file invent.f by scr 483.
- Input file to test: nuclides.dat.
- Fortran file to test: invent.f
- Pass / fail criteria: The tests pass if the readout is robust to data malformed by shifting one number one space left.

PL12 Test Procedure and Results:

Data file was malformed by inserting extra numbers between data values in nuclides.dat (see file nuclides_0.dat). The initial program (with formatted read statement, see file invent_0.f) has read the extra numbers. The read out was incorrect (see file scr483_0, record i=4), no error message was printed, but the error did not affect the read out of other records. Test failed.

Using the modified version with unformatted read (see file invent_y.f), the read out was incorrect (see file scr483_0m, record i=4), no error message was printed, and the error did not affect the read out of other records. Test failed.

There is a way to fix this problem, by reading each line as a string and counting contiguous sequences of strings or digits. This method was not implemented and the system TPA is prone to errors when nuclides.dat has extra numbers.

Data formed correctly are read without problems. Any value in a file can be incorrect, what is harder to detect than spotting an extra value. Data files must be carefully tested.

It can be seen how the minor tests for robustness failed for the initial code and also partly failed for the modified code (in one case of malformed nuclides.dat file)

All saved results are in the subdirectory test_results.

TOPIC: Fixing problems with non-robust read-out of file burnup.dat and nuclides.dat. Continuation.

If data is not expected to be in a specific order, then an index can be used with each data record, e.g. placed as the first column. Each time the data are recorded in the arrays under the index read from the file. If data are corrupted (e.g. missing), only the data existing in that record comprising error can be affected.

Index can be tested when reading data by comparing the value of the index read with the index of the loop:

```
    do i = 1,iNuclides
      read (ireadnuclidefile, 100) iNucIndex,
  &     namesisotopesave(iNucIndex), namesisotopesave2(iNucIndex),
  &     namesisotopesave3(iNucIndex), nameselemsave(iNucIndex),
  &     wmole(iNucIndex), halflife(iNucIndex), epalim(iNucIndex),
  &     cipermtuat10yr(iNucIndex), cipermtuat10yrglass(iNucIndex),
  &     idecaytype(iNucIndex)
      if (i .ne. iNucIndex) then
        print *, ' ***>>> Error reading data: <<<***'
        print *, ' Incorrect index idx =', iNucIndex, ' detected '
        stop
        end if
     enddo
100  format(i5,3(3xa6),3x,a2,3x,d7.1,3x,d7.3,3x,d7.2,2(3x,d8.2),3x,i5)
```

Using formatted vs. unformatted read statements

Malformed data does not comply with the format expected, therefore one may think that formatted statements can be used to detect malformed data. There is a trap, however with formatted read statements. When data is shifted beyond the field restricted for the value, it is simply cut off without any error message making formatted read statements a serious undetectable error-potential problem. Formatted statements do not detect malformed data.

Made a test bed by shifting one number one space left and the test failed (compare file nuclides_f.dat). Made the fix: replaced the formatted read statement by unformatted one. The modified program (with unformatted read statement, see file invent_y.f) has read the shifted number correctly, no error message was printed. Test passed.

TOPIC: Generating new file with kd and rd coefficients.

The goal is to create a file actinide_kdrd.out with kd and rd coefficients for all realizations of general use.

File nefii.inp for instance collecting rd coefficients, gets overwritten for each subarea and zone with new data.

Received the final version of the modifications of Rds from Carol.

Found location of the write of the rd coefficients modified by Carol into actinide_kdrd.out.

Realized on what finally identifies leg, layer, media and element in the program.
When computing and modifying Rds, inside the 3 nested loops, saved the layer and media in newly defined arrays leyers(INC) and medi(INC) where INC is the leg. Then printed inside 2 nested loops over the legs and elements to the actinide_kdrd.out the leyer name and Kds with each element name. The external loop is over the legs, and the nested loop is over the elements.
Tested the output by comparing contents of the file nefii.inp with the actinide_kdrd.out. Test passed.

August 20, 04

TOPIC: TEST by duplicating results of old version on modified code using old burnup and nuclide data, SCR-483. Continuation.

PL13:
NAME: Duplicate results of array amolepermtu and the results totdose.res of old version using old data files.
Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\
Path of results: \scr483\test_results\
Special input files or modifications to input files: burnup.dat and nuclides.dat.
Program modes to be used: NumberOfRealizations =1.
Description of **PL13** Test:
- Objective: See if results of array amolepermtu and the results totdose.res of old version using old data files can be duplicated on the new version test bed.
- Assumptions: All modification are marked in the new invent.f file by scr 483.
- Output file / array to test: array amolepermtu in subroutine newinventdb().
- Fortran files to test: invent.f
- Pass / fail criteria: The test passes if:
the results of array amolepermtu and the results totdose.res of old version using old data files can be duplicated on the new version test bed.
PL13 Test Procedure and Test Results:
Tested values in the array amolepermtu, output of subroutine allchain() called in subroutine newinventdb() in file invent.f.
(a) Prepared the test bed on the new version newinventdb() in file invent.f.
(b) Used old data from files burnup.dat and nuclides.dat in the new version tpa50w:
(i) Set yearofburnupdata in burnup.dat to 2008
(ii) Set yearofinventorydata to 2018
(iii) Replaced the new data by the old data, without touching anything above the names of fields of data
(c) Collected the results of array amolepermtu in the file scr483.tst of TPA50v and in the file scr483.tst of TPA50w. Recompiled the TPA50w code. Executed and compared the two files: scr483.tst of TPA50v and scr483.tst of TPA50w.
These two files were the same. Test PASSED. The array amolepermtu of old version using old data files can be duplicated on the new version.
(d) Compared the results totdose.res of old version to that of the new version. These two results totdose.res were the same. Test PASSED.
Saved the file scr483.tst as scr483_v_a.tst in TPA50v and as scr483_a.tst of TPA50w.
Saved the file invent.f of TPA50v as invent_v_a.f, and file invent.f of TPA50w as invent_a.f. Saved the results totdose.res of the old version as totdose_v_a.res and of the new version as totdose_a.res.
All saved results are in the subdirectory test_results.

PL14:
NAME: Duplicate results of array amolepermtuglass of old version using old data files.
Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\
Path of results: \scr483\test_results\
Special input files or modifications to input files: burnup.dat and nuclides.dat.

Program modes to be used: NumberOfRealizations =1.
Description of **PL14** Test:
- Objective: As in the previous test, to see if results of array amolepermtuglass of old version using old data files can be duplicated on new version test bed.
- Assumptions: All modification are marked in the new invent.f file by scr 483.
- Output file / array to test: array amolepermtuglass in subroutine newinventdb().
- Fortran files to test: invent.f
- Pass / fail criteria: The test passes if:
  the results of array amolepermtuglass of old version using old data files can be duplicated on the new version test bed.

PL14 test procedure and Test Results:
Tested array amolepermtuglass, output from another call to subroutine allchain() in subroutine newinventdb() in file invent.f.
(a) Used the test bed as above.
(b) Used old data from files burnup.dat and nuclides.dat to run the new version tpa50w with yearofburnupdata in burnup.dat set to 2008 and yearofinventorydata in nuclides.dat set to 2018.
Compared the two files: scr483.tst of TPA50v and scr483.tst of TPA50w.
These two files were the same.
Test PASSED. Results of previous version were duplicated by the new version when working on the old data files.
Saved scr483.tst as scr483_v_b.tst in TPA50v and as scr483_b.tst of TPA50w.
Saved invent.f as invent_v_b.f of TPA50v and invent.f as invent_b.f of TPA50w.
All saved results are in the subdirectory test_results.

PL15:
NAME: Duplicate results of array amolepermtuat10yr of old version using old data files.
Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\
Path of results: \scr483\test_results\
Special input files or modifications to input files: burnup.dat and nuclides.dat.
Program modes to be used: NumberOfRealizations =1.
Description of **PL15** Test Plan:
- Objective: As in the previous tests, to see if results of array amolepermtuat10yr of old version using old data files can be duplicated on new version test bed.
- Assumptions: All modification are marked in the new invent.f file by scr 483.
- Output file / array to test: array amolepermtuat10yr in subroutine newinventdb().
- Fortran files to test: invent.f
- Pass / fail criteria: The test passes if:
  the results of array amolepermtuat10yr of old version using old data files can be duplicated on the new version test bed.

PL15 Test Procedure and Test Results:
Tested array amolepermtuat10yr filled with data in subroutine newinventdb() in file invent.f:
(a) Used the test bed as above.
(b) Used old data from files burnup.dat and nuclides.dat to run the new version tpa50w as before with yearofburnupdata in burnup.dat set to 2008 and yearofinventorydata in nuclides.dat set to 2018.
(c) Compared the two files: scr483.tst of TPA50v and scr483.tst of TPA50w.
These two files were the same. Test PASSED.
Saved scr483.tst as scr483_v_c.tst of TPA50v.
Saved invent.f as invent_v_c.f of TPA50v.
Saved scr483.tst as scr483_c.tst of TPA50w
Saved invent.f as invent_c.f of TPA50w.
All saved results are in the subdirectory test_results.

PL16:
NAME: Duplicate results of array amolepermtuat10yrglass of old version using old data files.

Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\
Path of results: \scr483\test_results\
Special input files or modifications to input files: burnup.dat and nuclides.dat.
Program modes to be used: NumberOfRealizations =1.
Description of **PL16** Test Plan:
- Objective: As in the previous tests, to see if results of array amolepermtuat10yrglass of old version using old data files can be duplicated on new version test bed.
- Assumptions: All modification are marked in the new invent.f file by scr 483.
- Output file / array to test: array amolepermtuat10yrglass in subroutine newinventdb().
- Fortran files to test: invent.f
- Pass / fail criteria: The test passes if:
  the results of array amolepermtuat10yrglass of old version using old data files can be duplicated on the new version test bed.

PL16 Test Procedure and Test Results:
Tested the array amolepermtuat10yrglass filled with data in subroutine newinvent in db() file invent.f:
(a) Used the same test bed;
(b) Used old data from files burnup.dat and nuclides.dat to run the new version tpa50w as before with yearofburnupdata in burnup.dat set to 2008 and yearofinventorydata in nuclides.dat set to 2018.
(c) Compared the two files: scr483.tst of TPA50v and scr483.tst of TPA50w
These two files were the same. Test PASSED.
Saved scr483.tst as scr483_v_d.tst of TPA50v
Saved invent.f as invent_v_d.f of TPA50v.
Saved scr483.tst as scr483_d.tst of TPA50w
Saved invent.f as invent_d.f of TPA50w.
All saved results are in the subdirectory test_results.

PL17:
NAME: Duplicate results of arrays amolepermtuatemplace, cipermtuatemplace and amolepermtuatemplaceglass of old version using old data files.
Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\
Path of results: \scr483\test_results\
Special input files or modifications to input files: burnup.dat and nuclides.dat.
Program modes to be used: NumberOfRealizations =1.
Description of **PL17** Test Plan:
- Objective: As in the previous tests, to see if results of arrays amolepermtuatemplace, cipermtuatemplace and amolepermtuatemplaceglass of old version using old data files can be duplicated on new version test bed.
- Assumptions: All modification are marked in the new invent.f file by scr 483.
- Output file / array to test: arrays amolepermtuatemplace, cipermtuatemplace and amolepermtuatemplaceglass in subroutine newinventdb().
- Fortran files to test: invent.f
- Pass / fail criteria: The test passes if:
  the results of arrays amolepermtuatemplace, cipermtuatemplace and amolepermtuatemplaceglass of old version using old data files can be duplicated on the new version test bed.

PL17 Test Procedure and Test Results:
Tested / verified arrays amolepermtuatemplace, cipermtuatemplace and amolepermtuatemplaceglass. The arrays amolepermtuatemplace and amolepermtuatemplaceglass are the copies of amolepermtu and amolepermtuglass correspondingly which passed the testes already.
This way actually tested the array cipermtuatemplace in subroutine newinventdb() in file invent.f.
(a) Used test bed as before;.
(b) Used old data from files burnup.dat and nuclides.dat to run tpa50w as before with yearofburnupdata in burnup.dat set to 2008 and yearofinventorydata in nuclides.dat set to 2018.

(c) Compared the two files: scr483.tst of TPA50v and scr483.tst of TPA50w.
These two files were the same. Test PASSED.
Saved scr483.tst as scr483_v_e.tst of TPA50v
Saved invent.f as invent_v_e.f of TPA50v.
Saved scr483.tst as scr483_e.tst of TPA50w
Saved invent.f as invent_e.f of TPA50w.
All saved results are in the subdirectory test_results.

PL18:
NAME: Duplicate results of function qpermtu() of old version using old data files.
Path of run directory of initial code: \scr483\tpa50v\
Path of run directory of modified code: \scr483\tpa50w\
Path of results: \scr483\test_results\
Special input files or modifications to input files: burnup.dat and nuclides.dat.
Program modes to be used: NumberOfRealizations =1.
Description of **PL18** Test Plan**:**
- Objective: As in the previous tests, to see if results of function qpermtu() of old version using old data files can be duplicated on new version test bed.
- Assumptions: All modification are marked in the new invent.f file by scr 483.
- Output file / array to test: function qpermtu() in subroutine newinventdb().
- Fortran files to test: invent.f
- Pass / fail criteria: The test passes if:
the results of function qpermtu() of old version using old data files can be duplicated on the new version test bed.
PL18 Test Procedure and Test Results**:**
Tested output of function qwp=qpermtu() returning thermal output of blended HLW in nfenv.f:
(a) Used the test bed as before.
(b) Used old data from files burnup.dat and nuclides.dat to run tpa50w as before with
yearofburnupdata in burnup.dat set to 2008 and yearofinventorydata in nuclides.dat set to 2018.
Placed the write to a file 'ebs_tst.txt' of qwp when calling qwp=qpermtu() in nfenv.f both in old version tpa50v and in new version tpa50w.
Compiled and executed tpa.exe, then saved results in nfenv_tst.txt file, both in tpa50v and in new tpa50w.
(c) Compared these two outcome data files after 1st realization- they were the same.
Test PASSED.
Saved ebs_tst.txt as ebs_tst_v.txt of tpa50v.
Saved ebs_tst.txt as ebs_tst.txt of tpa50w.
Saved nfenv.f as nfenv_v.f of tpa50v.
Saved nfenv.f as nfenv.f of new tpa50w.
All saved results are in the subdirectory test_results.

TOPIC: Tracing the first time drip shield corrosion failure and mechanical failure.

It is expected that subroutine mechfail() in seismo2.f uses the fluoride concentration information coming from subroutine nfenvFl() and computes the first time corrosion failure or mechanical failure. This drip shield failure time then is used by the subroutine ebsfail().
Traced, it this happens after the modifications, i.e. after creating the new subroutine nfenvFl(), re-writing nfenv(), call to new nfenvFl() and re-arranging call to nfenv() in exec().
For the testing, set DripShieldCorrosionRate[m/yr] normal distribution to 3.5e-6, 4.3e-6 (from 8.7e-8, 4.3e-6), and DripShieldThickness[m] to 0.011 (from 0.015) in tpa.inp.
1. Checked if mechfail() is called after call to new nfenvFl() and before call to nfenv().
Made prints to the screen from seismo() in seismo2.f calling stand-alone code with mechfail().
In fact, seismo() calling mechfail() is executing after nfenvFl() and before nfenv() is called. Test passed.
2. Noticed that a message is printed to the screen in seismo() for subarea 4 of '<<warning>> drip shield corrosive failure: 2.75e3 [yr] <occurred prior to mechanical failure>'.

Traced seismo2.f to see why. This message is printed in SEISMO() when indicatorCorrosion=1. The indicatorCorrosion value is read from file mechfail.dat in SEISMO(). Stand alone code mechfail() writes to mechfail.dat.
Traced mechfail() to see, why. The indicatorCorrosion is set to 1 in mechfail() when corrosive_flag=true. Noticed, that not only ds_thickness>0.0 at time 0.0, but corrosive_flag = False up to time period 147 when it is set to True in mechfail.f in subroutine processMechanicalFailure() in the call to subroutine processElements(). Traced and learned that corrosive_flag = False as long as ds_thickness>0. Made a test bed to see that the corrosive_flag turns to True when ds_thickness turns to 0 at time period 147. Test passed. Printed the test results to file tst.dat in the run directory from mechfail(). Copied tst.dat to tst_f.dat and mechfail() with test bed to mechfail_f().
The corrosive_flag = False can be set to True only when dsFailureFlag=False, because only then the corrosion failure is prior to mechanical failure. This way corrosive_flag has the meaning: 'corrosion failure occurred prior to mechanical failure'. Figured out that the dsFailureFlag is set to True when ds_thickness()<=0.0 or as a result of mechanical events.

The array ds_thickness() is read from file 'mechfail.inp' in subroutine getRealArray(inFile, numTimes,ds_thickness, ...) called in subroutine processMechanicalFailure(..inFile, . . .) called in stand-alone code mechfail() in mechfail.f. The getRealArray() retrieves numTimes values of ds_thickness() from that file mechfail.inp.

Traced further and figured out that subroutine SEISMO(. ., dsThickness, ...) in exec.f. (see file seismo2.f) writes dsThickness data to that mechfail.inp file by call to setRealArray(outFile, numTime, dsThickness, ...) in subroutine buildMechanicalInputFile(..., dsThickness, ...).

To get values dsThickness(), the function getThickness() reads the dsthickness() from dsfailt.dat generated by dsFailt(). The function getThickness() is called in dsfail() after calling dsFailt(). The dsfail() is called by exec().

The stand-alone dsFailt() is called by dsfail() in exec() before getThickness() is called . The file name dsfailt.dat is read by subroutine seupCommons() in dsFailt() from line 5 of file dsfailt.def. The dsfailt.dat is the file keeping values of ds thickness vs. time. The line 4 of file dsfailt.def keeps the name fluoride.dat of file of fluoride concentrations vs. time based on which dsfailt.dat is computed. Then dsFailt() calls subroutine getDSThickness() The getDSThickness() takes fluoride.dat and computes dsfailt.dat with ds thickness vs. time. The ds thickness is computed as the difference between initial dripShieldThickness and penetration_depth by floride. Subroutine dsfailtime() in dsFailt() computes the penetration_depth using the fluoride concentrations vs. time.

After finding out that the links between the ds thickness and the fluoride were not broken, the question is now, whether the right values of concentrations exist in the file fluoride.dat after all the modifications. Traced further on how the file fluoride.dat is made. The time values vs. fluoride concentrations are written to file 'fluoride.dat' in dsfail() in subroutine buildInputFiles(). The array fluorideConcentration() keeping fluoride concentrations is the input parameter to dsfail(), and the dripShieldThickness() is the output parameter of dsfail(). The call to dsfail() is with the parameter flds() which is the array of fluoride concentrations. This call is made after the fluoride concentrations flds() are set in nfenvFl().

This way tested the sequence of calls in exec() after the modifications from the point of view of the ds corrosion by fluoride:
nfenvFl(), then dsfail() calling dsfailt(), then seismo(), then nfenv(), then ebsfail()
or
nfenvFl(), then dsfail() calling dsfailt(), then nfenv(), then seismo(), then ebsfail()
and finding out that both are correct.
The nfenvFl() sets fluoride concentrations, which are written to file fluoride.dat by dsfail(). The stand-alone code dsfailt() takes the file fluoride.dat and computes the ds thickness giving array dripShieldThickness() as output parameter. Corrosion failure is then computed based on the ds thickness as traced above. This drip shield corrosion / mechanical failure time then is available for the subroutine ebsfail() as needed.

August 27, 04

TOPIC: TEST by duplicating results of old version on modified code using old burnup and nuclide data, SCR-483. Continuation.

Completion of the topic: Splitting complicated tests into separate tests for clarity.
Making sure that the results exist for the tests made.

TOPIC: Add parameter RefluxEndTime[yr] to nfenv.f to control the end of reflux. SCR-516

Looked for key-word 'reflux' in the file nfenv.f. Read comments and code statements to find the places of the code related to the topic in question. Found closest places in the subroutines reflux2() and reflux3() in nfenv.f. Studied the role of the variable refluxend in the subroutines reflux2() and reflux3(). It gets the value 1.0e4 by hard-coding in these two subroutines reflux2() and reflux3() and there is no any other assignment of value to it. Figured out that reflux2() and reflux3() are called only in subroutine nfenvFl(). This way found the place for the input of the RefluxEndTime[yr]: in the nfenvFl() in nfenv.f file. Inserted statements to read refluxend in nfenvFl():

```
      call clearchar( 60, name )
      name = 'RefluxEndTime[yr]'
      irefluxend= ispquery( name )
      refluxend = valuesp( irefluxend )
```

Found the right location near the constant MaximumTime[yr] and added RefluxEndTime[yr] in the tpa.inp:

```
constant
RefluxEndTime[yr]
1.0e4
```

Implemented the read-out of refluxend by computing the index irefluxend of RefluxEndTime[yr], inside the ikey IF block, and then executed the valuesp() for this index also inside the ikey IF block. This way this parameter will be read only at the first realization, this parameter RefluxEndTime[yr] will be modifiable by the user in the file tpa.inp. Declared the two parameters in the nfenvFl()

```
      real refluxend
      integer irefluxend
```

Removed the assignment of the value 1.0e4 to refluxend in these two subroutines reflux2() and reflux3(). Instead, passing refluxend to the subroutines reflux2() and reflux3() by the common block. Declared this common block in the nfenvFl() and in these two subroutines reflux2() and reflux3():

```
      common / ref_com / refluxend
```

Tested the modification by inserting the print statements: in nfenvFl, reflux2(), and reflux3(), printing the value refluxend that is read through the parameter name RefluxEndTime[yr] from tpa.inp. Test passed.

TOPIC: Add data checks for burnup.dat and nuclides.dat. SCR-516

Implemented unformatted Fortran read statement to input nuclides.dat, because data numbers can be shifted left or right when editing file nuclides.dat. Any string or value outside of the field restricted by the format is cut off. This may cause errors in data inputted, and no error nor warning is printed. This may happen when data is edited by user. Unformatted statement fixes this problem, however, when user inserts by mistake an extra value, that unexpected data item is inputted as the right one, other values in the same record are shifted one column to the right. No error message is printed, and the values inputted are incorrect.

Devised, developed and implemented a module counting continuous strings in each record to remedy this problem. If the count of continuous strings is different from the number of continuous strings in the first record, then error message is printed and the execution is stopped. In the code module below, sCt is the number of continuous strings counted. The sCt is incremented by 1 when the beginning of a string is detected when scanning the line ae. This happens when the previous character was blank and the current character is not blank. The previous not a blank character is flagged by the logical st. The current character is scanned by ae(n:n) in the loop. The st = .true. and ae(n:n)<>' ' increments the sCt by 1. The module checks if the final value of sCt at each record of data is different from ct, where ct is the count of continuous strings in the first data record when i=1.

```
c     Count the number of continuous strings:
      read (ireadnuclidefile, '(a)') ae
      lh=len(ae)
      st=.false.          !Flag of string = not blank
      sCt=0
czw       if (i .eq. 4) print *,ae
      do 21 n=1,lh-1
        if (ae(n:n) .ne. ' ') then
          if( .not. st) then
```

```
          sCt=sCt+1
czw           if(i.eq.4)print *,ae(n:n)
          end if
         st=.true.
        else
         st=.false.
        endif
21      continue
czw      if (i .eq. 4) print *,'i=',i, ' sCt=',sCt, ' ct=',ct
     if (i .eq. 1) then
       ct=sCt
       else
        if (sCt .ne. ct) then
         print *, ' **>> Error in newinventdb <<**'
         print *, '     Error in nuclides.dat '
         print *, ' Incorrect number of data items =', sCt,
   &            ' detected in data record ', i
         stop
         endif
       end if
     backspace(ireadnuclidefile)
```

Tested the code module by inserting two print statements marked with czw. Record number i=4 was selected for this test because for testing purposes the file nuclides.dat was modified by adding a few extra numbers in record 4. The statement:

```
if (i .eq. 4) print *,ae
```

prints this line of data to see where the beginning of each string was detected.
The beginning of each string detected is printed by the statement:

```
if(i.eq.4)print *,ae(n:n)
```

The test passed: the module counts the right beginnings of continuous strings.
The module prints the error message only when the number of continuous strings in subsequent lines is not consistent.

TOPIC: Integrating data checks for burnup.dat and nuclides.dat on versions TPA50w and TPA50x, SCR-516.

1. The data checks for burnup.dat and nuclides.dat were added on the version TPA50w, and were not installed yet on the TPA50x (TPA50x comprises the new near field environment). The integration took place as follow:
(i) Replaced the old invent.f in TPA50x with modified invent.f from TPA50w;
(ii) Replaced the old inventi.i in TPA50x with modified inventi.i from TPA50w (comprising new parameters, e.g. yearofburnupdata, yearofemplacelent);
(iii) Commented out the call to setage() in exec.f in TPA50x:

```
cc rwj 7-24-04; SCR483
cc    Replace AgeOfWaste with CalendarYearOfEmplacement
cc prepare INVENT database
cc     call clearchar( 60, name )
cc     name = 'AgeOfWaste[yr]'
cc     ivalue = ispquery( name )
cc     age = valuesp(ivalue)
cc     call setage( age )
```

(iv) Inserted new parameter in tpa.inp of TPA50x:

```
** rwj 7-24-04; scr483
** This parameter must appear before the aqueousnuclides keyword,
** since aqueousnuclides invokes subroutine newinventdb() which
```

** uses the year of emplacement.
constant
CalendarYearOfEmplacement[A.D.]
2034.
(v) Replaced old files burnup.dat and nuclides.dat in TPA50x with modified burnup.dat and nuclides.dat from TPA50w.
Successfully compiled and executed.

TOPIC: Integrating new parameter RefluxEndTime[yr] on versions TPA50w and TPA50x, SCR-516.

(i) Added new
constant
RefluxEndTime[yr]
1.0e4
in tpa.inp of TPA50x, the same way as on the TPA50w;
(ii) Duplicated changes made in nfenvFl() of TPA50x associated with new RefluxEndTime[yr]
on nfenv() in TPA50w, the same way as on the version TPA50x.
Successfully compiled and executed.

September 3, 2004

TOPIC: Checking TPA system for potential problems with inputting malformed *.dat files.

Started to check all *.dat files to see if there may exist any potential problem similar to those with inputting data from files nuclides.dat and burnup.dat (as described above). Listed all *.dat files and looked among them for these files which were updated most recently, as most frequently updated. Found among them files updated in 2003: multifbe.dat, multifaf.dat, ia.dat, coefkdeq.dat, coefkdeqr.dat,. Used grep on UNIX to see the *.f Fortran files and statements reading data from these data files. Statements with names multifbe.dat, multifaf.dat were all commented out even though these files multifbe.dat, multifaf.dat were still in the system. Did not see name coefkdeqr.dat, in any *.file, it could be specified in a .def file from where the name coefkdeqr.dat could be inputted.

1. Examined inputting file ia.dat in iareader.f. Data from ia.dat is inputted by reading the whole line by using:
```
    read (iaunit,'(a80)', END=300) cline
```
and then this string *cline* is searched for'=' sign and data following it.
There are no potential danger to read data with the keyword '=', because it is obligatory, what is tested by:
```
    ipos = index(cline,'=')
    if (ipos .eq. 0) then
cc     Equal sign not present.
      print *, ''
      print *, ' ***>>> Error in file ia.dat <<<*** '
      print *, 'Unrecognized input at line number ', ilinenum
      stop
    end if
```
Also, any key-word checked, such as'VALUE' or 'Value' is converted to upper-case by function ucljsg() and will not be miss-interpreted. Inputting file ia.dat is safe.

2. Started to examine inputting file drythick.dat in nfenv.f.

TOPIC: Testing temperature of repository in ebsrel.ech to be the same in versions tpa50v and tpa50w.

A. Set OutputMode(...) to 1 in tpa.inp in both versions tpa50v and tpa50w.
Then executed both versions tpa50v and tpa50w and tested if the column temprep in ebsrel.ech is identical. The column temprep was identical in both versions tpa50v and tpa50w. Test passed.

TOPIC: Testing temperature of repository in ebsrel.ech to be very similar in versions tpa50x and tpa50w in the first 50 years. Tested and found the temperature of repository too high in the first 50 years.
Started to test, why.

Figured out that the data gets written to ebsrel.ech in exec() from array temprep() in a loop over time steps. This array temprep() is outputted from subroutine nfenvFl().
Tested to see if there are updates of temprep() in nfenv(), in addition to updates in nfenvFl(). None found, test passed.
Found the first assignment of temperature value to temprep(), it is by call to procedure cond3dxyzt().
Compared the temperatures just after the call to cond3dxyzt() in tpa50v and tpa50w. They were different - test failed. Traced input values to cond3dxyzt(). The value cond was different:
cond~1.40 in tpa50_nf, cond~1.51 in tpa50v. Then got order to suspend work because of lack of funds.

GEOSCIENCES AND ENGINEERING DIVISION

SCIENTIFIC NOTEBOOK REVIEW CHECKLIST RECORD

Scientific Notebook No.: 612-12E Project Numbers: _____

Accomplished

JRW	1. Initial entries per QAP-001
JRW	2. Dating of entries
JRW	3. Corrections (crossed out, one line through w/initials/date) N/A
JRW	4. No White out used
JRW	5. Page number visible on copy or original notebook
JRW	6. In process entries per QAP-001
JRW	7. Figure information present Note – QAP-001 provides no guidance on Figure information
JRW	8. Text readable
JRW	9. Copyrighted material is identified N/A
JRW	10. Permanent ink or type only
JRW	11. Signing of entries (not required on each page) Entries not signed, but all entries were made by Zbigniew Wojcik as indicated on headers for each page.
JRW	12. Electronic media in the scientific notebook properly labeled N/A
JRW	13. NRC Supplementary Scientific Notebook Questions are addressed. see Attached pg 41

Any discrepancies must be resolved before notebook closeout.

==========

I have reviewed this scientific notebook and find it in agreement with QAP-001.

JR Walker 3/14/07

Manager's Signature Date

Attach this completed form to the last page of the notebook.

Form QAP-01 (10/2006)