MOL.19980918.0004

CRWMS/M&O

Calculation Cover Sheet

Complete only applicable items.

1. QA: N/A
Page: 1 Of: ~~77~~ 47
~~77~~ 47 9-15-98

2. Calculation Title
EQ6 Calculations for Chemical Degradation of Pu-Ceramic Waste Packages

3. Document Identifier (including Revision Number)
BBA000000-01717-0210-00018 REV 00

4. Total Pages
~~77~~ 47 9-15-98

5. Total Attachments
2

6. Attachment Numbers - Number of pages in each
I-27, II-3

	Print Name	Signature	Date
7. Originator	H.W. Stockman	Harlan W. Stockman	9-10-98
8. Checker	P.L. Cloke	Jan E Clare for P.L. Cloke	9/10/98
9. Lead Design Engineer	Peter Gottlieb	Jan E Clare for Peter Gottlieb	9/10/98

10. Remarks
Editorial changes: "Of: 77" in block 1 of cover sheet corrected to 47.
"77" in block 4 of cover sheet corrected to 47.
— Harlan W. Stockman 9-15-98.
Peter Gottlieb 9/15/98

Revision History

11. Revision No.	12. Date Approved	13. Description of Revision

Table of Contents

List of Tables

Figures

Item Page

1. Purpose

In this study, the long-term geochemical behavior of a waste package (WP), containing Pu-ceramic, was modeled. The ceramic under consideration contains Ti, U, Pu, Gd and Hf in a pyrochlore structure; the Gd and Hf stabilize the mineral structure, but are also intended to provide criticality control. The specific study objectives were to determine:

1) the extent to which criticality control material, suggested for this WP design, will remain in the WP after corrosion/dissolution of the initial package configuration (such that it can be effective in preventing criticality), and

2) the extent to which fissile plutonium and uranium will be carried out of the degraded WP by infiltrating water (such that internal criticality is no longer possible, but the possibility of external criticality may be enhanced), and

3) the nominal chemical composition for the criticality evaluations of the WP design, and to suggest the range of parametric variations for additional evaluations.

For this purpose, the chemical compositions (and consequent criticality evaluations) are modeled for time periods up to 10^6 years. This longer time frame is consistent with the one million years time horizon recently recommended by the National Academy of Sciences to the Environmental Protection Agency for performance assessment related to a nuclear repository (Ref. 1).

The calculation included elements with high neutron-absorption cross-sections, notably Gd and Hf, as well as the fissile materials. In the calculations, the thermodynamic/geochemical behavior of Hf was represented by Zr; this substitution was necessary, due to the lack of thermodynamic data for Hf, and is justified by the extreme chemical similarity of the two elements (Assumption 3.16). The results of this analysis will be used to ensure that the type and amount of criticality control material used in the WP design will prevent criticality.

2. Method

The method used for this analysis involves the following steps:

- Use of basic EQ3/6 (software package, Section 4.1) capability for tracing the progress of reactions with evolution of the chemistry, including the estimation of the concentrations remaining in solution and the composition of the precipitated solids. (EQ3 is used to determine a starting fluid composition for EQ6 calculations; it does not simulate reaction progress.)

- Evaluation of available data on the range of dissolution rates for the materials involved, to be used as material/species input for each time step.

- Use of "pseudo flow-through" mode in which:

    1) Water is added continuously to the WP and builds up in the WP over a sequence of time steps (typically 15 to 18 steps per sequence, except for the initial sequence). The first sequence typically ranges from 200 to 600 steps. The duration of a time step modeled for the individual EQ6 time steps range from 0.01 seconds to 1000 days as determined automatically by the program. The modeled duration of a sequence, including the initial sequence, stays constant within the limits imposed internally by the program. This time is determined from the selected drip rate, e.g., 0.15 m^3/yr entering the WP, and the percentage of added water selected. This percentage is set at 10% at the beginning of a set of runs, and may be increased to 100% to enable modeling of very long times after initial relatively rapid chemical changes have settled down to a quasi-steady state.

    2) Flushing action (removal of water added during one EQ6 sequence) is simulated by specifying smaller amounts of water and solutes for input to the next EQ6 sequence than were present at the end of the preceding sequence. The mass of water simulated as removed equals the mass of water added, adjusted for water calculated to enter or exit solids. Solutes are removed in proportion to their concentrations in that mass of water.

- Determination of fissile concentrations in solution as a function of time (from the output of EQ6 sequences over times up to 10^6 years).

- Calculation of the amount of fissile material released from the WP as a function of time (which thereby reduces the chance of criticality within the WP).

- Determination of concentrations of neutron absorbers, such as Gd, Hf and B, in solution as a function of time (from the output of EQ6 sequences over times up to or somewhat greater than 10^6 years).

- Calculation of the amount of neutron absorbers retained within the WP as a function of time.

- Composition and amounts of solids (precipitated minerals or corrosion products, and unreacted package materials).

Further detail on the specific methods employed for each step is available in Section 5 of this set of calculations.

3. Assumptions

All assumptions are for preliminary design; these assumptions will require verification before this analysis can be used to support procurement, fabrication, or construction activities. All assumptions are used throughout the calculations.

3.1 It is assumed that an aqueous solution fills all voids within WPs, and that the solutions that drip into the package will have the composition of J-13 well water (as given in Ref. 2; this composition is given in Table 5.1.1.2-1) for ~10^6 years. The basis for the first part of this assumption is that it provides the maximum degradation rate with the potential for the fastest flushing of the neutron absorber from the WP, and is thereby conservative. The basis for the second part of the assumption is that the groundwater composition is controlled largely by transport through the host rock, over pathways of hundreds of meters, and the host rock composition is not expected to change substantially over 10^6 years. For a few thousand years after waste emplacement the composition may differ because of perturbations resulting from reactions with engineered materials and from the thermal pulse. These are not taken into account in this calculation because the corrosion allowance and corrosion resistant barriers are not expected to breach until after that perturbed period. Therefore, the early perturbation is not relevant to the calculations reported in this document. See assumption 3.3.

3.2 It is assumed that the density of J-13 well water is 1.0 g/cm^3. The basis is that for dilute solutions, the density differs extremely little from that for pure water and that any differences are insignificant in respect to other uncertainties in the data and calculations. Moreover, this number is used only initially in EQ3/6 to convert concentrations of dissolved substances from parts per million to molalities.

3.3 It is assumed that (1) water infiltrating the WP will have only a minimal contact, if any at all, with undegraded metal in the corrosion allowance barrier, and (2) that any effects of contact with the drift liner will be minimal after a few thousand years. The basis for the first part of this assumption is that the water should move rapidly enough through openings in the WP barriers that its residence time in the corroded barrier will be too small for significant reaction to occur. Furthermore, the water flowing through the barriers will be in contact with the corrosion products left from the barrier corrosion that created the holes in the first place, but these corrosion products will closely resemble iron oxides and hydroxides in the overlying rock. Consequently, the water should already be close to equilibrium with these compounds and would be unaffected by further contact with them, even if it flowed slowly enough to permit significant reaction. The second part of this assumption is justified by the following: (1) The drift liner at the top of the drift is expected to collapse with the roof support well before 1000 years, and (2) The travel time of water through the liner will be much less than the travel time through the rock above the repository, and the liner will likely have pre-equilibrated with the ambient atmosphere, forming calcite in the exposed cement.

3.4 It is assumed water may circulate freely enough in the partially degraded WP that all degraded solid products may react with each other through the aqueous solution medium. The basis is that this provides one bound for the extent of chemical interactions within the WP and conservatively simulates potential preferential loss of neutron absorbers from the WP by facilitating contact of any acid that may result from corrosion of steel with neutron absorbers in spent fuel. Such circulation is expected, driven by buoyancy and thermal gradient (Assumption 3.11).

3.5 It is assumed that the database supplied with the EQ3/6 computer package is sufficiently accurate for the purposes of this report. The basis is that the data have been carefully scrutinized by many experts over the course of several decades and carefully selected by Lawrence Livermore National Laboratory (LLNL) for incorporation into the data base (Ref. 3; Ref. 4; Ref. 5; Ref. 6). These databases are periodically updated and/or new databases added, such as one including extensive data on the lanthanides (Ref. 7). Every run of either EQ3 or EQ6 documents automatically which database is used. The databases include references internally for the sources of the data. The reader is referred to this documentation, included in electronic files labeled data0 that accompany this report, for details (Ref. 16). Nevertheless, this review and documentation do not absolutely guarantee that all the data are adequate.

3.6 In general it is assumed that chromium and molybdenum will oxidize fully to chromate (or dichromate) and molybdate, respectively. This assumption is based on the available thermodynamic data, which indicate that in the presence of air the chromium and molybdenum would both oxidize to the VI valence state. Laboratory observation of the corrosion of Cr and Mo containing steels and alloys, however, indicates that any such oxidation would be extremely slow. In fact, oxidation to the VI state may not occur at a significant rate in respect to the time frame of interest, or there may exist stable $Cr^{(III)}$ solids (not present in the EQ3/6 thermodynamic database) that substantially lower aqueous Cr concentration. For the present analyses, the assumption is made that over the times of concern the oxidation will occur. This is conservative for times of several thousand years after WP breach, when the high pH solution from any drift liner effects, has been flushed out of the WP. Acidification of the water will enhance solubility and transport of neutron absorber out of the WP thereby separating it preferentially from fissile material.

3.7 It is assumed that the inner corrosion resistant barrier of the WP will react so slowly with the infiltrating water (and water ponded in the WP) as to have negligible effect on the chemistry. The bases consist of the facts that this metal (Alloy 22; see nomenclature in section 5.1.1) corrodes very slowly compared (1) to other reactions in the WP and (2) to the rate at which soluble corrosion products will likely be flushed from the package.

3.8 It is assumed that gases in the solution in the WP will remain in equilibrium with the ambient atmosphere outside the WP. In other words, it is assumed that there is sufficient contact with the gas phase in the repository to maintain equilibrium with the CO_2 and O_2 present, whether or not this be the normal atmosphere in open air or rock gas that seeps

out of the adjacent tuff. Under these conditions the partial pressure of CO_2 exerts important controls on the pH and carbonate concentration in the solution and hence on the solubility of uranium, gadolinium and other elements. As discussed in Ref. 8, the measured composition of J-13 water is not in equilibrium with the partial pressure of CO_2 in the atmosphere. By adjusting the average measured composition of the water slightly, well within the standard deviation of the measurements, it is possible to determine a partial pressure of CO_2 nearly ten times atmospheric (Ref. 9, Table 8; and Ref. 10, p. F-210), with which this water was apparently in equilibrium at depth in the well. This high partial pressure is close to the maximum found by measurement of the rock gas composition (Ref. 9, Table 8). Therefore this high partial pressure was chosen for most of the computer runs used in this analysis. Two runs with normal atmospheric CO_2 levels was used to determine the sensitivity of the calculations to this parameter. The high CO_2 tends to increase the concentration of free carbonate ion and its complexation with the dissolved $U^{(VI)}$, thereby tending to increase the solubility of U, but this is moderated by the reduction of the pH. There is little overall net effect on actinide solubility for otherwise comparable conditions. The effect on Gd solubility is somewhat more complex; higher CO_2 pressures decrease pH, thereby increasing the chance that solid $GdOHCO_3$ will be dissolved; however, higher CO_2 pressures also increase the capacity of the system to buffer toward intermediate pH (~7.7).

3.9 It is assumed that precipitated solids that are deposited remain in place, and are not mechanically eroded or entrained as colloids in the advected water. The basis for this assumption is that it conservatively maximizes the size of potential deposits of fissile material inside the WP.

3.10 It is assumed that the corrosion rates will not be significantly enhanced by biologically mediated corrosion. The bases for this assumption are that even at the time that the repository is closed there will be little organic material present to serve as raw materials for growth of living organisms, and that by the time the corrosion barriers are breached essentially all of such material will most likely have decayed to carbon dioxide and dissipated.

3.11 It is assumed that sufficient decay heat is retained within the WP over times of interest to cause convective circulation and mixing of the water inside the package. The analysis that serves as the basis for this assumption is discussed in Ref. 11 (Attachment VI).

3.12 It is assumed that the alkalinity reported in analyses of J-13 water corresponds to bicarbonate (HCO_3^-) alkalinity. Contributors to alkalinity in J-13 water, in addition to bicarbonate, potentially include borate, phosphate and silicate. However, at pH less than 9 the contribution of silicate will be small, and in any case the concentrations of all three of these components in J-13 water is small. Fluoride ion will not contribute to a typical measured alkalinity because the titration will not be carried out to a sufficiently low pH for its influence to be detectable. Nitrate will likewise not contribute. The validity of this assumption is justified by the observation that the calculated electrical neutrality, using

the assumption, is zero within the analytical uncertainty, as it should be. The same assumption is implicitly made in Ref. 2 (Table 4.1, p. 4.2).

3.14 It is assumed that the rate of entry of water into, as well as the rate of egress from, a WP is equal to the rate at which water drips onto the package. For most of the time frame of interest, i.e. long after the corrosion barriers become largely degraded, it is more reasonable to assume that all or most of the drip will enter the degraded package than to assume that a significant portion will instead be diverted around the remains. Diversion of the water with a consequent lower entry rate has not been incorporated into the present calculations.

3.15 It is assumed that the most insoluble solids for a fissile radionuclide will form, i.e. that equilibrium will be reached. This is conservative for internal criticality because the assumption will lead to simulation for maximal retention of fissile material within the WP.

3.16 It is assumed that the thermodynamic behavior of Hf can be modeled as if it were Zr. This assumption is based on the extreme similarity of the chemical behaviors of the two elements (Ref. 12). Thermodynamic data for many important Hf solids and aqueous species are lacking, thus Zr was substituted for Hf in the calculations.

3.17 It is assumed that the decay of ^{239}Pu to ^{235}U can be conservatively modeled by a simple exponential correction to the reported amounts of solids in the EQ6 runs, after completion of each run. EQ6 currently has no built-in capability to handle radioactive decay. For internal criticality, this assumption is conservative, since Pu solids are generally less soluble than U solids; the assumption causes an overestimate of the amount of ^{235}U remaining in the package with time.

4. Use of Computer Software

This section describes the computer software used to carry out the analysis.

4.1 EQ3/6 Software Package

The EQ3/6 software package originated in the mid-1970s at Northwestern University (Ref. 3). Since 1978 Lawrence Livermore National Laboratory has been responsible for its maintenance. It has most recently been maintained under the sponsorship of the Civilian Radioactive Waste Management Program of the U.S. Department of Energy. The major components of the EQ3/6 package include: EQ3NR, a speciation-solubility code; EQ6, a reaction path code which models water/rock interaction or fluid mixing in either a pure reaction progress mode or a time mode; EQPT, a data file preprocessor; EQLIB, a supporting software library; and several (>5) supporting thermodynamic data files. The software deals with the concepts of the thermodynamic equilibrium, thermodynamic disequilibrium and reaction kinetics. The supporting data files contain both standard state and activity coefficient-related data. Most of the

data files support the use of the Davies or B-dot equations for the activity coefficients; two others support the use of Pitzer's equations. The temperature range of the thermodynamic data on the data files varies from 25 °C only for some species to a full range of 0-300 °C for others. EQPT takes a formatted data file (a data0 file) and writes an unformatted near-equivalent called a data1 file, which is actually the form read by EQ3NR and EQ6. EQ3NR is useful for analyzing groundwater chemistry data, calculating solubility limits and determining whether certain reactions are in states of partial equilibrium or disequilibrium. EQ3NR is also required to initialize an EQ6 calculation.

EQ6 models the consequences of reacting an aqueous solution with a set of reactants which react irreversibly. It can also model fluid mixing and the consequences of changes in temperature. This code operates both in a pure reaction progress frame and in a time frame. In a time frame calculation, the user specifies rate laws for the progress of the irreversible reactions. Otherwise, only relative rates are specified. EQ3NR and EQ6 use a hybrid Newton-Raphson technique to make thermodynamic calculations. This is supported by a set of algorithms which create and optimize starting values. EQ6 uses an ordinary differential equation (ODE) integration algorithm to solve rate equations in time mode. The codes in the EQ3/6 package are written in FORTRAN 77 and have been developed to run under the UNIX operating system on computers ranging from workstations to supercomputers. Further information on the codes of the EQ3/6 package is provided in Refs. 3, 4, 5 and 6.

In this study EQ3/6 was used to provide:

1) a general overview of the nature of chemical reactions to be expected,

2) the degradation products likely to result from corrosion of the waste forms and canisters, and

3) an indication of the minerals, and their amounts, likely to precipitate within the WP.

The programs have not been used outside the range of parameters for which they have been verified. The EQ3/6 calculations reported in this document used version 7.2b of the code, which is appropriate for the application, and were executed on Pentium series (including "Pentium II") personal computers (PCs).

The EQ3/6 package has been verified by its present custodian, Lawrence Livermore National Laboratory. The source codes were obtained from Software Configuration Management in accordance with M&O QAP-SI-3 (Ref. 13). The code was installed on the Pentium PCs according to an M&O-approved Installation and Test procedure (Ref. 14).

4.2 Software Routines for Chaining Successive EQ6 Cases

The following software routines were developed specifically for this study for the purpose of facilitating the setup and execution of successive cases of EQ6, by transforming the output of one case to the input of the following case. An individual EQ6 run diluted the solution

constituents to reflect the inflow of fresh water, and the routines periodically remove water and solutes corresponding to the inflow. The routines also read the output of one run and reformat it as input for the next run. The data reformatting aspect of these routines was verified by visual inspection in accordance with QAP-SI-0, 5.3.2C by an individual independent of the person doing the original development. The mathematical algorithms for these routines are given in attachment I; the checking of the routines is discussed in Ref. 17. The routines were originally developed for a Hewlett-Packard HP6000 computer (UNIX operating system), and were subsequently modified for use on Pentium personal computers. Both versions have been checked. The CSCI numbers apply to both the HP and the corresponding PC versions.

4.2.1 File nxti_bat.c, CSCI# 30046 V1.1

This routine is derived from the UNIX shell script nxtinput.bat (CSCI# 30046 V1.0), used in previous geochemical analyses (Ref. 15). To obtain the same functionality as the UNIX script, the batch file was replaced by an executable, that uses the spawn() function to run other executable programs in synchronous mode. The program nxti_bat:

1) runs the program nxtinput.exe (compiled from nxtinput.c), to transfer the output from one iteration to the input of the next iteration,

2) runs the next iteration of EQ6, and

3) repeats steps 1 and 2 until a specified number of iterations have been reached, or until an abnormal condition occurs (which causes nxtinput.exe to write an error message to a file which is read and interpreted by nxti_bat.exe).

Nxti_bat.exe requires that the input file be copied to a file named bldinput.out before invocation, and also requires the file bldinput.in be properly formatted with the root file name and the desired simulation time (deltimemax) for each EQ6 run.

4.2.2 File nxtinput.c, CSCI# 30047 V1.0

This C program reads the output and pickup (program file names) files of an EQ3/6 iteration and generates the input file for the next iteration. In this process it makes two basic data changes:

1) the amounts of all the species in solution are reduced to simulate the flushing out of an amount of solution corresponding to an infusion of fresh J-13 water into the WP as calculated by EQ6, and

2) some alternative species are switched into, or out of, the basis set for the chemical reactions, according to which member of the alternative set has achieved the largest concentration.

4.2.3 File Alln_bat.c, CSCI# 30050 V1.1

This C program is derived from the UNIX script allpost.bat (CSCI# 30050 V1.0) and operates in essentially the same way as nxti_bat.c, but in addition runs the C program newpost.c (compiled to newpost.exe) and deletes the allout files produced by these programs after the desired data have been extracted. This deletion avoids complete filling of available file space. Compared to the UNIX version, some slight functionality is added to allow specification of inner and outer loops sizes from the command line, to allow automatic renaming of the output files, and to allow a variable minerals list to be passed to the post-processing routine, newpost.c (described below).

4.2.4 File newpost.c, CSCI# 30049 V1.1

This C program is a slight modification of postproc.c (CSCI# 30049 V1.0). The modifications consist of allowing the program to pass a minerals list as an ascii text file, rather than "hard-wiring" the allowed minerals in the code. The modified code was verified in the same manner as was the original. This C program locates specific data outputs in the concatenated EQ6 output files generated by running the program nxtinput.c (through nxti_bat.c or alln_bat.c), and copies the selected data to a separate file to facilitate analysis and entry into spreadsheets.

4.2.5 Files Lastpost.c, CSCI# 30051 V1.0

This C program processes the output of alln_bat.c and reduces the still extensive output to a form more amenable to plotting by selecting only every tenth output line.

4.3 Spreadsheets

Spreadsheet analyses were performed with Microsoft Excel version 97, installed on a PC. The specific spreadsheets used for results reported in this document are included in Ref 16.

4.4 Software Approved for QA Work

The software package, EQ3/6, Version 7.2b, was approved for QA work by LLNL (Memorandum to File from Royce E. Monks, dated March 28, 1997, QA designator 97/026). An installation and test report (Ref. 14) was written and submitted to the SCM, and the proper installation was verified, before the runs described in this report were made.

5. Calculation

The general scheme of the calculations starts with obtaining data for compositions, amounts, surface areas, and reaction rates of the various components of the Pu-ceramic WPs. These quantities are recalculated to the form required for entry into EQ6; mostly this consists of making such conversions as weight percentages of elements or component oxides to mole fractions of elements, degradation rates in micrometers/year into moles per square centimeter per second, etc. Spreadsheets (Ref. 16) provide details of these calculations, and the procedure is also described in detail in Ref. 17. The final part of the input to EQ6 consists of the composition

of J-13 well water together with a rate of influx to the WP that corresponds to suitably chosen percolation rates into a drift and drip rate into a WP (See Section 5.1.1.3). From time to time the water added to the WP from this simulated influx is removed, together with its solutes, to approximate reactive flow and transport through the WP via routines described in Section 4.2.1. The EQ6 output provides the results of modeling of the chemical degradation of the WP, or components thereof. Sometimes the degradation of the WP is divided into phases, e.g. degradation of HLW high level waste (HLW) glass before breach and exposure of the ceramic to the water. The results include the compositions and amounts of solid products and of substances in solution. Details of the results are presented below.

5.1 Calculation Inputs

5.1.1 WP Materials and Performance Parameters

This section provides a brief overview of the chemical characteristics of ceramic WPs. The emphasis is on the chemical composition and reactivity, rather than on the physical configurations within the packages, although the configurations were used for volume calculations to determine the overall chemistries and surface areas.

Material nomenclature used throughout this document includes: SB-575 N06022 (hereafter referred to as Alloy 22), SA-240 S31603 (hereafter referred to as 316L), UNS N06625 (hereafter referred to as Alloy 625) and SA-240 S30403 (hereafter referred to as 304L).

5.1.1.1 Chemical Characteristics of Representative Pu-Ceramic WPs

As modeled, the WP consists of a corrosion-resistant shell (consisting of an outer corrosion-allowance barrier of carbon steel, and a liner of Alloy 22), containing 5 HLW glass-pour canisters and a central spacer; the canisters are fabricated from 304L stainless steel. Each glass-pour canister contains 7 cylindrical sleeves, each consisting of 4 cans, for a total of 28 cans per glass-pour canister; the cans will be fabricated in 316L stainless steel (Ref. 19 specifies 316 stainless, which is assmed to be 316L; the compositions are extremely similar for purposes of this calculation). Each can contains a stack of 20 ceramic disks, each ~2.54 cm in thickness. After the 28 cans are emplaced in the canister, the canister is back-filled with molten HLW glass. An overview of the co-disposal concept can be found in Ref. 18, and the geometries and configurations of the Pu-ceramic, within the glass-pour canisters, are given in Ref 19.

The corrosion-resistant shell is assumed to have little effect on the geochemical reactions internal to the WP (Assumption 3.7). It is presumed that water transport through cracks in the carbon steel will be rapid, and the crack surfaces will readily oxidize to hematite or goethite, essentially in equilibrium with the corrosion products inside the package (e.g., sections 5.3.1 through 5.3.3).

For modeling the chemical behavior of this system, the chemical compositions of each of these materials, their masses, their surface areas and their corrosion or degradation rates are required. The spreadsheet masses5.xls (included in Ref.16) gives the calculations of masses and surface

areas. Table 5.1.1.1-1 gives the composition of the ceramic waste form, and table 5.1.1.1-2 gives the HLW glass composition. Table 5.1.1.1-3 gives the compositions of the stainless steel alloys.

Several simplifications were necessary to complete the calculations. The final WP design may include a central spacer, serving to hold the 5 glass-pour canisters apart; at the time the calculations reported herein were performed, there were few details available on the construction and corrosion rate of this central spacer, so it was ignored in the calculations. In addition, the glass-pour canisters may contain numerous small steel components, such as scalloped plates to hold the cans of ceramic in place. These small steel components were also ignored in the calculations.

Table 5.1.1.1-1. Pu-Ceramic Composition[1]

Oxide	Weight %
CaO	10.00
TiO_2	35.90
HfO_2	10.60
Gd_2O_3	7.90
UO_2	23.70
PuO_2	11.90

1. Ref. 19.

Table 5.1.1.1-2. Raw Glass Composition[1]

Component	Weight %
Ag	0.05
Al_2O_3	3.96
B_2O_3	10.28
$BaSO_4$	0.14
$Ca_3(PO_4)_2$	0.07
CaO	0.85
$CaSO_4$	0.08
Cr_2O_3	0.12
Cs_2O [2]	0.08
CuO	0.19
Fe_2O_3	7.04
FeO	3.12
K_2O	3.58
Li_2O	3.16
MgO	1.36
MnO	2.00
Na_2O	11.00
Na_2SO_4	0.36
NaCl	0.19
NaF	0.07
NiO	0.93
PbS	0.07
SiO_2	45.57
ThO_2 [2]	0.21
TiO_2 [2]	0.99
U_3O_8	2.20
Zeolite [3]	1.67
ZnO [2]	0.08
^{237}Np [4]	0.000751
^{239}Pu [4]	0.012342
Tc [4]	0.010797
Zr [4]	0.026415

1. Ref. 20 (Attachment I, Table 3.3.8, except as explained in note 4 below).
2. Not carried through EQ3/6 calculation, due to small amount relative to other WP components, or judgement of little significance.
3. Assumed to be analcime, due to high pour temperature of glass and high Na content.
4. Obtained by taking the "Grams/canister" entry of Ref. 20 (Attachment I, Table 3.3.3), multiplying by 100% and dividing by the presumed mass/canister of 1682 kg (Ref 20, Attachment I, footnote to Table 3.3.3). All Tc presumed to be ^{99}Tc; all Zr presumed to be ^{93}Zr.

Table 5.1.1.1-3. Metal Compositions (Weight Percent)[1]		
Element	316L	304L
C	0.030	0.030
N	0.100	0.100
Si	0.750	0.750
P	0.045	0.045
S	0.030	0.030
Cr	17.000	19.000
Mn	2.000	2.000
Fe	65.545	68.045
Ni	12.000	10.000
Mo	2.500	
Density	7.9497 g/cm^3	(not used)
1. Ref. 21.		

5.1.1.2 Chemical Composition of J-13 Well Water

It was assumed that the water composition entering the WP would be the same as for water from well J-13 (Assumptions 3.1 and 3.3). This water has been analyzed repeatedly over a span of at least two decades (Ref. 2). This composition is reproduced in Table 5.1.1.2-1.

Table 5.1.1.2-1. Composition of J-13 Well Water[1]

Component	Units[2]
Na^+	45.8
K^+	5.04
Ca^{++}	13.0
Mg^{++}	2.01
NO_3^-	8.78
Cl^-	7.14
F^-	2.18
SO_4^{--}	18.4
Si	28.5
PO_4^{---}	0.12
Alkalinity[3]	128.9
pH	7.41

1. Ref. 2.
2. mg/L, except for pH
3. Assumed to be HCO_3^-.

5.1.1.3 Drip Rate of J-13 Water into a WP

It is assumed (Assumption 3.14) that the drip rate onto a WP is the same as the rate at which water flows through the WP. The drip rate is taken from a correlation between percolation rate and drip rate (Ref. 22). Specifically percolation rates of 40 mm/yr and 8 mm/yr correlate with drip rates onto the WP of 0.15 m^3/yr and 0.015 m^3/yr, respectively. The choice of these particular percolation and drip rates is discussed in detail in Ref. 17.

For the present study, the range of allowed drip rates was extended to include an upper value of 0.5 m^3/yr and a lower value of 0.0015 m^3/yr. The upper value corresponds to the 95 percentile upper limit for a percolation rate of 40 mm/yr (as determined in Ref. 22 and Ref.23), and the lower value is simply $1/10^{th}$ the mean value for the 8 mm/yr percolation rate. These extreme values were used, because prior studies (Ref.18 above; Ref. 24; Ref. 25) suggested that when ceramic waste forms are co-disposed with glass, the greatest chance of Gd removal occurs when

initial high drip rates cause glass leaching and removal of alkali, and subsequent low drip rates allow acid to build from steel degradation (thus increasing the solubility of solid $GdOHCO_3$).

5.1.1.4 Densities and Molecular Weights of Solids

For input to criticality calculations conversions one must convert moles of solids, estimated to form, to solid volumes. A few solid phases contribute the overwhelming bulk of the total volume; the Table 5.1.1.4-1 provides some the densities and molar volumes for these phases.

Table 5.1.1.4-1. Densities and Molecular Weights of Precipitated Solids

Solid	Density (kg/m^3)	Molecular Weight[c]	Mol. Vol., cm^3/mol [c]	Calc. Dens., g/cm^3
Diaspore (AlOOH)	3400[a]	59.988	17.760	3.378
Hematite (Fe_2O_3)	5240[b]	159.692	30.274	5.275
Pyrolusite (MnO_2)	5060[a]	86.937	17.181	5.060a
Goethite (FeOOH)		88.854	20.820	4.268
Ni_2SiO_4		209.463	42.610	4.916
Nontronite-Ca		424.293	131.100	3.236
Nontronite-K		430.583	135.270	3.183
Nontronite-Mg		421.691	129.760	3.250
Nontronite-Na		425.267	132.110	3.219

References
[a] Roberts et al., 1974 (Ref. 26)
[b] Weast, 1977 (Ref. 10)
[c] Ref. 16 (EQ3/6 Data base, data0.nuc.R8), g/mole, except for pyrolusite, which is calculated from the density and molecular weight.

5.1.1.5 Atomic Weights

Atomic weights were taken from Ref. 21 and Ref. 27. These are listed in Ref. 16 (spreadsheet volmas21a, sheet VOLMASS).

5.1.1.6 Corrosion Rates

The corrosion rates used in the EQ6 runs are summarized in Table 5.1.1.6-1. Emphasis was placed on highest probable corrosion rates, to obtain the most favorable (conservative) conditions for removal of Gd from the system. For the HLW glass, the high rate was used in the two-stage runs (section 5.3) to ensure that the alkaline glass would be degraded and leached before the Pu-ceramic was exposed. The higher Pu-ceramic rates ensured the Gd in the ceramic would be exposed to the acid, produced by degradation of the stainless steel, in the two-stage runs. All Pu-ceramic rates are for metamict material. The high and average stainless steel

These three prior studies suggested that the greatest removal of Gd would occur at low drip rates in the second stage described above.

Two cases were modeled. In the first case, all package materials degrade together, albeit at different rates, and the drip rate of J-13 water into the package is kept constant throughout the run. Only one simulation of this first type was run, because this scenario tends to maintain moderate to high pH, and provides little opportunity for loss of gadolinium. For this one run, the amount of HLW glass was arbitrarily reduced to 75% of the amount expected for the WP (all other runs used the full amount of glass calculated for the WP as specified in spreadsheet masses5.xls in Ref. 16). This reduction in the amount of glass is conservative, since it reduces the alkali, potentially allowing more U and Pu to remain in the package, and increases the probability that Gd will be removed. For purposes of this report, use of the lower glass amount underscores that it is very difficult to achieve significant Gd loss when degradation of glass overlaps, in time, degradation of the Pu-ceramic.

In the second case, the run is broken into two stages with different drip rates. In the first stage, the HLW glass and steel package materials react together, but the ceramic is not included in the calculation; the drip rate is kept high (0.5 m^3/yr). The first stage models degradation of the HLW glass, and removal of most of the alkaline glass components, as if the ceramic were perfectly protected from corrosion. The first stage is terminated when the pH reaches a plateau minimum of ~6, at ~3.8×10^3 years. In the second stage, drip rate is reduced to 0.015 or 0.0015 m^3/yr, and the ceramic is included in the calculation and reacts with the remaining steels and the minerals formed from degradation of the HLW glass. The pH may then drop to ~5.25, as steel continues to corrode, but the rate of influx of J-13 (which is mildly alkaline) is reduced. There follows a period of relatively low pH, which may persist for thousands to tens of thousands of years; in this period of low pH, the solubility of $GdOHCO_3$ is highest. For the runs with drip rate = 0.0015 m^3/yr, dissolved Gd concentrations can reach 10^{-3} to 3.5×10^{-2} molal (up to 5500 ppm); for the runs with a drip rate of 0.015 m^3/yr, the maximum Gd concentration does not exceed ~2×10^{-4} molal. In the remainder of each run, the pH gradually rises, due to several factors: the inherent alkalinity of the J-13 water; the alkalinity built into the ceramic waste form; and the buffering capacity of the clays in the system. Seven simulations of this second type were run; only four produced a significant loss of Gd (~10 to 15%) from the WP.

Table 5.3-1 summarizes the conditions used and total Gd loss for the 8 runs, and Table 5.3-2 gives the peak aqueous Gd concentrations and the minimum pH achieved in the runs. With the range of conditions tested, the period of high Gd solubility is generally not long enough to remove substantial Gd from the systems. In all runs, formation of muscovite and celadonite was suppressed; these are high-temperature sheet silicates that are not observed to form in low-temperature environments, and the effect of the suppression is to generate more smectite clays in the run (a more realistic result). In runs 2 through 8, formation of zircon was suppressed for conservatism, since zircon lowers Zr (Hf) solubility and zircon is also not observed to form in low-temperature environments. However, few natural environments would have such high Zr(Hf) concentrations as the degraded WP, and formation of amorphous zircon may indeed be possible on the time scales of ~10^6 years.

Prior studies (Ref.18; Ref. 24; Ref. 25) have suggested higher Gd loss from degraded WPs. To ensure there were no errors in the methodology of the present study, the calculations were contrasted with those in Ref.18, Ref. 24 and Ref. 25. Several explanations were found for the comparatively low Gd loss of the present study. First, in the current study, the mass of acid-producing metals is less; Alloy 22 is used for the inner corrosion resistant barrier, and corrodes much more slowly than the Alloy 625 suggested for the barrier in Ref.18. In addition, the U-Al alloy considered in Ref. 18 is itself a producer of acid during oxidation, and the U-Al alloy is admixed with abundant steel plates, which also produce acid upon degradation. In contrast, the Pu-ceramic is somewhat alkaline, and is capable of neutralizing some of the acid produced by the other metals. Given the comparatively small amount of acid produced, even the alkalinity of the J-13 water is enough to bring the system pH back up to ~7.7 before substantial Gd can be lost.

Table 5.3-1. Summary of EQ6 Runs

Run #	Stages	EQ6 File Name[1]	Corrosion Rates[2]	J-13 Drip Rates	Modeled Time, yrs	% Gd Loss[3]
1	1	Cer0_0015I!.6i	HLW: avg SS: avg Cer: avg	0.0015 m^3/yr	1.07×10^6	0.0432
2	2	Cerd1N0_5I.6i & Cerd1W0_0015I.6i	HLW: high SS: high Cer: avg	0.5 & 0.0015 m^3/yr	3.77×10^3 & 6.49×10^5	1.86
3	2	Cerd1N0_5I.6i & Cerd1W0_015I.6i	HLW: high SS: high Cer: avg	0.5 & 0.015 m^3/yr	3.77×10^3 & 1.12×10^5	1.24
4	2	Cerd2N0_5I.6i & Cerd2W0_0015I.6i	HLW: high SS: avg Cer: avg	0.5 & 0.0015 m^3/yr	3.78×10^3 & 1.46×10^6	14.8
5	2	Cerd1N0_5I.6i & Cerd3W0_0015I.6i	HLW: high SS: high Cer: high	0.5 & 0.0015 m^3/yr	3.77×10^3 & 6.52×10^5	9.58
6	2	Cerd1N0_5I.6i & Cerd3W0_0015I_CO2_LO.6i	HLW: high SS: high Cer: high	0.5 & 0.0015 m^3/yr	3.77×10^3 & 6.50×10^5	13.2
7	2	Cerd2N0_5I.6i & Cerd4W0_015I.6i	HLW: high SS: avg Cer: high	0.5 & 0.015 m^3/yr	3.78×10^3 & 1.33×10^5	12.2
8	2	Cerd1N0_5I.6i & Cerd5W0_0015I_CO2_LO.6i	HLW: high SS: high Cer: very high	0.5 & 0.0015 m^3/yr	3.77×10^3 & 1.09×10^6	0.0369

1. Where 2 file names are given, the first represents the 1st stage of the run, to ~3.8×10^3 years, with HLW, steels, but NO Pu-ceramic, and the second represents the 2nd stage of the run, from ~3.8×10^3 years till the end, including the Pu-ceramic and all other remaining reactants.
2. For HLW, the avg. rate=2×10^{-4} $g/m^2/day$, and the high rate=2.8×10^{-2} $g/m^2/day$; stainless steel (SS, both 304L and 316L) the average rate=0.1 μm/yr, and the high rate=1 μm/yr; and for ceramic (cer), the very high rate=4×10^{-1} $g/m^2/day$, the high rate=4×10^{-2} $g/m^2/day$, and the avg rate=2×10^{-3} $g/m^2/day$ (see spreadsheet masses5.xls in Ref. 16 for rationale; all ceramic rates are for metamict material).
3. Calculated in spreadsheet Gd%remaining.xls (Ref. 16).

Table 5.3-2. Maximum Aqueous Gd and Minimum pH for EQ6 Runs[1]

Run #	Gd Maximum (Molal)	Time of Maximum Gd Concentration[2]	pH Minimum	Time of pH Minimum[2]
1	6.15×10^{-3}	13.4	6.33	1.34
2	1.14×10^{-3}	13.5	5.25	5.35
3	1.20×10^{-4}	6.33	5.49	5.48
4[3]	5.32×10^{-3} / 4.03×10^{-3}	21.2 / 32.5	5.49 / 5.47	22.5 / 60.0
5	1.59×10^{-2}	7.26	5.32	5.60
6	1.80×10^{-2}	8.24	5.32	5.60
7	2.31×10^{-4}	58.6	5.87	59.8
8	1.06×10^{-4}	3.77	6.13	3.77

1. Taken from corresponding allpost file (Ref. 16); used "changing timesteps elements" blocks.
2. Thousands of years.
3. Run 4 had two distinct peaks; the 2nd is responsible for the bulk of the Gd loss.

Sections 5.3.1 through 5.3.3 below discuss, in detail, runs 1, 2 and 6 (run 6 produced the second highest Gd loss); the emphasis of these sections is to establish the long-term chemistry of the degraded WP, over time scales of 10^5 to 10^6 years. The amounts listed in the figures are per initial liter of void volume; this normalization is necessary to simplify the EQ3/6 calculations (in particular, to maintain ~1 liter of fluid in equilibrium with the precipitated solids). The modeled system contains 3737.921 liters of void volume (calculated in spreadsheet masses5.xls, Ref. 16), and the amounts in the figures must be multiplied by 3737.921 to obtain the total moles or volumes of minerals in the package. Section 5.3.4 examines the aqueous Gd maxima for runs 6 and 7, at a higher resolution, to show the covariation among pH, Gd concentrations, and the presence of other solids in the system.

5.3.1 Run 1: Ceramic, Steel, and HLW degrade simultaneously

Figure 5.3.1-1 shows volumes of principal minerals formed during the run. By volume, smectite clay is overwhelmingly the most significant solid phase, followed by hematite and nickel silicate. The smectites have a high proportion of nontronite ($Fe_2(Na,Mg,½Ca,½Mg)_{0.33}Al_{0.33}Si_{3.67}H_2O_{12}$), and grow at the expense of the hematite as the run progresses. Zircon was not suppressed in this run. The calculations also predict that several Ca-rich carbonates and rutile (TiO_2) will also precipitate (not shown in figure 5.3.1-1).

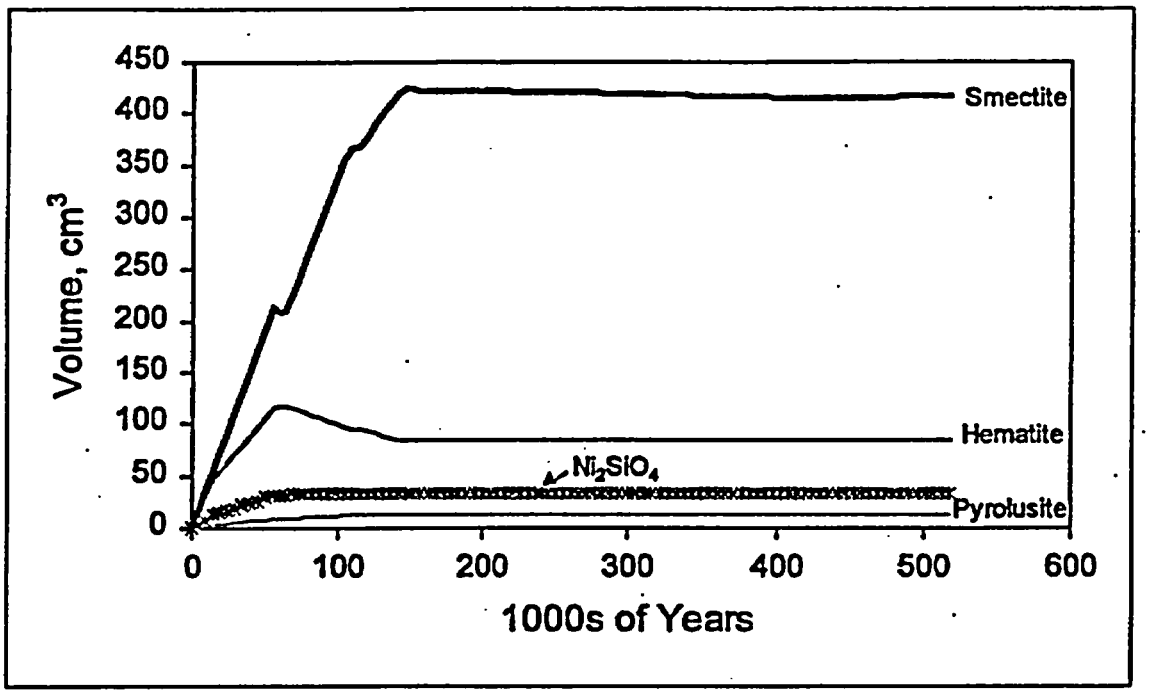

Figure 5.3.1-1. Volumes of Some Principal Minerals, Run 1 (cm^3 per Liter Initial Void Space)

Figure 5.3.1-2 gives the pH and molality of B, Gd, Pu and U as functions of time. The lowest pH (6.33) is reached early in the run, when the rate of steel degradation is still high; because the alkaline HLW glass degrades in the same stage, it is not possible to reach lower pH. Since this run uses the average HLW corrosion rate (see table 5.3-1), the alkaline glass continues to decompose after the steels has largely reacted, causing pH to reach a high of ~9.2 at 10^5 years. In the period of higher pH, both U and Pu have high solubility as carbonate complexes. The highest Gd solubility, ~10^{-4} molal, is reached early in the run (<2×10^4 years), and is not of sufficient extent or intensity to cause much Gd loss from the system.

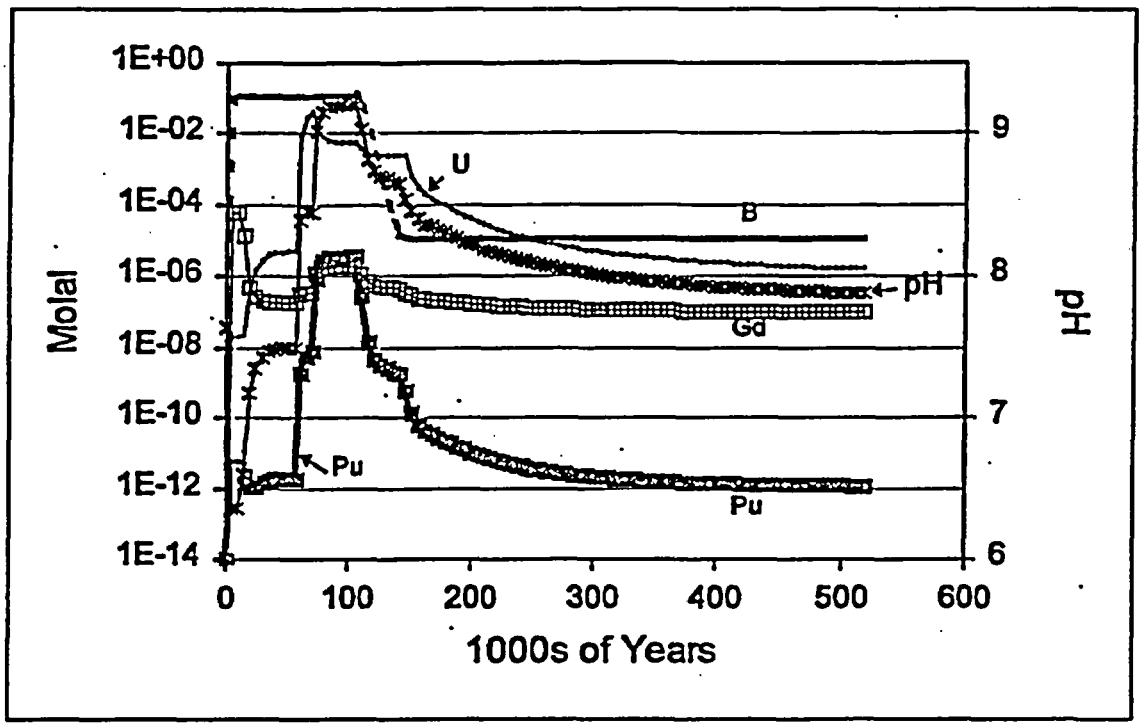

Figure 5.3.1-2. Aqueous Concentrations of Selected Elements and pH, Run 1

The amounts of minor minerals in the system are shown in figure 5.3.1-3, on a log scale. The peak in U solubility is due not just to the high pH. Uranium solubility is also enhanced by the temporary consumption of silica in the form of petalite ($LiAlSi_4O_{10}$) and potassium feldspar (not plotted). This silica loss destabilizes soddyite ($(UO_2)_2(SiO_4)\cdot 2H_2O$), the principal solubility-controlling phase for uranium. The phase $GdOHCO_3$ begins to form after ~2×10^4 years, after the pH climbs above 7, and steadily builds until ~4×10^5 years, when the supply of Pu-ceramic becomes exhausted.

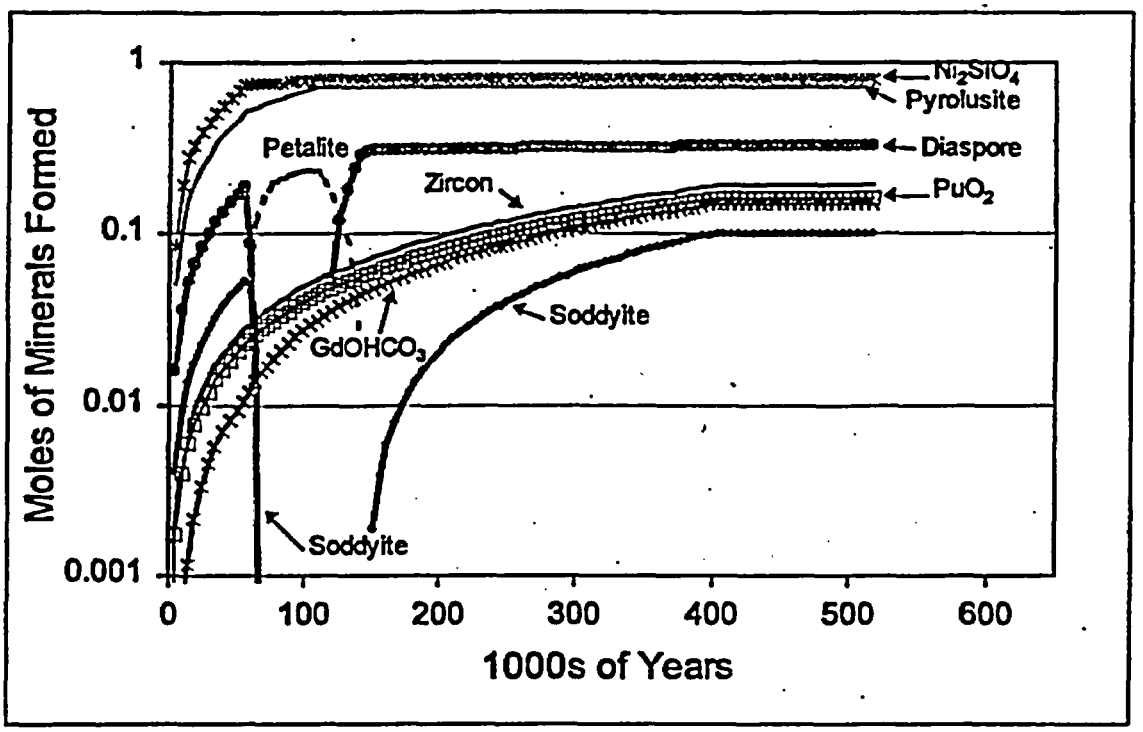

Figure 5.3.1-3. Moles of Minor Solid Phases Formed in Run 1

Table 5.3.1-1 gives the percent remaining (of original package contents) for selected elements. The minima for Al and Si are artifacts; with the newpost program described in section 4.2.4, all minerals anticipated to form, must be specified explicitly, else their masses will not be tracked in the EQ6 output. In this case, K-feldspar ($KAlSi_3O_8$) is predicted to form for several tens of thousand of years near the beginning of the run, then redissolve. Because K-feldspar was not explicitly tracked, its Al and Si did not appear in the mass balance. It is also possible for the total Si to rise slightly above 100% at long times, because the J-13 water contains dissolved Si. Table 5.3.1-2 gives the total mass in kg, remaining as solids, for the entire package. A column is included for the approximate ^{235}U generated by decay of ^{239}Pu, on the assumption that moles $^{235}U_{gen} = {}^{239}Pu_{initial}\,[1-(\frac{1}{2})^{(t/\tau)}]$, where t is time in years, and τ is the half-life of ^{239}Pu in years (taken from Ref. 27).

Table 5.3.1-1. Percentages of Selected Elements Remaining in Degraded WP: Run 1, Cer0_0015I!.6i, All Components Degrade Together[1]

Time[2]	U[3]	Hf (Zr)[4]	Pu[5] (raw)	Pu[6] (decay)	Fe	Ni	Mn	Na	Al	Si	Gd
0.00	100.00	100.00	100.00	100.00	99.97	99.97	99.99	100.00	100.00	100.00	100.00
4.29	100.00	100.00	100.00	88.40	100.00	99.76	100.00	96.13	100.01	99.99	99.81
9.64	100.00	100.00	100.00	75.78	99.98	99.55	99.99	91.30	100.01	99.97	99.85
23.87	100.01	100.00	100.01	50.33	99.98	99.48	99.99	78.74	100.02	99.93	99.84
59.79	97.22	100.03	100.04	17.92	99.97	99.46	99.99	48.66	93.07	97.54	99.87
104.10	54.62	100.03	100.00	5.01	99.96	99.47	99.99	10.08	84.43	94.53	99.85
204.50	46.01	100.07	100.05	0.28	99.97	99.46	99.99	2.25	100.02	99.28	99.89
302.80	45.95	100.11	100.11	0.02	99.96	99.46	99.99	0.97	100.02	98.95	99.91
402.70	45.93	100.15	100.11	0.00	99.95	99.46	99.99	0.68	99.99	98.58	99.96
517.50	45.93	100.15	100.11	0.00	99.95	99.46	99.99	0.63	100.01	98.65	99.96

1. Data from spreadsheet %remain.xls, sheet run1_%rem (Ref.16).
2. Time in thousands of years.
3. As calculated by EQ6, not including ^{235}U generated from ^{239}Pu decay.
4. Hf is modeled as Zr in EQ6 runs (see assumption 3.16)
5. Pu calculated by EQ6 (see assumption 3.17).
6. Pu decayed, as if all Pu were ^{239}Pu.

Table 5.3.1-2. Kilograms of Selected Elements Remaining as Solids in Entire Degraded WP: Run 1, Cer0_0015I!.6i, All Components Degrade Together[1]

Time[2]	U	Hf (Zr)[3]	^{239}Pu raw[4]	^{239}Pu decayed[5]	^{235}U gen[5]	Fe	Ni	Mn	Na	Al	Si	Gd
0.00	391.70	125.71	143.96	143.96	0.00	2420.61	346.70	147.50	489.18	132.21	1263.19	93.61
4.29	391.70	125.71	143.96	127.47	16.22	2421.19	345.97	147.51	470.21	132.21	1263.07	93.44
9.64	391.71	125.71	143.96	109.48	33.91	2420.71	345.24	147.50	446.60	132.22	1262.84	93.47
23.87	391.73	125.71	143.97	73.10	69.69	2420.71	345.01	147.49	385.17	132.23	1262.28	93.46
59.79	380.81	125.75	144.02	26.37	115.68	2420.64	344.94	147.49	238.02	123.05	1232.07	93.49
104.10	213.94	125.74	143.97	7.49	134.19	2420.36	344.95	147.49	49.30	111.62	1194.05	93.47
204.50	180.21	125.80	144.04	0.43	141.20	2420.56	344.94	147.49	11.01	132.23	1254.10	93.51
302.80	179.99	125.85	144.12	0.03	141.68	2420.39	344.94	147.49	4.74	132.23	1249.96	93.53
402.70	179.92	125.90	144.13	0.00	141.71	2419.97	344.94	147.49	3.31	132.19	1245.25	93.57
517.50	179.92	125.90	144.13	0.00	141.71	2420.14	344.94	147.49	3.06	132.22	1246.17	93.57

1. Data from spreadsheet %remain.xls, sheet run1_TOT (Ref. 16).
2. Time in thousands of years.
3. Hf is modeled as Zr in EQ6 runs (see assumption 3.16)
4. As calculated by EQ6, which currently has no provision for radioactive decay.
5. All Pu taken as ^{239}Pu and decayed to produce ^{235}U (see assumption 3.17).

5.3.2 Run 2: HLW Glass Degrades in First Stage with High Drip Rates; Ceramic Degrades in Second Stage with Average Rate

The volumes of principal minerals, per liter of void space, are shown in figure 5.3.2-1. Once again, smectite clay overwhelms the volume of the product phases. Ultimately, ~38% more smectite forms than in run 1, apparently at the expense of hematite. At long times, the total volume of product minerals is 2.4 m^3 for the entire package. The volume calculation uses the formal smectite and hematite particle densities of 3.2 and 5.3, respectively, and takes no account of imperfect packing and interstitial water. Given the total volume inside the corrosion barrier is 7.1 m^3, and an initial void space of 3.7 m^3, clays can be expected to fill a large portion of the degraded WP. The calculations also predict that several Ca-rich carbonates and rutile (TiO_2) will also precipitate (not shown in figure 5.3.2-1).

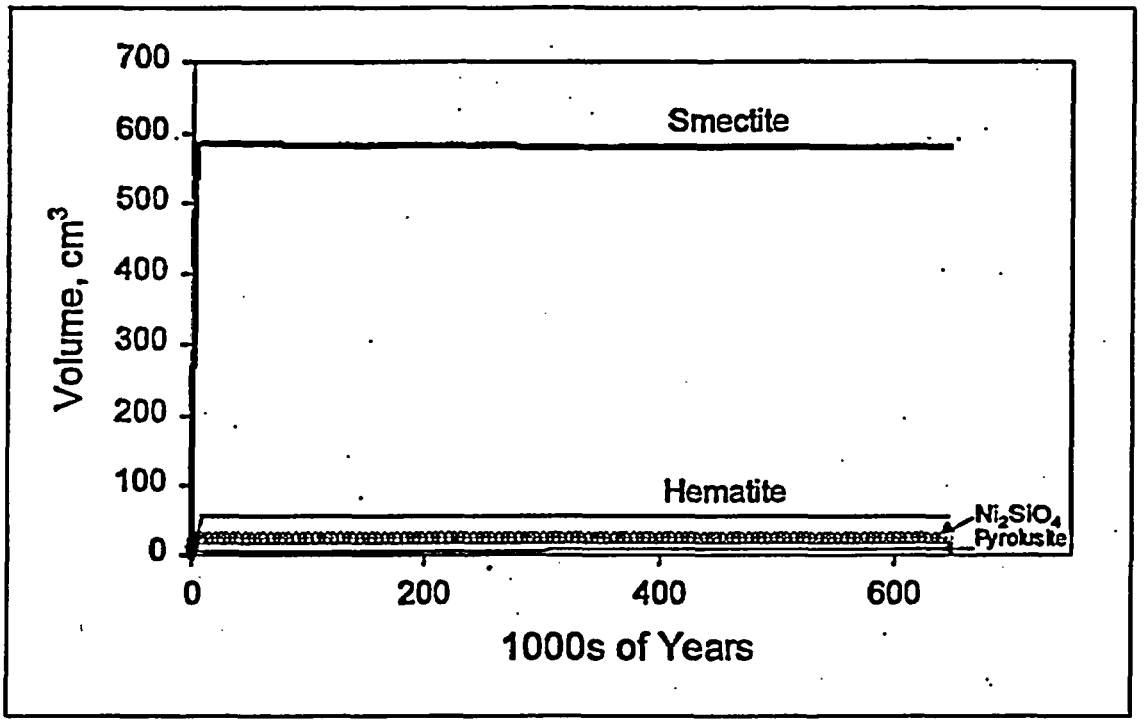

Figure 5.3.2- 1. Volumes of Some Principal Minerals, Run 2 (cm^3 per Liter Initial Void Space)

The pH and molality of aqueous B, Gd, Pu and U are shown in figure 5.3.2-2. The pH reaches 9 in the first 10^3 years, as the HLW completely degrades, then the degradation of steels gradually drops the pH to 5.32. At 3.8×10^3 years, the Pu-ceramic is exposed. Since the Pu and U in the ceramic are never exposed to the high pH (typical of glass degradation), these elements never reach very high concentrations in the aqueous phase. The peak in aqueous Gd (~10^{-3} moles/liter) reflects the period of lowest pH, between 5×10^3 and 10^4 years. However, the low flow rate in

this period prevents substantial loss of Gd. Eventually, the pH climbs up to 7.7, controlled principally by the influx of J-13 water, and the aqueous Gd drops to 10^{-7} molal.

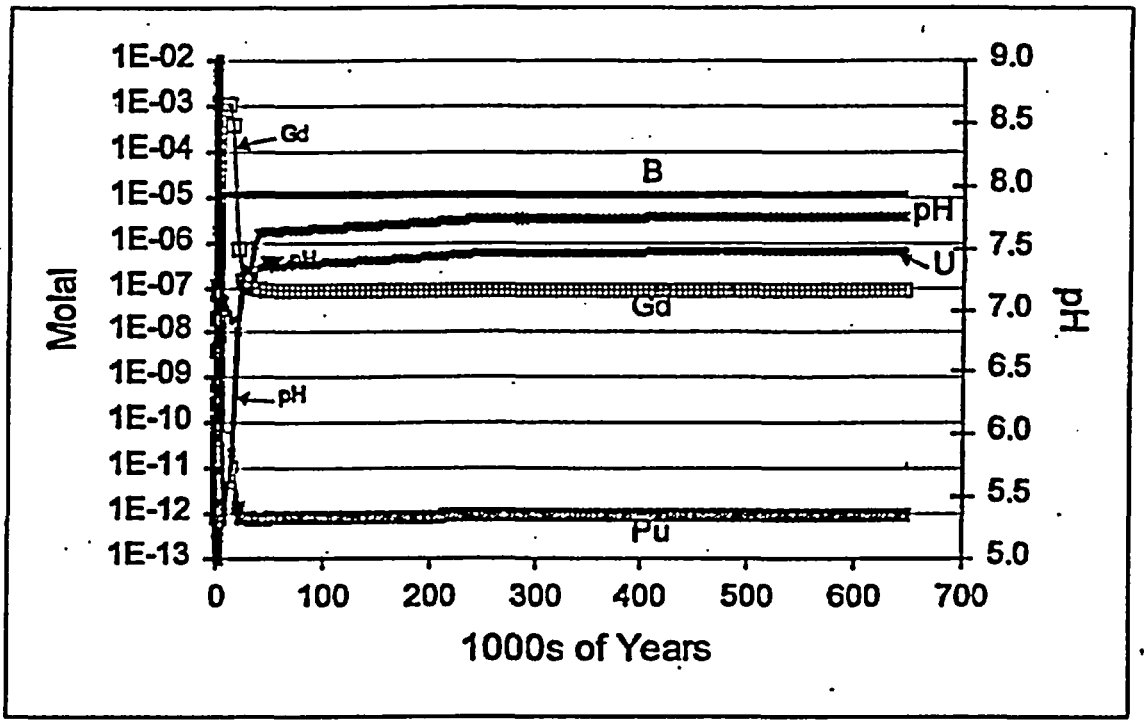

Figure 5.3.2-2. Aqueous Concentrations of Selected Elements and pH, Run 2

Table 5.3.2-1 gives the percent remaining (of original package contents) for selected elements. The minima for Al and Si at early times (<3770 years) are artifacts; with the newpost program described in section 4.2, all minerals anticipated to form, must be specified explicitly, else their masses will not be tracked in the EQ6 output. In this case, several minor silicates formed in the early part of the run, then redissolved. The Al and Si from these minor phases do not appear in the mass balance. Table 5.3.2-2 gives the total mass in kg, remaining as solids, for the entire package. A column is included for the approximate ^{235}U generated by decay of ^{239}Pu.

Table 5.3.2-1. Percentages of Selected Elements Remaining in Degraded WP: Run 2, Cerd1N0_5I.6i & Cerd1W0_0015I.6i, Pu-Ceramic Degrades in Second Stage[1]

Time[2]	U[3]	Hf (Zr)[4]	Pu[5] (raw)	Pu[6] (decay)	Fe	Ni	Mn	Na	Al	Si	Gd
0.00	100.00	100.00	100.00	100.00	99.97	99.97	99.99	100.00	100.00	100.00	100.00
0.10	96.60	100.00	100.00	99.70	99.91	99.90	99.97	90.15	100.02	98.09	100.00
0.30	90.22	100.00	100.00	99.14	99.62	99.59	99.87	71.69	100.06	94.51	100.00
1.00	67.34	100.01	99.99	97.15	98.66	98.59	99.54	5.50	100.17	81.65	100.00
3.77	66.77	100.00	99.99	89.71	98.39	93.86	99.42	0.60	100.17	100.23	100.00
7.80	66.76	100.00	99.99	79.89	98.40	71.05	99.43	0.58	100.19	100.25	98.99
11.53	66.76	100.00	99.99	71.77	98.40	71.40	99.43	0.54	100.20	100.25	98.06
30.20	66.77	100.01	100.00	41.95	98.37	72.01	99.43	0.43	100.16	100.22	97.96
62.12	66.77	100.02	100.01	16.75	98.38	72.01	99.43	0.50	100.17	100.24	97.97
100.30	66.79	100.04	100.03	5.59	98.38	72.01	99.43	0.56	100.17	100.25	97.99
204.10	66.82	100.09	100.09	0.28	98.40	72.01	99.43	0.65	100.19	100.31	98.04
302.00	66.85	100.13	100.16	0.02	98.38	72.01	99.43	0.68	100.18	100.32	98.11
400.30	66.89	100.19	100.16	0.00	98.39	72.00	99.43	0.69	100.18	100.35	98.10
503.80	66.88	100.16	100.20	0.00	98.39	72.00	99.43	0.70	100.17	100.38	98.14
601.80	66.88	100.16	100.20	0.00	98.39	72.00	99.43	0.71	100.18	100.41	98.14
645.60	66.88	100.16	100.20	0.00	98.39	72.00	99.43	0.71	100.18	100.43	98.14

1. Data from spreadsheet %remain.xls, sheet run2_%rem (Ref. 16).
2. Time in thousands of years.
3. As calculated by EQ6, not including ^{235}U generated from ^{239}Pu decay.
4. Hf is modeled as Zr in EQ6 runs (see assumption 3.16)
5. Pu calculated by EQ6 (see assumption 3.17).
6. Pu decayed, as if all Pu were ^{239}Pu.

Table 5.3.2-2. Kilograms of Selected Elements Remaining as Solids in Entire Degraded WP: run 2, Cerd1N0_5I.6i & Cerd1W0_0015I.6i, Pu-Ceramic Degrades in Second Stage[1]

Time[2]	U	Hf (Zr)[3]	^{239}Pu raw[4]	^{239}Pu decayed[5]	^{235}U gen[5]	Fe	Ni	Mn	Na	Al	Si	Gd
0.00	427.37	126.70	144.20	144.20	0.00	2561.13	360.67	177.09	653.23	176.55	1679.39	93.61
0.10	412.82	126.70	144.20	143.77	0.42	2559.47	360.42	177.06	588.89	176.58	1647.32	93.61
0.30	385.55	126.70	144.20	142.98	1.20	2552.02	359.31	176.89	468.27	176.65	1587.17	93.61
1.00	287.79	126.71	144.19	140.15	3.98	2527.58	355.69	176.30	35.93	176.84	1371.15	93.61
3.77	285.34	126.70	144.19	129.54	14.40	2520.45	338.62	176.07	3.90	176.84	1683.11	93.61
7.80	285.32	126.70	144.18	115.54	28.17	2520.84	256.32	176.09	3.78	176.88	1683.51	92.66
11.53	285.32	126.70	144.19	103.93	39.58	2520.84	257.59	176.09	3.56	176.88	1683.58	91.80
30.20	285.35	126.71	144.20	61.17	81.64	2520.14	259.79	176.09	2.83	176.83	1683.08	91.70
62.12	285.37	126.72	144.22	24.71	117.50	2520.22	259.79	176.09	3.26	176.84	1683.34	91.71
100.30	285.42	126.75	144.24	8.36	133.61	2520.30	259.79	176.09	3.64	176.83	1683.47	91.73
204.10	285.58	126.81	144.34	0.44	141.48	2520.89	259.79	176.09	4.21	176.88	1684.51	91.78
302.00	285.69	126.86	144.43	0.03	141.98	2520.43	259.79	176.09	4.42	176.85	1684.73	91.84
400.30	285.88	126.95	144.44	0.00	142.01	2520.63	259.74	176.09	4.49	176.85	1685.18	91.83
503.80	285.81	126.90	144.49	0.00	142.07	2520.51	259.74	176.09	4.57	176.84	1685.68	91.87
601.80	285.81	126.90	144.49	0.00	142.07	2520.68	259.74	176.09	4.61	176.85	1686.22	91.87
645.60	285.81	126.90	144.49	0.00	142.07	2520.59	259.74	176.09	4.62	176.85	1686.53	91.87

1. Data from spreadsheet %remain.xls, sheet run2_TOT (Ref. 16).
2. Time in thousands of years.
3. Hf is modeled as Zr in EQ6 runs (see assumption 3.16).
4. As calculated by EQ6, which currently has no provision for radioactive decay.
5. All Pu taken as ^{239}Pu and decayed to produce ^{235}U (see assumption 3.17).

5.3.3 Run 6: HLW Glass Degrades in First Stage; High Drip Rates; Ceramic Degrades in Second Stage with High Rate, and Lower CO_2 Levels

Like run 2, run 6 is staged. There is a rapid leaching of the HLW glass in the first stage, under a relatively high drip rate; followed by corrosion of the Pu-ceramic in a second stage with a much lower drip rate. However run 6 differs from both runs 1 and 2 in several respects. First, run 6 has a significant Gd loss (13.2%). The ceramic degradation rate is 20 times greater than in runs 1 and 2, and the ambient CO_2 pressure was dropped by ~1 order of magnitude relative to runs 1 and 2.

Figure 5.3.3-1 shows the volumes of principal minerals, per initial liter of void space (as discussed in section 5.3, to obtain the total volume of precipitated minerals in the system, the values in figure 5.3.3-1 must be multiplied by 3737.921, the initial number of liters of void space). The calculations also predict that several Ca-rich carbonates and rutile (TiO_2) will also precipitate (not shown in figure 5.3.3-1). Overall, the volumes of principal minerals are very similar to those for run 2.

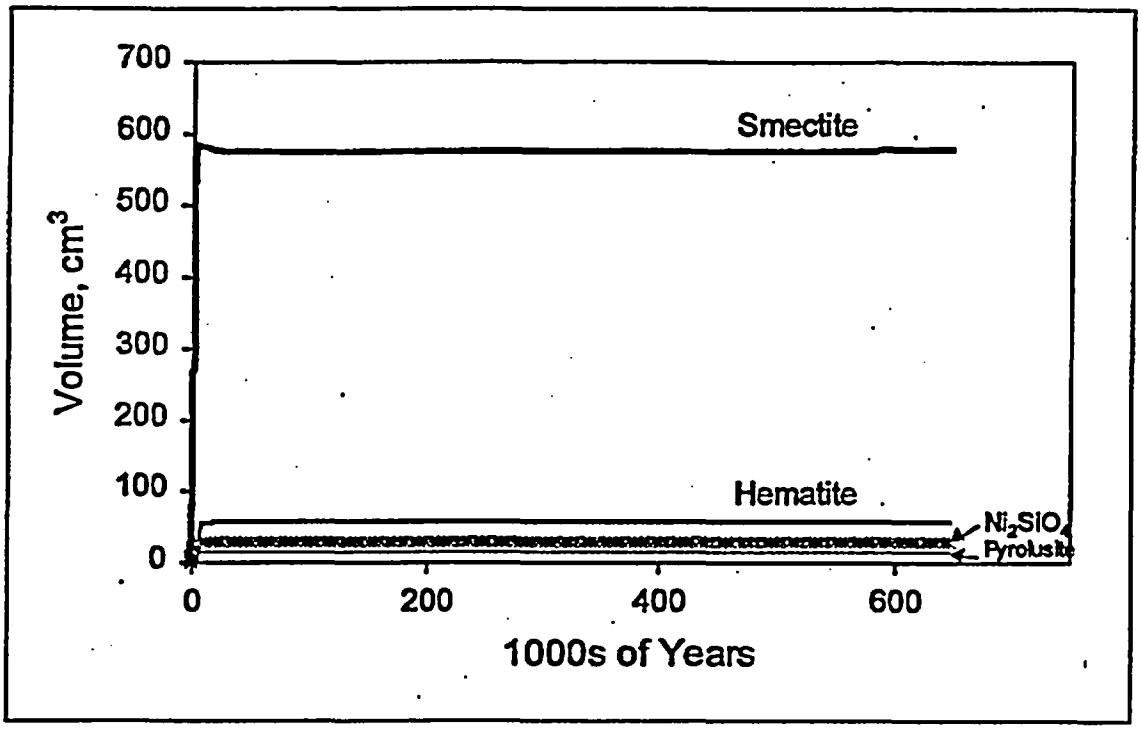

Figure 5.3.3- 1. Volumes of Some Principal Minerals, Run 6 (cm^3 per Liter Initial Void Space)

The pH and molality of selected elements, for run 6, are shown in figure 5.3.3-2. As with run 2 The pH reaches 9 in the first 10^3 years, as the HLW completely degrades. Then the degradation of steels gradually drops the pH to 5.5. At 3.8×10^3 years, the Pu-ceramic is exposed. Since the Pu and U in the ceramic are never exposed to the high pH (typical of glass degradation), these elements never reach very high concentrations in the aqueous phase. Compared to run 2, the period of low pH and high Gd concentrations is shorter in duration; however, the peak in aqueous Gd (~10^{-2} moles/liter) is apparently sufficient to yield 13.2% Gd loss from the system. Eventually, the pH climbs up to 8.5, controlled principally by equilibrium of J-13 water at the lower CO_2 pressure ($10^{-3.5}$ atm) specified for this run.

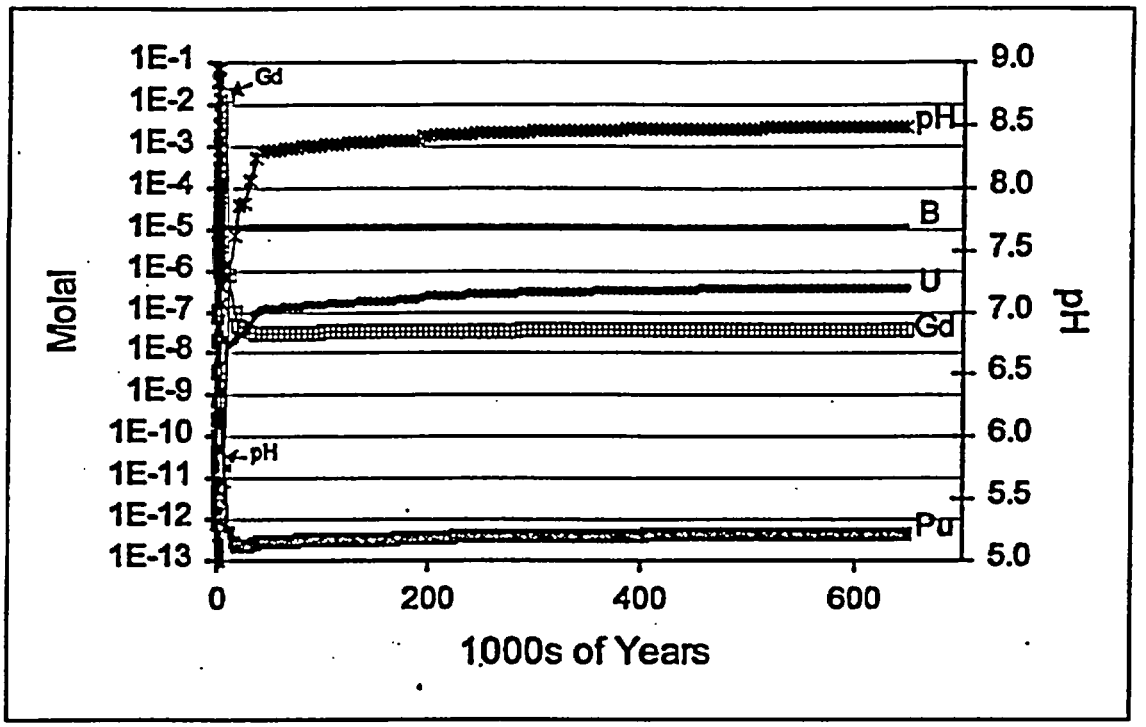

Figure 5.3.3-2. Aqueous Concentrations of Selected Elements and pH, Run 6

Table 5.3.3-1 gives the percent remaining (of original package contents) for selected elements. The minimum for Si at early times (<3770 years) is artificial; with the newpost program described in section 4.2, all minerals anticipated to form, must be specified explicitly, else their masses will not be tracked in the EQ6 output. In this case, several minor silicates formed in the early part of the run, then redissolved. The Si from these minor phases does not appear in the mass balance. Table 5.3.3-2 gives the total mass in kg, remaining as solids, for the entire package. A column is included for the approximate ^{235}U generated by decay of ^{239}Pu.

Table 5.3.3-1. Percentages of Selected Elements Remaining in Degraded WP: run 6, Cerd1N0_5I.6i & Cerd3W0_0015I_CO2_LO.6i, Pu-ceramic Degrades in Second Stage, CO_2 Pressure Lowered to $10^{-3.5}$ atm[1]

Time[2]	U[3]	Hf (Zr)[4]	Pu[5] (raw)	Pu[6] (decay)	Fe	Ni	Mn	Na	Al	Si	Gd
0.00	100.00	100.00	100.00	100.00	99.97	99.97	100.00	100.00	100.00	100.00	100.00
0.10	96.60	100.00	100.00	99.70	99.91	99.90	99.98	90.15	100.02	98.09	100.00
0.30	90.22	100.00	100.00	99.14	99.62	99.59	99.88	71.69	100.06	94.51	100.00
1.00	67.34	100.01	99.99	97.15	98.66	98.59	99.54	5.50	100.17	81.65	100.00
3.77	66.77	100.00	99.99	89.71	98.39	93.86	99.41	0.60	100.17	100.23	100.00
7.85	66.76	100.00	99.99	79.79	98.39	81.05	99.42	0.36	100.17	100.22	79.69
11.50	66.77	100.01	100.01	71.84	98.40	83.26	99.42	0.27	100.19	100.25	86.81
30.86	66.75	100.00	100.01	41.17	98.39	83.26	99.42	0.30	100.19	100.25	86.78
63.11	66.75	100.00	100.01	16.28	98.39	83.26	99.42	0.45	100.18	100.24	86.78
101.30	66.75	100.00	100.01	5.43	98.38	83.26	99.42	0.58	100.17	100.25	86.78
205.10	66.75	100.00	100.01	0.27	98.39	83.26	99.42	0.82	100.17	100.29	86.78
303.40	66.75	100.00	100.01	0.02	98.40	83.26	99.42	0.97	100.19	100.34	86.78
401.60	66.75	100.00	100.01	0.00	98.39	83.26	99.42	1.07	100.18	100.36	86.78
505.10	66.75	100.00	100.01	0.00	98.39	83.26	99.42	1.13	100.18	100.38	86.78
603.20	66.75	100.00	100.01	0.00	98.40	83.26	99.42	1.17	100.18	100.42	86.78
647.00	66.75	100.00	100.01	0.00	98.39	83.26	99.42	1.19	100.18	100.43	86.78

1. Data from spreadsheet %remaining.xls, sheet run6_%rem (Ref. 16).
2. Time in thousands of years.
3. As calculated by EQ6, not including ^{235}U generated from ^{239}Pu decay.
4. Hf is modeled as Zr in EQ6 runs (see assumption 3.16)
5. Pu calculated by EQ6 (see assumption 3.17).
6. Pu decayed, as if all Pu were ^{239}Pu.

Table 5.3.3-2. Kilograms of Selected Elements Remaining as Solids in Entire Degraded WP: run 6, Cerd1N0_5I.6i & Cerd3W0_0015I_CO2_LO.6i, Pu-ceramic Degrades in Second Stage, CO_2 Pressure Lowered to $10^{-3.5}$ atm[1]

Time[2]	U	Hf (Zr)[3]	^{239}Pu raw[4]	^{239}Pu decayed[5]	^{235}U gen[5]	Fe	Ni	Mn	Na	Al	Si	Gd[6]
0.00	427.37	126.70	144.20	144.20	0.00	2561.13	360.67	177.11	653.23	176.55	1679.39	93.61
0.10	412.82	126.70	144.20	143.77	0.42	2559.47	360.42	177.08	588.89	176.58	1647.32	93.61
0.30	385.55	126.70	144.20	142.98	1.20	2552.02	359.31	176.91	468.27	176.65	1587.17	93.61
1.00	287.79	126.71	144.19	140.15	3.98	2527.58	355.69	176.31	35.93	176.84	1371.15	93.61
3.77	285.34	126.70	144.19	129.54	14.40	2520.45	338.62	176.07	3.90	176.84	1683.11	93.61
7.85	285.32	126.70	144.19	115.39	28.32	2520.46	292.40	176.09	2.37	176.84	1683.08	74.60
11.50	285.35	126.71	144.21	104.03	39.50	2520.89	300.39	176.09	1.80	176.87	1683.44	81.27
30.86	285.27	126.70	144.22	60.04	82.77	2520.66	300.39	176.09	1.93	176.87	1683.56	81.23
63.11	285.27	126.70	144.22	24.03	118.18	2520.51	300.39	176.09	2.91	176.85	1683.43	81.23
101.30	285.27	126.70	144.22	8.12	133.81	2520.26	300.39	176.09	3.79	176.84	1683.58	81.23
205.10	285.27	126.70	144.22	0.43	141.38	2520.51	300.39	176.09	5.35	176.85	1684.20	81.23
303.40	285.27	126.70	144.22	0.03	141.78	2520.93	300.39	176.09	6.36	176.87	1684.97	81.23
401.60	285.27	126.70	144.22	0.00	141.80	2520.59	300.39	176.09	6.99	176.86	1685.43	81.23
603.20	285.27	126.70	144.22	0.00	141.80	2520.76	300.39	176.09	7.67	176.85	1686.36	81.23
647.00	285.27	126.70	144.22	0.00	141.80	2520.59	300.39	176.09	7.75	176.85	1686.59	81.23

1. Data from spreadsheet %remain.xls, sheet run6_TOT (Ref. 16).
2. Time in thousands of years.
3. Hf is modeled as Zr in EQ6 runs (see assumption 3.16)
4. As calculated by EQ6, which currently has no provision for radioactive decay.
5. All Pu taken as ^{239}Pu and decayed to produce ^{235}U (see assumption 3.17).
6. At 7.85×10^3 years, ~6.7 kg Gd is in solution; this Gd later reprecipitates.

5.3.4 Comparison of Aqueous Gd Behavior in Run 6 and Run 7

Previous studies suggest pH may exert a strong control on Gd loss from the WP (Ref. 18; Ref. 24; Ref. 25). In the current study, the pH minima and peak Gd aqueous concentrations occur over scales of 10^3 to 10^4 years, and are not well-resolved in figures 5.3.1-2, 5.3.2-2 and 5.3.3-2. This section provides more detailed plots of the covariation among Gd concentrations, pH and amounts of certain solids in the system, for run 6 and run 7. These two runs had comparatively high total Gd loss (13.2% and 12.2%, respectively). These two runs also employed different 2nd-stage drip rates (0.0015 m^3/yr for run 6, and 0.015 m^3/yr for run 7); as shown in Table 5.3-2, all the runs with the lower drip rate had Gd maxima before 15 thousand years, and the two runs with the higher drip rate had Gd maxima after 50 thousand years. The system pH is plotted for reference in each of figures 5.3.4-1 through 5.3.4-6 below.

Figures 5.3.4-1 through 5.3.4-3 show the behavior of run 6. Figure 5.3.4-1 shows that the pH reaches a minimum at ~5.6×10^3 years, then rises steadily; the aqueous Gd also continues to rise until ~8.2×10^3 years, and then falls sharply. As shown in figure 5.3.4-2, the aqueous Gd drop

coincides with the precipitation of $GdOHCO_3$. The solubility product for this phase is $K_{sp} = [Gd^{3+}][OH^-][CO_3^{=}]$, so it is not remarkable that precipitation follows after an increase in both pH and aqueous Gd concentrations. Figure 5.3.4-3 shows that the initial drop in pH corresponds to the consumption of 304L stainless steel.

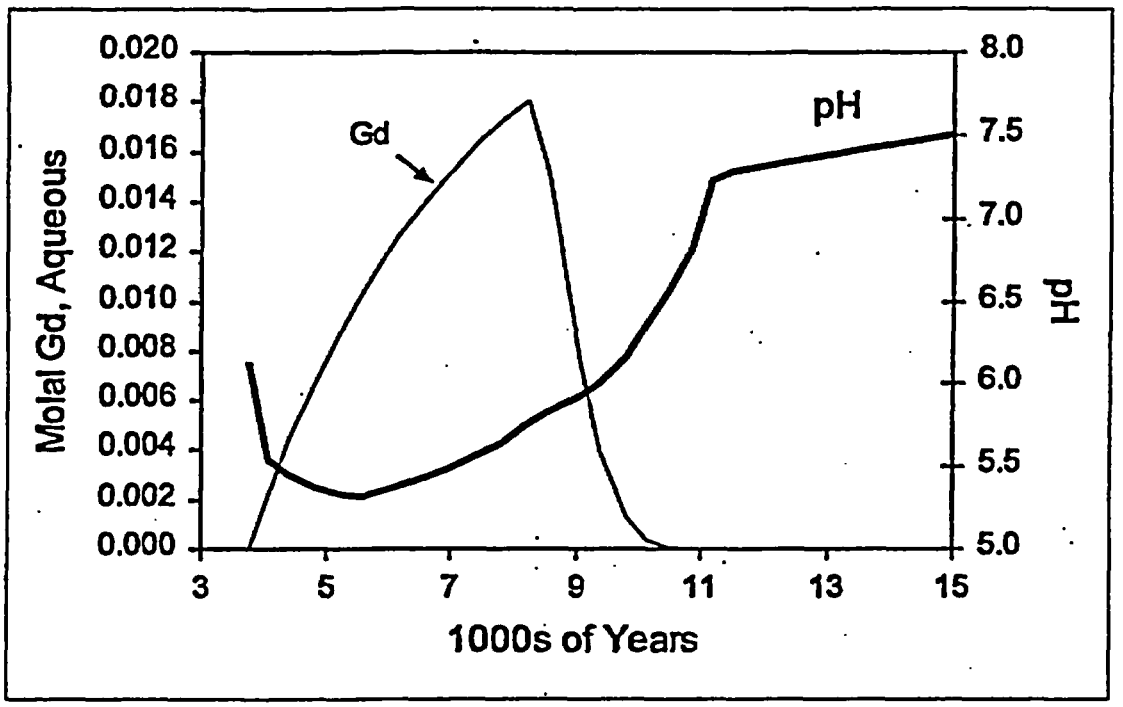

Figure 5.3.4-1. Aqueous Gd Concentrations and pH for Run 6

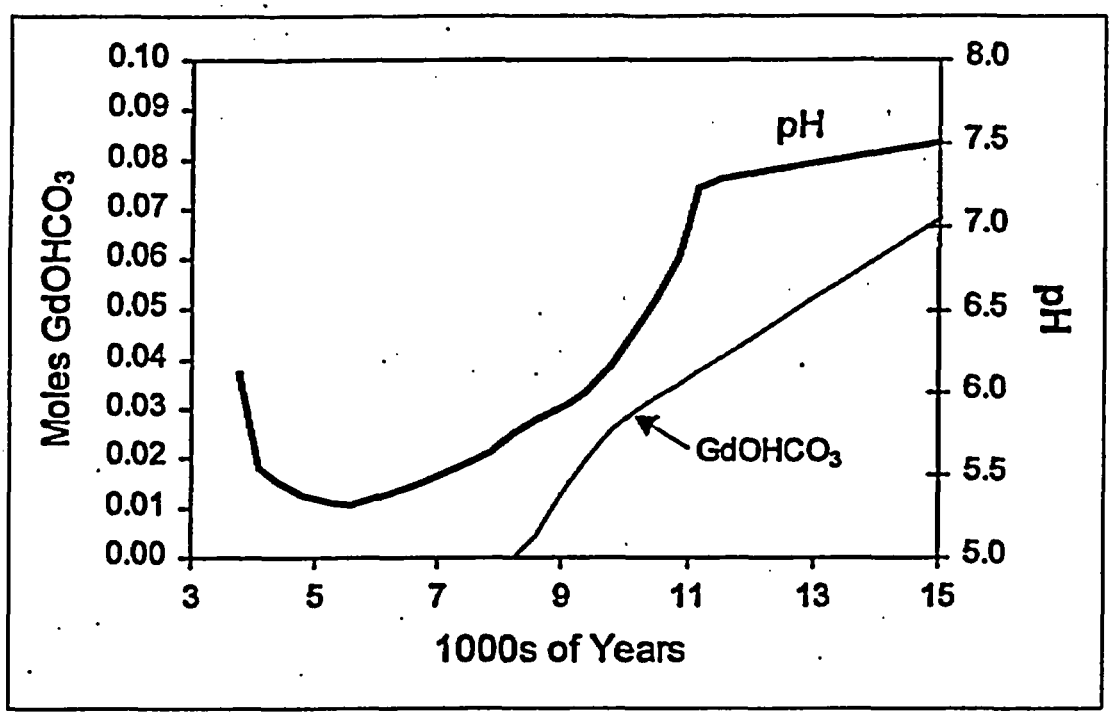

Figure 5.3.4-2. Moles Solid $GdOHCO_3$ (per Liter Initial Void Space) and pH for Run 6

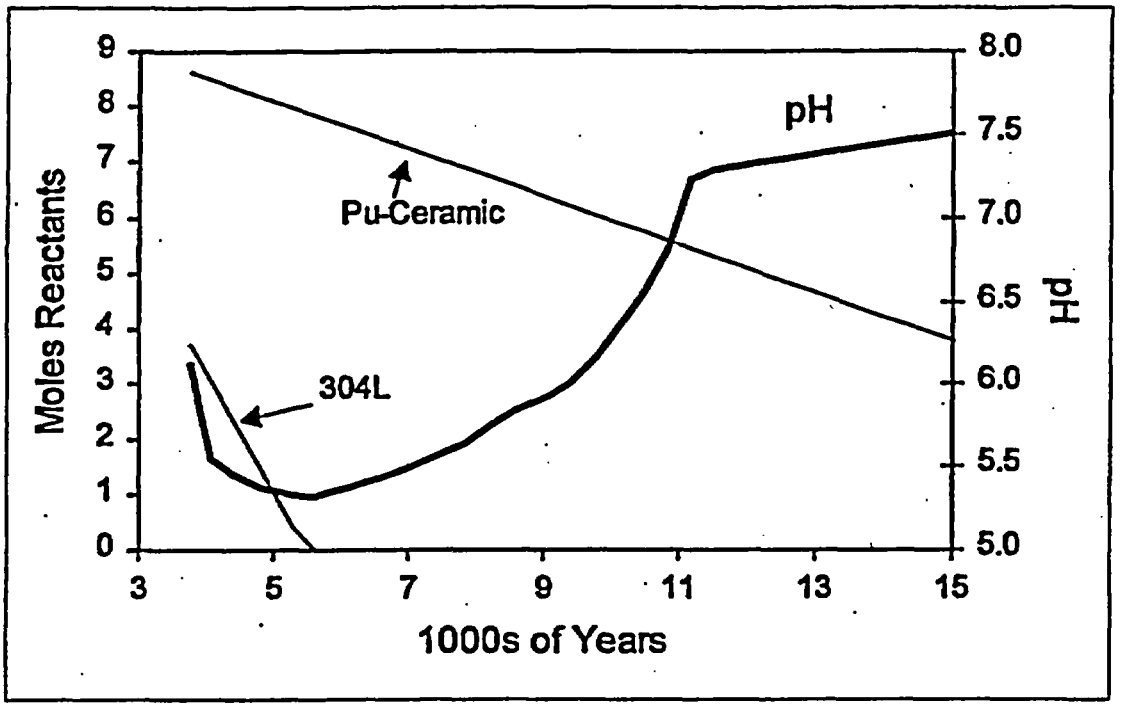

Figure 5.3.4-3. Moles Reactants (WP Materials, per Liter Initial Void Space) Remaining in the Package, and pH, for Run 6

Figures 5.3.4-4 through 5.3.4-6 show the behavior of run 7. The aqueous Gd peak for run 7 (figure 5.3.4-4) is much broader and flatter than the corresponding peak for run 6, and corresponds to broad, flat pH minimum. Figures 5.3.4-5 and 5.3.4-6 show two events that appear to control the beginning and end of the pH minimum, respectively. The disappearance of dolomite ($CaMg(CO_3)_2$) allows the pH to drop below 6 (figure 5.3.4-5); and the final consumption of all the 304L stainless steel (figure 5.3.4-6) allows the pH to rise once again. (Ordered dolomite forms slowly in nature, but the exact identity of the solid alkaline carbonate is probably not significant. If dolomite were prevented from forming in the EQ6 run, with the suppress option, another alkaline carbonate, such as calcite, would take its place, with a similar buffering effect on pH.) Some solid $GdOHCO_3$ is lost during the pH minimum, but most remains.

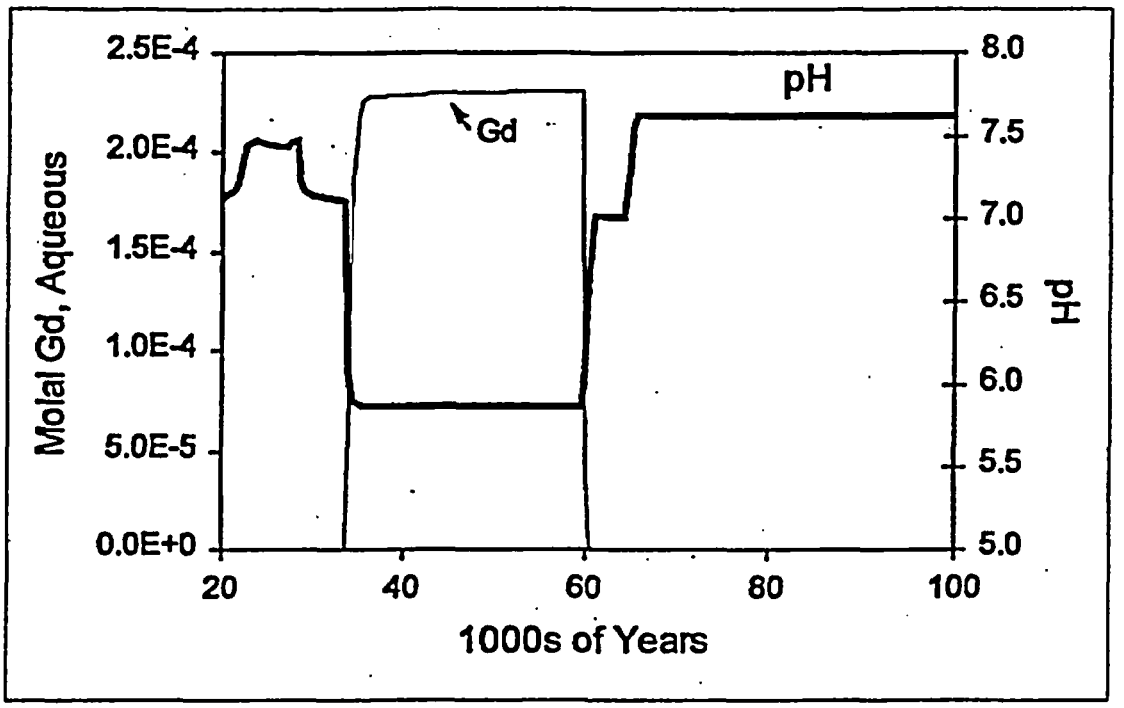

Figure 5.3.4-4. Aqueous Gd Concentrations and pH for Run 7

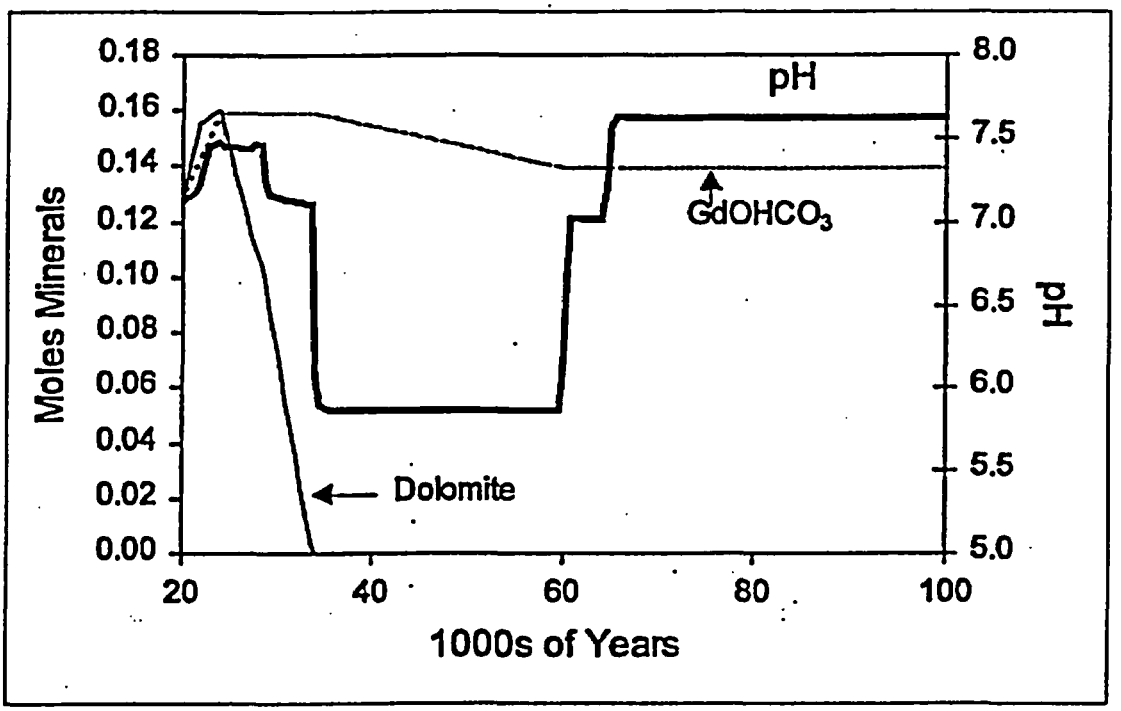

Figure 5.3.4-5. Moles Solid Dolomite and $GdOHCO_3$ (per Liter Initial Void Space) and pH for Run 7

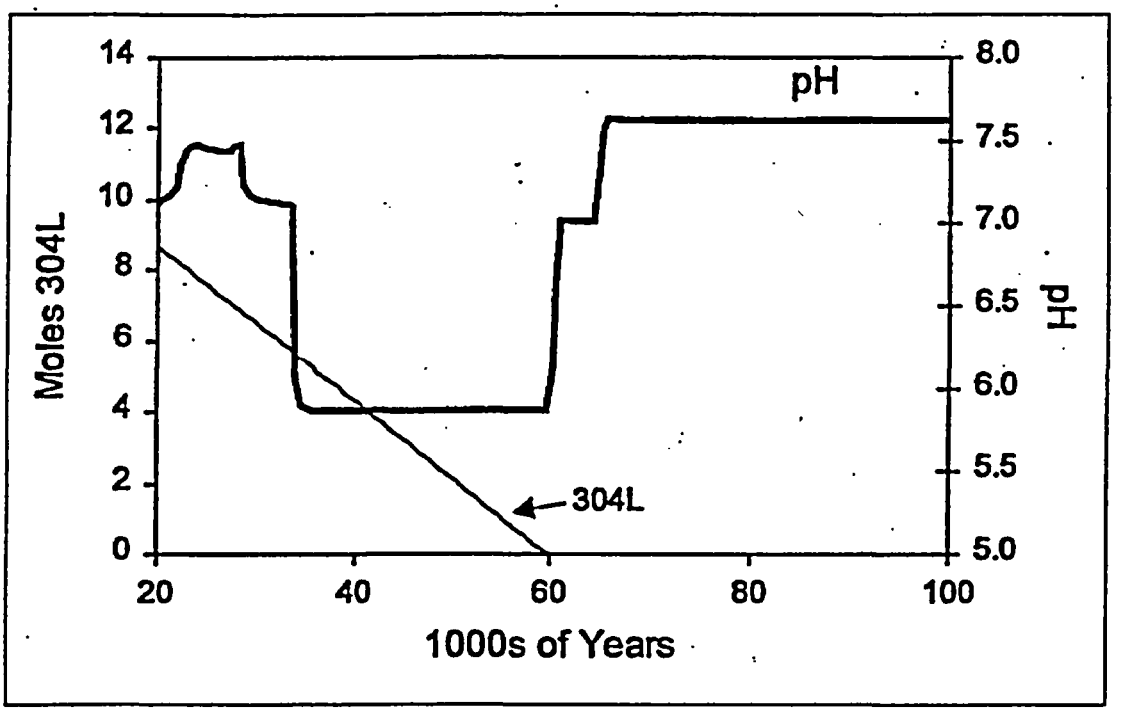

Figure 5.3.4- 6. Moles Reactants (WP Materials, per Liter Initial Void Space) Remaining in the Package, and pH, for Run 7

5.3.5 Sensitivity to Metastability of Goethite

The calculations summarized in Table 5.3-1 invariably predict the formation of hematite (Fe_2O_3), which is thermodynamically more stable than goethite (FeOOH). However, goethite is a common constituent of rust, and may persist metastably. To determine the sensitivity of the calculations to the formation of hematite *vs.* goethite, run 6 (Table 5.3-1) was repeated with hematite suppressed, so that goethite precipitated instead. The resulting calculation (archived as run Cerd3W0_0015I_CO2_LO_nH.6i in Ref. 16) predicted 14.9% Gd loss, compared with 13.2% loss when hematite was allowed to form. The peak Gd concentration was 2.1×10^{-2} molal, *vs.* 1.8×10^{-2} molal when hematite was allowed to form. These differences are small, and are probably within the uncertainty of the calculations.

6. Results

A principal purpose of the calculations was to assess chemical conditions that could lead to a loss of neutron absorbers (particularly Gd and Hf) from the package, as well as conditions that would allow the fissile materials to remain. A base case (run 1) was established in which all materials were allowed to degrade together, albeit at differing rates. This first scenario gave little opportunity for Gd loss, as the alkalinity of the HLW glass prevented the system from achieving the low pH conducive to Gd dissolution. To increase conservatism, a set of 7 two-stage

scenarios (runs 2-8 in Table 5.3.1) was developed, in which the Pu-ceramic was kept intact and unexposed to water until the HLW had degraded. In the two-stage scenarios, pH as low as 5.25 was achieved due to degradation of the steels in the package. However, Gd loss was still moderated by the formation of solid alkaline carbonates, the limited amounts of acid-producing steels in the package, the alkalinity of the Pu-ceramic waste itself, and by the inherent alkalinity of the J-13 water. In all scenarios, essentially all the Hf (represented by Zr, per assumption 3.16) and all the Pu are retained, while up to 54% of the U is lost.

The calculations suggest that nearly all the initial Fe, Mn, Al and Si in the packages will be retained, and from 72 to 100% of the Ni will be retained. A few principal minerals will dominate the bulk volume of the degraded WP, and will account for the retention of Fe, Mn, Al, Si and Ni. The calculations predict that smectite clay (an Fe-rich nontronite) will overwhelmingly constitute the bulk of the volume, followed by hematite, pyrolusite and Ni_2SiO_4. The original Na will be almost completely lost over the course of ~10^5 years.

7. References

1 National Research Council of the National Academy of Sciences 1995. *Technical Bases for Yucca Mountain Standards.* Washington, DC: National Academy Press. TIC Cat. No. 104273

2 Harrar, J. E., Carley, J. F., Isherwood, W. F., and Raber, E. 1990. *Report of the Committee to Review the Use of J-13 Well Water in Nevada Nuclear Waste Storage Investigations.* UCID-21867. Livermore, California: Lawrence Livermore National Laboratory. MOL.19980416.0660

3 Wolery, Thomas J. 1992a. *EQ3/6, A Software Package for Geochemical Modeling of Aqueous Systems: Package Overview and Installation Guide (Version 7.0).* UCRL-MA-110662 PT I. Livermore, California: Lawrence Livermore National Laboratory. TIC Cat. No. 205087

4 Daveler, Stephanie A., and Wolery, Thomas J. 1992. *EQPT, A Data File Preprocessor for the EQ3/6 Software Package: User's Guide, and Related Documentation (Version 7.0).* UCRL-MA-110662 PT II. Livermore, California: Lawrence Livermore National Laboratory. TIC Cat. No. 205240

5 Wolery, Thomas J. 1992b. *EQ3NR, A Computer Program for Geochemical Aqueous Speciation-Solubility Calculations: Theoretical Manual, User's Guide, and Related Documentation (Version 7.0).* UCRL-MA-110662 PT III. Livermore, California: Lawrence Livermore National Laboratory. TIC Cat. No. 205154

6 Wolery, Thomas J., and Daveler, Stephanie A. 1992. EQ6, A Computer Program for Reaction Path Modeling of Aqueous Geochemical Systems: Theoretical Manual, User's Guide, and Related Documentation (Version 7.0). UCRL-MA-110662 PT IV. Livermore, California: Lawrence Livermore National Laboratory. TIC Cat. No. 205002

7 Spahiu, K., and Bruno, J. 1995. *A selected thermodynamic database for REE to be used in HLNW performance assessment exercises.* SKB Technical Report 95-35. Stockholm, Sweden: Swedish Nuclear Fuel and Waste Management Co. TIC Cat. No. 225493

8 Civilian Radioactive Waste Management System (CRWMS) Management & Operating Contractor (M&O) 1997a. *Evaluation of the Potential for Deposition of Uranium/Plutonium from Repository Waste Packages.* BBA000000-01717-0200-00050 REV 00. Las Vegas, Nevada. MOL.19980216.0001

9 Yang, I. C., Rattray, G. W., and Yu, P. 1996. *Interpretation of Chemical and Isotopic Data from Boreholes in the Unsaturated-Zone at Yucca Mountain, Nevada.* WRIR 96-4058. Denver, Colorado: U. S. Geological Survey. MOL.19980528.0216

10 Weast, R. C., ed. 1977. *CRC Handbook of Chemistry and Physics, 58th Ed.* Cleveland, Ohio: CRC Press, Inc. NNA.19900827.0177

11 CRWMS M&O 1996a. *Second Waste Package Probabilistic Criticality Analysis: Generation and Evaluation of Internal Criticality Configurations (SCPB: N/A).* CSCI: BBA000000-01717-2200-00005 REV 00. Las Vegas, Nevada: P. Gottlieb. MOL19960924.0193

12 Latimer, W. M. 1952. *The Oxidation States of the Elements and Their Oxidation Potentials in Aqueous Solutions.* New York, New York: Prentice-Hall Inc. p 272. TIC Cat. No. 238748

13 CRWMS M&O 1997b. Bowman, G. N., and Carlisle, G. P. 1997. *Software Configuration Management.* QAP-SI-3, REV 2. Las Vegas, Nevada: MOL.19980205.0331

14 CRWMS M&O 1998a. *EQ3/6 Software Installation and Testing Report for Pentium Based Personal Computers (PCs).* CSCI: LLYMP9602100. Las Vegas, Nevada: H. Stockman. MOL.19980813.0191

15 CRWMS M&O 1997b. *Geochemical and Physical Analysis of Degradation Modes of HEU SNF in a Codisposal Waste Package with HLW Canisters.* CSCI: BBA000000-01717-0200-00059 REV 01. Las Vegas, Nevada: P. Cloke. MOL.19980624.0325

16 CRWMS M&O 1998b. *Electronic Media for BBA000000-01717-0210-00018 REV 00.* QIC-80 DT 350 Tape. MOL.19980831.0169

17 CRWMS M&O 1998c. *EQ6 Calculations for Chemical Degradation of PWR and MOX Spent Fuel Waste Packages.* CSCI: BBA000000-01717-0210-00009 REV 00B. Las Vegas, Nevada: P. Cloke. MOL.19980701.0483

18 CRWMS M&O 1998d. Evaluation of Codisposal Viability for Aluminum-Clad DOE-Owned Spent Fuel: Phase II. Degraded Codisposal Waste Package Internal Criticality. CSCI: BBA000000-01717-5705-00017 REV 01. Las Vegas, Nevada: J. McClure. MOL.19980616.0098

19 Plutonium Immobilization Project. 1998. *Data for Yucca Mountain Total Systems Performance Assessment,* Rev. 1, PIP Milestone Report, Milestone 2.b.b., PIP 98-012. Livermore, California: Lawrence Livermore National Laboratory. MOL.19980818.0349

20 CRWMS M&O 1996b. *DHLW Glass Waste Package Criticality Analysis (SCPB: N/A).* BBAC00000-01717-0200-00001 REV 00. Las Vegas, Nevada: J. Davis. MOL.19960919.0237

21 CRWMS M&O 1996. *Material Compositions and Number Densities For Neutronics Calculations (SCPB: N/A).* BBA000000-01717-0200-00002 REV 00. Las Vegas, Nevada. MOL.19960624.0023

22 CRWMS M&O 1998. *Unsaturated-Zone Flow: Preliminary Draft Section 2.3 of TSPA-VA Document.* B00000000-01717-2200-00201. Las Vegas, Nevada: C.K. Ho. MOL.19980428.0202

23 CRWMS M&O 1998d. *Controlled Design Assumptions Document.* B00000000-01717-4600-00032 REV 05. Las Vegas, Nevada: W. Clem. MOL.19980804.0481

24 CRWMS M&O 1996c. *Status Report on Degraded Mode Criticality Analysis of Immobilized Plutonium Waste Forms in a Geologic Repository.* A00000000-01717-5705-00013 REV 00. Vienna, Virginia: TRW Environmental Safety Systems Inc. MOL.19970625.0445

25 CRWMS M&O 1997c. Degraded Mode Criticality Analysis of Immobilized Plutonium Waste Forms in a Geologic Repository. A00000000-01717-5705-00014 REV 01. Las Vegas, Nevada: P. Gottlieb. MOL.19980422.0911

26 Roberts, W. L., Jr., Rapp, G. R., Jr., and Weber, J. 1974. *Encyclopedia of Minerals.* New York, New York: van Nostrand Reinhold Co. TIC Cat. No. 238571

27 Walker, E. Wm., Parrington, J. R. and Feiner, F. 1989. *Nuclides and Isotopes, 14th ed., Chart of the Nuclides.* San Jose, California: General Electric Co. TIC Cat. No. 201637

28 CRWMS M&O 1995. *Total System Performance Assessment -- 1995. An Evaluation of the Potential Yucca Mountain Repository.* B00000000-01717-2200-00136 REV 01. Las Vegas, Nevada. MOL.19960724.0188

8. Attachments

Attachment I. C programs To Implement Flushing Mode (27 pages)

Attachment II. Files Archived on QIC-80 DT-350 Tapes (3 pages)

Attachment I. C programs To Implement Flushing Mode

```
/* This file is NXTI_BAT.c, PC version of nxtinput.bat. */
/*********************************************************/
/*  Compile with MSVC++ 4.x or later, as Console application --HWStockman */
/* NOTE eq6.exe is copied to eq6_dum.exe in the local directory. */

#include <stdio.h>
/* #include <conio.h> */
#include <stdlib.h>
#include <process.h>
#include <ctype.h>
#include <time.h>
#include <string.h>
#include <io.h>
#include <math.h>
#include <dos.h>

/* Reads the number of iterations as the sole command-line arg.*/
int main(int argc, char *argv[])
{
#define NBUF 128
#define MAX_COUNT 1000
FILE *sfilePtr;
int count=1,countMax=200;  /* countMax from command line...*/
char buf[NBUF];

if(argc>1){
     sscanf(argv[1],"%d",&countMax);
     if(countMax < 1) countMax=1;
     else if(countMax>MAX_COUNT) countMax = MAX_COUNT;
     }

for(; count<=countMax; printf("nxti_bat count = %d\n",++count)){
     system("move bldinput.out input");
     spawnl(_P_WAIT,"eq6_dum.exe","eq6_dum.exe",NULL);
     system("copy /b allin+input allin");
     system("copy /b allpick+pickup allpick");
     system("copy /b allout+output allout");
     system("copy /b alltab+tab alltab");
     spawnl(_P_WAIT,"nxtinput.exe","nxtinput.exe",NULL);

     sfilePtr = fopen("sfile","r");
     if(sfilePtr){
          fgets(buf,NBUF-1,sfilePtr);
          if(buf[0]!='g' || buf[1]!='o'){  /* may want strstr() instead */
               fclose(sfilePtr); puts(buf); puts("abort from nxti_bat.exe");
               exit(3);
               }
          fclose(sfilePtr);
          }
     else {
          puts("Couldn't open sfile, exiting from nxti_bat loop"); exit(3);
          }
     }    /* END for() */
spawnl(_P_WAIT,"rename.exe","rename.exe",NULL);
exit(0);
#undef NBUF
#undef MAX_COUNT
}       /* END main() for nxti_bat.c */
```

```
/* This file is nxtinput.c */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
double getfloat(char*,int,int),gettobar(char*,int);
void setup(),bldpick(),infromstd(),infromlast(),
   convert(double,double,FILE*,FILE*),
   strinsert(char*,char*,int,int);
int locaterw(char*,FILE*,FILE*),locatero(char*,FILE*),
   locate2(char*,char*,FILE*),tobar(char*,int),findinline(char*),
   puttobar(char*,char*,int),locate1of2(char*,char*,FILE*);
int finished=0;
double mash2oend,duration;
char dummy[100],tdummy[100];
char froot[20],cname[20];
FILE *fout,*fpick,*fotemp,*fptemp,*fstd,*foutout,*finin,
   *fttemp,*fs,*fin,*fname;

void main()
{int i,j,k,flag;
fs=fopen("sfile","w");
fprintf(fs,"go\n");
flag=1;
fout=fopen("bldinput.out","w");/*file to be moved to input*/
infromlast();}

void infromlast()
{int i,j,k,dot;
char tempstr[30],carbstr[7],*cp,sstring[60],tempstr2[20];
double dmj13,msh2o,msh2ox,xx,yy,moles,dmoles,delmaxtime;
fin=fopen("bldinput.in","r");/*input parameters special to this case*/
fstd=fopen("input","r");/*template from last input file*/
fpick=fopen("pickup","r");/*old pickup file; extract section to bldinput.out*/
foutout=fopen("output","r");/*from last iteration to new input*/
finin=fopen("input","r");/*from last iteration to new input*/
fotemp=fopen("otemp","w");/*store intermediate segments from output*/
fptemp=fopen("ptemp","w");/*store intermediate segments from pickup*/
fgets(dummy,90,fin); /*readthrough labels*/
fscanf(fin,"%s %s %s %lf\n",
     tempstr,tempstr,tempstr,&delmaxtime);/*only 1 param used this prgrm*/
locatero("                    Moles of solvent H2O",foutout);
msh2ox=getfloat(dummy,44,12);  /*optional  parameter from the first block*/
foutout=freopen("output","r",foutout);
strcpy(sstring,"                         Reaction progress");
if(locatero(sstring,foutout)==-1)  /*find output block of interest*/
  {printf("bad output file\n");
  exit(0);}
fputs(dummy,fotemp);         /*and write it to temporary*/
while(fgets(dummy,90,foutout)!=NULL)
  {fputs(dummy,fotemp);
  if(strncmp(dummy,sstring,strlen(sstring))==0)
     {fotemp=freopen("otemp","w",fotemp);
     fputs(dummy,fotemp);}}
fotemp=freopen("otemp","r",fotemp);/* re-open to find water*/
strcpy(sstring,"Mass of solvent H2O");
if(locatero(sstring,fotemp)!=1)
  {printf("Can't find ending water\n");
  fs=fopen("sfile","w");
  fprintf(fs,"cant find ending water");
  exit(0);}/*ending water*/
mash2oend=getfloat(dummy,44,12);
fotemp=freopen("otemp","r",fotemp);/*now reopen for use*/
```

```
if(locatero("c pickup file",fpick)==-1)   /*start copying here*/
  {printf("bad pickup file\n");
  exit(0);}
fputs(dummy,fptemp);
for(i=0;i<2;i++)  /*readwrite through first "|EQ"*/
  {fgets(dummy,90,fpick);
  fputs(dummy,fptemp);}
while(fgets(dummy,90,fpick)!=NULL) /*pickup to ptemp*/
  {fputs(dummy,fptemp);
  if(strncmp(dummy,"|EQ",3)==0) /*read through without copying*/
    while(fgets(dummy,90,fpick)!=NULL)
      if(strncmp(dummy,"c pickup file",strlen("c pickup file"))==0)
        {fptemp=freopen("ptemp","w",fptemp);/*start copying over again*/
        fputs(dummy,fptemp);
        for(i=0;i<2;i++)
          {fgets(dummy,90,fpick);
          fputs(dummy,fptemp);}
        break;}}
fptemp=freopen("ptemp","r",fptemp); /*now reopen for use*/
if(locaterw("|EQ",fstd,fout)==-1)
  {printf("bad input file\n");
  exit(0);}
i=0;   /*the next 10 lines increment the case number in the first line*/
while((i<strlen(dummy))&&(dummy[i]!='.'))i++;
dot=i;
i=0;
while((dummy[dot-i-1]<='9')&&(dummy[dot-i-1]>='0'))i++;
for(j=0;j<i;j++)tempstr[j]=dummy[dot-i+j];
tempstr[i]='\0';
k=atoi(tempstr);
sprintf(tempstr,"%u%s",k+1,".6i");
strinsert(dummy,tempstr,dot-i,strlen(tempstr));
fputs(dummy,fout);
i=dot;
while((dummy[i]!=' ')&&(dummy[i]!='='))i--; /*now get root filename*/
strncpy(tempstr,dummy+i+1,dot-i-1);
tempstr[dot-i-1]='\0';
fname=fopen("rootname","w");  /*and write the root filename*/
fprintf(fname,"%s\n",tempstr);
fclose(fname);
fgets(dummy,90,fotemp);/*get ending value of zi from first line*/
xx=getfloat(dummy,48,22);
if(locaterw("| starting value of zi",fstd,fout)==-1)
  {printf("can't find starting zi in input file\n");
  exit(0);}
sprintf(tempstr,"%15.8lE",xx);
i=tobar(dummy,1);
strinsert(dummy,tempstr,i+1,strlen(tempstr));
fputs(dummy,fout); /*and put into input*/
fgets(dummy,90,fstd);
fputs(dummy,fout);
fgets(tdummy,90,fstd);/*this takes us to entry for starting time*/
if(locatero("       Time increased from",fotemp)==-1)
  {printf("can't find last ending time in output\n");
  exit(0);}
fgets(dummy,90,fotemp);/*this line will have end time of last run*/
xx=getfloat(dummy,31,12);
sprintf(tempstr,"%11.5lE",xx);
i=tobar(tdummy,1);
if(i==-1)
  {printf("cant find slot for starttime\n");
  exit(0);}
strinsert(tdummy,tempstr,i+1,strlen(tempstr));
```

```
i=tobar(tdummy,i+1);
i=tobar(tdummy,i+1);
if(i==-1)
  {fs=freopen("sfile","w",fs);
  printf("cant find slot for maxtime\n");
  exit(0);}
/*yy=gettobar(tdummy,i+1); */
sprintf(tempstr,"%12.4lE",xx+delmaxtime);
strinsert(tdummy,tempstr,i+1,strlen(tempstr));
fputs(tdummy,fout); /*and put into input*/
fotemp=freopen("otemp","r",fotemp);/*last read was beyond current interest*/
if(locatero("     Reactant                    Moles     Delta moles",fotemp)==-1)
  {printf("cant find values for reactants in the output file\n");
  exit(0);}
fgets(tdummy,90,fotemp);
fgets(tdummy,90,fotemp);/*get to first reactant in otemp*/
while((finished==0)&&(strncmp(tdummy,"\n",1)!=0))/*loop to do all reactants*/
  {moles=getfloat(tdummy,29,10);
  dmoles=getfloat(tdummy,42,10);
  locaterw("| moles remaining",fstd,fout);/*next reactant*/
  sprintf(tempstr,"%10.4lE",moles);
  strinsert(dummy,tempstr,20,strlen(tempstr));
  if(strncmp(tdummy,"  J-13 water",12)!=0)
    {sprintf(tempstr2,"%10.4lE",dmoles);
    strinsert(dummy,tempstr2,58,strlen(tempstr2));}
  else
    {dmj13=dmoles;
    finished=1;}                          /*Water is the last reactant*/
  fputs(dummy,fout);
  fgets(tdummy,90,fotemp);}
if(locatero("                    Moles of solvent H2O",fotemp)==-1)
  {fprintf(fs,"cant find moles water in output\n");
  exit(0);}
msh2o=getfloat(dummy,44,12);
k=locatero(" ---  The reaction path has terminated normally",fotemp);
if(k==-1) fputs("abnormal reaction path termination\n",fs);
fotemp=freopen("otemp","r",fotemp);/*back to the top again*/
if((k=locate2(" CO3--"," HCO3-",fotemp))==1) strcpy(carbstr,"| CO3--");
else if (k==2) strcpy(carbstr,"| HCO3-");
fttemp=fopen("ttemp","w");/*will later attach to input*/
if(locate1of2("| CO3--","| HCO3-",fptemp)==-1)/*also copies ptemp to ttemp*/
  {fprintf(fs,"cant find line to insert carbonates in pickup\n");
  exit(0);}
strinsert(dummy,carbstr,0,strlen(carbstr));
fputs(dummy,fttemp);
while(fgets(dummy,90,fptemp)!=NULL)fputs(dummy,fttemp);/*rest of ptemp to ttemp*/
fttemp=freopen("ttemp","r",fttemp);
if(locaterw("c pickup file",fstd,fout)==-1)/*transfer the relevant remainder of the template*/
  {fprintf(fs,"cant find start for pickup info\n");
  exit(0);}
convert(msh2o,dmj13/3,fstd,fttemp);}

int locate1of2(char sstring1[50],char sstring2[50],FILE *fp)
{int found1=0,found2=0;
while((found1==0)&&(found2==0))
  {if(fgets(dummy,90,fp)==NULL)return -1;
  if(found1==0)
     if(strncmp(dummy,sstring1,strlen(sstring1))==0)
       found1=1;
  if(found2==0)
     if(strncmp(dummy,sstring2,strlen(sstring2))==0)
       found2=1;
  if((found1==0)&&(found2==0))fputs(dummy,fttemp);}
```

```
if((found1==0)&&(found2==0))return -1;
else return 0;}

void strinsert(char inline[90],char insert[90],int start,int len)
{int i;
for(i=0;i<len;i++) inline[start+i]=insert[i];}

int locate2(char sstring1[50],char sstring2[50],FILE *fp)
{int i,found1=0,found2=0;
double x1=0,x2=0;
char buffer[100];
while((fgets(dummy,90,fp)!=NULL)&&((found1==0)||(found2==0)))
  {strcpy(buffer,dummy);
  if(found1==0)
    if(strncmp(dummy,sstring1,strlen(sstring1))==0)
      {found1=1;
      x1=getfloat(dummy,28,12);}
  if(found2==0)
    if(strncmp(dummy,sstring2,strlen(sstring2))==0)
      {found2=1;
      x2=getfloat(dummy,28,12);}}
if(x1<x2) return 2;
else return 1;}

int locatero(char sstring[60],FILE *fp)/*read only*/
{while(fgets(dummy,90,fp)!=NULL)
  if(findinline(sstring)==1)return 1;
return -1;}

int locaterw(char sstring[60],FILE *fpin,FILE *fpout)/*read&write*/
{while(fgets(dummy,90,fpin)!=NULL)
  {if(strncmp(dummy,sstring,strlen(sstring))==0)return 1;
  fputs(dummy,fpout);}
return -1;}

void convert(double x,double z,FILE *fins,FILE *finp)
{int i,count=0;
double u,v,w,r;
char buffer[100],temp[50],temp2[50];
r=x/(x+z);
if(mash2oend*r>1)     /* to bring the free water back to 1 kg */
  {r=1/mash2oend;
   printf("converted to %f\n",r);}
if(locaterw("| elements, moles",finp,fout)==-1)/*readwrite to this point*/
  {printf("cant locate place to put new values of reagents in input\n");
  exit(0);}
fputs(dummy,fout);
fgets(buffer,90,finp);
fputs(buffer,fout);
fgets(buffer,98,finp);
while(strncmp(buffer,"|-------",8)!=0)
  {w=getfloat(buffer,55,21);
  v=w*r;
  u=getfloat(buffer,30,21)-w*(1-r);
  sprintf(temp,"%22.15lE",u);
  strinsert(buffer,temp,29,strlen(temp));
  sprintf(temp,"%22.15lE",v);
  strinsert(buffer,temp,54,strlen(temp));
  fputs(buffer,fout);
  fgets(buffer,90,finp);
  count++;}
fputs(buffer,fout);
for(i=0;i<2;i++)
```

```
  {fgets(buffer,100,finp); /*readthrough to species table*/
  fputs(buffer,fout);}
for(i=0;i<count;i++)
  {fgets(buffer,100,finp);
  w=getfloat(buffer,56,22);
  sprintf(temp,"%+20.15lE",w+log10(r));
  strinsert(buffer,temp,56,strlen(temp));
  fputs(buffer,fout);}
while(fgets(buffer,100,finp)!=NULL) fputs(buffer,fout);}

double getfloat(string,start,len)
char string[100];
int start,len;
{char temp[30];
strncpy(temp,string+start,len);
temp[len]='\0';
return atof(temp);}

double gettobar(char line[100],int start)
{int i;
char temp[30];
i=start;
while((i<strlen(line))&&(line[i]!='|'))
  {temp[i-start]=line[i];
  i++;}
temp[i]='\0';
if(line[i]!='|')return -1;
return atof(temp);}

int puttobar(char line[100],char string[30],int start)
{int i,k;
i=start;
k=strlen(string);
while((i<strlen(line))&&(line[i]!='|')&&(i-start<k))
  {line[i]=string[i-start];
  i++;}
if(line[i]=='|')return i;
else return -1;}

int tobar(char line[100],int start)
{int i;
i=start;
while((i<strlen(line))&&(line[i]!='|'))i++;
if(line[i]=='|')return i;
else return -1;}

int findinline(char sstring[50])
{int i=0;
while(i<100)
  {if(strncmp(dummy+i,sstring,strlen(sstring))==0) return 1;
  else i++;}
return 0;}
```

```
/* ALLN_BAT.C, Win95 Console program to perform functions of UNIX allpost.bat. */
/*************************************************************************/
/* To be compiled with Microsoft VC++ 4.x or later.  Modified from allp_bat.c, */
/* to use newpost.exe (newpost.c) replacement for postproc.exe (postproc.exe). */
/* Adds simple convenience features not in original UNIX allpost.bat:    */
/*  Command line arguments: see See SetFlags() for command-line list and */
/*    implementation status, of use -h command-line arg to see status.   */
/*    The -mfilename argument passes the minerals list in filename to    */
/*    newpost. Can change inner and outer loop counts, and automatically */
/*    rename the final files to reflect rootname in bldinput.in.         */
/*  Distinctive header: alln_bat writes a header to the tops             */
/*    of the *.allpost, *.lastpost files, with run info, dates, etc.     */
/*  Last input file: is saved as last_inp to make restarts simpler. -HWS */
/*************************************************************************/
/* HW Stockman */
#include <direct.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <process.h>
#include <ctype.h>
#include <time.h>
#include <string.h>
#include <io.h>
#include <math.h>
#include <dos.h>

#define MAX_INNER 1000
#define MIN_INNER 1
#define MAX_OUTER 200
#define MIN_OUTER 1
#define MAX_RENAME 2 /* for now 0 is no rename, non-zero to auto-rename */
#define MIN_RENAME 0
#define MAX_RUNLASTPOST 2 /* for now 0 is don't run lastpost, non-zero run  */
#define MIN_RUNLASTPOST 0
#define FLAG_HELP   -1

/* NOTE the -e feature (elemName) is not implemented as of 8-12-98 */
struct flags_struct {
     int inner; /* inner loop iterations, -i ## */
     int use_inner;
     int outer; /* outer loop iterations, -o ## */
     int use_outer;
     int rename; /* auto-rename files, -r # */
     int use_rename;
     int runLastpost; /* auto-run lastpost files, -l # */
     int use_runLastpost;
     char *elemName;  /* list of elements to postproP, -e filename.txt */
     int use_elemName;
     char *minName;  /* list of minerals to postproc, -m filename.txt */
     int use_minName;
     } FLAG;
/*--------------------FUNCTION DECLARATIONS-------------------------*/
int Y_or_N(void);
int SetFlags(int argc, char *argv[], struct flags_struct *fp);
FILE *MakeHead(int *ierr, char *headName);
int FileOK(char *str);
int GetIDName(char *str, char *fname, char *token, int nstr);
```

```
/*-------------------------MAIN()-----------------------------------*/
int main(int argc, char *argv[])
{
#define NBUF 256
FILE *sfilePtr,*bldinput_inPtr, *head_fp;
int iCount=1,iCountMax=200;
int oCount=1,oCountMax=9, iTemp;
int i;
/* some internal flags */
int iRunLastpost=0, iRename=0;
int len, lenEnding, lenRen, lenSrc; /* length of string, ending */
/* Use sAllpost, sLastpost so we must change names here only */
/*  (i.e. may change ending to ".sum" later). */
char *sAllpost = ".allpost";
char *sLastpost = ".lastpost";
char *sAlltab = ".alltab";
char *sAllin = ".allin";
/* we use "move" rather than "ren" since latter will rename directories */
char *sRename = "move ";  /*  DOS rename command, followed by a space. */
char buf[NBUF],fname[NBUF]; /* Add filename fname for cleanup block */
char strDB[NBUF], strEQ6[NBUF], strEQLIB[NBUF];
char *headName = "head-_-_.-_-";

/*^^^^^^^^^^^^^^^^ Translate flags to local vars ^^^^^^^^^^^^^^*/
if(SetFlags(argc, argv, &FLAG)==FLAG_HELP){
     exit(0);  /* if we displayed help, quit */
     }
/* have structure set, use to set vars local to main() */
if(FLAG.use_inner) iCountMax = FLAG.inner;
if(FLAG.use_outer) oCountMax = FLAG.outer;
/* query user in case the inner count is less than outer */
if(iCountMax < oCountMax){
     printf("The inner count %d is less than the outer count %d; switch? (Y/N)\n",
       iCountMax,oCountMax);
     if(Y_or_N()=='Y'){
          iTemp = iCountMax;
          iCountMax = oCountMax; oCountMax = iTemp;
          }
     }
if(FLAG.use_rename) iRename = FLAG.rename;
if(FLAG.use_runLastpost) iRunLastpost = FLAG.runLastpost;

/*^^^^^^^^^^^^^^^^ END translating flags to local vars ^^^^^^^^^^^^^^*/


/* Start making the header for the allpost and lastpost files */
head_fp = MakeHead(&i, headName);
/* Write the command line and actual args used */
if(head_fp){
     fprintf(head_fp, "Command line= ");
     for(i=0; i<argc; i++){
          fprintf(head_fp, " %s", argv[i]);
          }
     /* write actual arguments */
     fprintf(head_fp,"\nactual inner loop count= %d, actual outer= %d\n", iCountMax,
oCountMax);
     }
/* Note we try to run rest of alln_bat, even if we couldn't make a header; */
/*  for case where header file is left in ambiguous state by system crash. */
/*  Below we will check that head_fp is non-NULL. */

/* Clean-up, to prevent concatenating old files. */
remove("allin"); remove("alltab"); remove("allin_"); remove("alltab_");
```

```
remove("allout"); remove("allpost");

/**************** MAIN OUTER LOOP ******************/
for(oCount=1; oCount<=oCountMax; printf("alln_bat oCount = %d\n",++oCount)){
     for(iCount=1; iCount<=iCountMax; printf("alln_bat iCount = %d\n",++iCount)){
          system("move bldinput.out input");
          /* The eq6.exe must be locally named eq6_dum.exe */
          spawnl(_P_WAIT,"eq6_dum.exe","eq6_dum.exe",NULL);
          system("copy /b allin_ +input allin_");
          system("copy /b allout+output allout");
          system("copy /b alltab_ + tab alltab_");
          /* 6-19-98 add extra test */
          if(FileOK("input")) system("copy /b input last_inp");
          spawnl(_P_WAIT,"nxtinput.exe","nxtinput.exe",NULL);

          sfilePtr = fopen("sfile","r");
          if(sfilePtr){
               fgets(buf,NBUF-1,sfilePtr);
               if(buf[0]!='g' || buf[1]!='o'){  /* may want strstr() instead */
                    fclose(sfilePtr); puts(buf); puts("abort from alln_bat inner");
                    exit(3);
                    }
               fclose(sfilePtr);
               }
          else {
               puts("Couldn't open sfile, bye-bye from alln_bat inner loop"); exit(3);
               }
          } /* END inner for() */
     system("del rootname");

     if(FLAG.use_minName && !FLAG.use_elemName){
          spawnl(_P_WAIT,"newpost.exe","newpost.exe","-m",FLAG.minName,NULL);
          }
     else spawnl(_P_WAIT,"newpost.exe","newpost.exe",NULL);
     system("copy /b allpost+postproc.out allpost");
     remove("allout"); /* ansi */
     /* Concat inner loop versions to outer loop versions */
     system("copy /b allin + allin_ allin");
     remove("allin_");
     system("copy /b alltab + alltab_ alltab");
     remove("alltab_");
     } /* END outer for() */
/*************** END MAIN OUTER LOOP ***************************/

/* Finish adding more header info to lastpost/allpost, to increase traceability: */
if(head_fp){
     i = GetIDName(strDB, "alltab", "data0.", NBUF);
     if(i) fprintf(head_fp,"database (from alltab)= %s\n", strDB);
     i = GetIDName(strEQ6, "alltab", "eq6.", NBUF);
     if(i) fprintf(head_fp,"eq6 revision (from alltab)= %s\n", strEQ6);
     i = GetIDName(strEQLIB, "alltab", "eqlib.", NBUF);
     if(i) fprintf(head_fp,"eqlib revision (from alltab)= %s\n", strEQLIB);
     fprintf(head_fp,"+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+\n");
     fclose(head_fp);
     }

/*:::::::::::::::::::: Cleanup block below ::::::::::::::::::::*/
/* activity in this block conditional on whether a flag is set or not... */
/* Find longest ending of name: */
lenEnding = strlen(sLastpost);
len = strlen(sAllpost); if(len>lenEnding) lenEnding=len;
len = strlen(sAlltab); if(len>lenEnding) lenEnding=len;
len = strlen(sAllin); if(len>lenEnding) lenEnding=len;
```

```
/* If no lastpost option on command line, query user */
if(!FLAG.use_runLastpost){
     printf("Run lastpost to condense the allpost file? (Y/N)\n");
     iRunLastpost = (Y_or_N()=='Y') ? 1 : 0;
     }

if(iRunLastpost){
     spawnl(_P_WAIT,"lastpost.exe","lastpost.exe",NULL);
     puts(".....ran lastpost....");
     }

/* Add a header to allout and lastpost.out . */
/* We control size of concat strings, so don't need strict length controls. */

strncpy(buf,"copy /b ",NBUF);
buf[NBUF-1] = 0;
strcat(buf, headName);
strcat(buf, " + allpost temp_-_-._-_");
system(buf);
system("move temp_-_-._-_ allpost");
/* On bizarre filename; Since we are in a multi-tasking environment, */
/*  and many processes may be making and reusing temporary files, we */
/*  use a temp file name that is unlikely to be used by other processes. */

if(iRunLastpost){
     strncpy(buf,"copy /b ",NBUF);
     buf[NBUF-1] = 0;
     strcat(buf, headName);
     strcat(buf, " + lastpost.out temp_-_-._-_");
     system(buf);
        system("move temp_-_-._-_ lastpost.out");
     }


/* fetch the desired file group name from bldinput.in */
fname[0] = 0;
bldinput_inPtr = fopen("bldinput.in","r");
if(bldinput_inPtr){
     fgets(buf,NBUF-1,bldinput_inPtr); /* dummy read of header */
     fgets(buf,NBUF-1,bldinput_inPtr);
     fclose(bldinput_inPtr);
     }
else {
     puts("Couln't open bldinput.in in alln_bat cleanup block, bye-bye"); exit(3);
     }
sscanf(buf,"%s",fname); fname[NBUF-1] = 0;
len = strlen(fname); /* locate end of rootname */

/*$$$$$$ Block for automatically renaming files from allpost, etc. to $$$$$$*/
/*$$$$$$  rootname.allpost, etc. where rootname is from bldinput.in . $$$$$$*/
if(len+lenEnding<NBUF-1){
     if(!FLAG.use_rename){
          printf("Do you want to rename:\n");
          if(iRunLastpost){
              printf(" lastpost.out to %s%s ,\n",fname,sLastpost);
              }
          printf("  allpost to %s%s ,\n",fname,sAllpost);
          printf("  alltab  to %s%s ,\n",fname,sAlltab);
          printf("  allin   to %s%s.?\n",fname,sAllin);
          iRename = (Y_or_N()=='Y') ? 1 : 0;
          }
```

```
        if(iRename){
             /* note system() returns 0 for failure! */
             len = strlen(fname);
             lenRen = strlen(sRename); strncpy(buf,sRename, NBUF-1);
             /* allpost */
             lenEnding = strlen(sAllpost);
             lenSrc = strlen("allpost "); /* note space */
             strncpy(buf+lenRen,"allpost ",NBUF-1-lenRen);
             strncpy(buf+lenRen+lenSrc,fname,NBUF-1-lenRen-lenSrc);
             strncpy(buf+lenRen+lenSrc+len,sAllpost, NBUF-1-lenRen-lenSrc-len);
             buf[NBUF-1]=0;
             system(buf);
             /* alltab */
             lenEnding = strlen(sAlltab);
             lenSrc = strlen("alltab "); /* note space */
             strncpy(buf+lenRen,"alltab ",NBUF-1-lenRen);
             strncpy(buf+lenRen+lenSrc,fname,NBUF-1-lenRen-lenSrc);
             strncpy(buf+lenRen+lenSrc+len,sAlltab, NBUF-1-lenRen-lenSrc-len);
             buf[NBUF-1]=0;
             system(buf);
             /* allin */
             lenEnding = strlen(sAllin);
             lenSrc = strlen("allin "); /* note space */
             strncpy(buf+lenRen,"allin ",NBUF-1-lenRen);
             strncpy(buf+lenRen+lenSrc,fname,NBUF-1-lenRen-lenSrc);
             strncpy(buf+lenRen+lenSrc+len,sAllin, NBUF-1-lenRen-lenSrc-len);
             buf[NBUF-1]=0;
             system(buf);
             if(iRunLastpost){
                  /* lastpost */
                  lenEnding = strlen(sLastpost);
                  lenSrc = strlen("lastpost.out "); /* note space */
                  strncpy(buf+lenRen,"lastpost.out ",NBUF-1-lenRen);
                  strncpy(buf+lenRen+lenSrc,fname,NBUF-1-lenRen-lenSrc);
                  strncpy(buf+lenRen+lenSrc+len,sLastpost, NBUF-1-lenRen-lenSrc-len);
                  buf[NBUF-1]=0;
                  system(buf);
                  }
             }
        }
   /*$$$$$$ END Block for automatically renaming files from allpost, etc.$$$$$$*/
   /*::::::::::::::::::::: END Cleanup block ::::::::::::::::::::::::::::::::::::*/

   puts("======== REMINDER: the last input file was saved as \"last_inp\" ========");
   return(0);
   #undef NBUF
   }       /* END main() */
   /***************************** END main() ********************************/
   /*============================ FUNCTIONS ================================*/
   /* Y_or_N() requests a yes or no from the keyboard */
   /*********/
   int Y_or_N(void)
   {
   int i=0;
   for(;;){
        i = toupper(getch());
        if(i=='Y' || i=='N') break;
        puts("Please answer Y or N    ");
        }
   return(i);
   }
   /*-------------------------------------------------------*/
   /* Set Flag structure based on command line arguments. */
```

```
/* NOTE here fp is NOT a file pointer!                    */
/* RETURNS: FLAG_HELP !=1 if finds a "help" character; 1 otherwise. */
/*  The -1 condition should be used to terminate the program... */
/* SEE flags_struct for other notes */
/*******************************************/
int SetFlags(int argc, char *argv[], struct flags_struct *fp)
{
int i,flag,uflag, len;
char *argptr;

static char *str=
" FLAG     ARGUMENT      WHAT IT SETS                              IMPLEMENTED?\n\
 ----     --------      ------------                             ------------\n\
  -i       integer      Inner loop counter, over nxtinput            y\n\
  -o       integer      Outer loop counter, over newpost             y\n\
  -r       integer      auto-Rename files using bldinput.in root     y\n\
                         (0=no, 1=yes; default prompts user)          \n\
  -l       integer      run Lastpost at end to condense allpost      y\n\
                         (0=no, 1=yes; default prompts user)          \n\
  -e       string       filename with Elements to postprocess        N\n\
  -m       string       filename with Minerals to postprocess        y\n\
-h | -?    <none>       display this list                            y";

fp->use_inner=0; fp->use_outer=0; fp->use_rename=0;
fp->use_elemName=0; fp->use_minName=0; fp->use_runLastpost=0;

for(i=1; i<argc; i++){
     /* find next '-' switch */
/* NOTE i<argc MUST come first to avoid core dump at end of list */
     for(;i<argc && argv[i][0]!='-'; i++);
     if(i>=argc) break;
     flag = argv[i][1];
     uflag = toupper(flag);
     len = strlen(argv[i]);
     if(len<=1) continue;
     if(len==2 && (uflag=='H' || flag=='?')){
          puts(str);
          return(FLAG_HELP);
          }
     argptr = argv[i]+2; /* normal case, e.g. -fmyfile */
     if(len==2){    /* has form -f; check if next token is argument */
          if(i<argc-1 && argv[i+1][0] != '-'){
               argptr = argv[i+1];
               }
          else continue;
          }
     else argptr = argv[i] + 2;

     /* set flags according to switch */
     switch(uflag){
          case 'E':
               fp->elemName = argptr;
               fp->use_elemName = 1;
               break;
          case 'M':
               fp->minName = argptr;
               fp->use_minName = 1;
               break;
          case 'I':
               sscanf(argptr,"%d",&(fp->inner));
               if(fp->inner < MIN_INNER){
                    fp->inner = MIN_INNER;
                    }
```

```
                if(fp->inner > MAX_INNER){
                    fp->inner = MAX_INNER;
                    }
                fp->use_inner = 1;
                break;
            case 'O':
                sscanf(argptr,"%d",&(fp->outer));
                if(fp->outer < MIN_OUTER){
                    fp->outer = MIN_OUTER;
                    }
                if(fp->outer > MAX_OUTER){
                    fp->outer = MAX_OUTER;
                    }
                fp->use_outer = 1;
                break;
            case 'R': /* rename the all* and lastpost.out files */
                sscanf(argptr,"%d",&(fp->rename));
                if(fp->rename < MIN_RENAME){
                    fp->rename = MIN_RENAME;
                    }
                if(fp->rename > MAX_RENAME){
                    fp->rename = MAX_RENAME;
                    }
                fp->use_rename = 1;
                break;
            case 'L': /* run lastpost on allpost */
                sscanf(argptr,"%d",&(fp->runLastpost));
                if(fp->runLastpost < MIN_RUNLASTPOST){
                    fp->runLastpost = MIN_RUNLASTPOST;
                    }
                if(fp->runLastpost > MAX_RUNLASTPOST){
                    fp->runLastpost = MAX_RUNLASTPOST;
                    }
                fp->use_runLastpost = 1;
                break;
            }
        }
    return(1);
    }    /* END SetFlags() */
/*-------------------------------------------------------*/
/* This function creates the first part of a header for the (renamed) */
/*  allpost and lastpost files.  The header begins with the descriptive */
/*  portion at the start of bldinput.out, which at the invocation of */
/*  alln_bat.c, contains the initial *.6i file.  The bldinput.in is also */
/*  echoed, along with the command line, current working directory, */
/*  and some information about the system.  */
/* RETURNS: the pointer to the header file.  Note if this is NULL, the */
/*  calling program does not attempt to write a header; though the */
/*  circumstances when a NULL would be returned would be extremely unusual. */
/***************/
FILE *MakeHead(int *ierr, char *headName)
{
#define MAKEHEAD_SZ 128
#define MAKEHEAD_LINES  128
FILE *bldin_fp, *bldout_fp, *head_fp=NULL, *junk_fp;
int i,n,istatus, len;
char buf[MAKEHEAD_SZ], dum[MAKEHEAD_SZ], *str_ptr;
struct tm *ptm;
time_t ltime;
static char *dashes =
  "\n-------------------------------------------------------\n";

*ierr = 0;
```

```
head_fp = fopen(headName,"w");
if(!head_fp){
     return(head_fp); /* can't open main file, give up and report NULL */
     }

/* BEGIN echoing bldinput.out */
bldout_fp = fopen("bldinput.out","r");
if(!bldout_fp) *ierr = 1;
else { /* read bldinput.out */
     for(i=0; i<MAKEHEAD_LINES; i++){
          str_ptr = fgets(buf,MAKEHEAD_SZ,bldout_fp);
          if(str_ptr == buf){
               /* paranoia */
               buf[MAKEHEAD_SZ-1] = 0;
               strncpy(dum,buf,MAKEHEAD_SZ);
               strlwr(dum);
               if(strstr(dum,"| calculational mode")) break; /* end of stuff we want */
               fputs(buf,head_fp);
               }
          else *ierr = 1;
          }
     fclose(bldout_fp);
     }
/* END echoing bldinput.out */
/* BEGIN echoing bldinput.in */
bldin_fp = fopen("bldinput.in","r");
if(bldin_fp){
     fprintf(head_fp,"bldinput.in from cwd=\n");
     fgets(buf, MAKEHEAD_SZ, bldin_fp);
     fputs(buf, head_fp);
     fgets(buf, MAKEHEAD_SZ, bldin_fp);
     fputs(buf, head_fp);
     fclose(bldin_fp);
     }
/* END echoing bldinput.in */
fputs(dashes,head_fp);
/* BEGIN printing localtime, current working directory, apparent user name */
time(&ltime);
ptm = localtime(&ltime);
fprintf(head_fp,"time of run: %02d/%02d/%04d %02d:%02d:%02d\n", ptm->tm_mon+1,
   ptm->tm_mday, ptm->tm_year+1900,ptm->tm_hour, ptm->tm_min, ptm->tm_sec);

/* A note on localtime and Y2K compliance: localtime will correctly print */
/* the year to 2036, which is 1900 + (int)(pow(2.,32.))/(seconds_per_year).. */
/* If the year is clearly incorrect, print a flag to that effect, in case */
/*  someone is still using this program in 2037. */

if(ptm->tm_year>=136 || ptm->tm_year<70){
     fprintf(head_fp, "WARNING: the date above may print year accurately\n");
     fprintf(head_fp, "if true year is >= 2036 (gmtime(), localtime()). \n");
     }

getcwd(buf, MAKEHEAD_SZ);
fprintf(head_fp, "CWD= %s\n", buf);

/* MIGHT eventually encaps next block in an #ifndef UNIX. */
/* The following is specific to DOS systems; aim is to find enough info */
/*  to identify users of the machine (via password list file names) */
/*  without invoking complicated OS username routines. */
/* Find the user name implicit in pwl (Windows password list files) */
/* NOTE: could use GetUserName under win95, but that doesn't identify the */
/*  machine; doing a dir on pwl files gives volume serial number too... */
```

```
/* Below we captute the password filenames, as well as disk serial number, */
/* in a temporary file. */

system("dir c:\\windows\\*.pwl > -_-_-_-_-.-_-");
junk_fp = fopen("-_-_-_-_-.-_-","r");
if(junk_fp){
    fprintf(head_fp,"This machine has following password lists:\n");
    for(i=0; i<10; i++){
        str_ptr = fgets(buf, MAKEHEAD_SZ, junk_fp);
        if(str_ptr==NULL) break;
        /* check for lines we don't need to print; fgets() shoud 0-terminate... */
        len = strlen(buf);
        /* skip lines that are very short or start with 3 blanks -- the latter */
        /* are lines giving bytes free, etc. */
        if(!(len<2 || (len>3 && buf[0]==' ' && buf[1]==' ' && buf[2]==' ')))
            fputs(buf, head_fp);
        }
    fclose(junk_fp);
    unlink("-_-_-_-_-.-_-");
    }

/* END printing localtime, current working directory, apparent user name */
fputs(dashes, head_fp);

return(head_fp);
#undef MAKEHEAD_SZ
#undef MAKEHEAD_LINES
}     /* END MakeHead() */
/*-------------------------------------------------------------*/
/* If a file can be opened to read, and has non-trivial length, it is OK. */
/*  Assume string is 0 terminated. */
int FileOK(char *str)
{
FILE *fp;
int fn,i;

fp = fopen(str,"r");
if(!fp) return(0);
fn = fileno(fp);
i = filelength(fn);
if(i<2){
    fclose(fp); return(0);
    }
fclose(fp);
return(1);
}     /* END FileOK() */
/*-------------------------------------------------------------*/
/* Like GetDBName, but finds full ID name from specified filename, */
/*  compared against specified token (e.g. data0., eq6., eqlib.)
/*  and put it info str; nstr is max size str... currently assume */
/*  file is text, could pass "rb"... */
int GetIDName(char *str, char *fname, char *token, int nstr)
{
#define GETID_SZ 128
FILE *fp;
int i, icode;
char buf[GETID_SZ], *cptr;
icode = 0;

fp = fopen(fname,"r");
if(!fp) return(0);
```

```
for(i=0; i<1000; i++){
     if(!fgets(buf,GETID_SZ,fp)){
          fclose(fp); return(0);
          }
     strlwr(buf);
     cptr = strstr(buf, token);
     if(cptr){
          /* find ending whitespace */
          strtok(cptr, " \n\t,;:"); /* will write a \0 at any */
          strncpy(str, cptr, nstr); str[nstr-1]=0;
          fclose(fp); return(1);
          }
     }
fclose(fp);
return(0);
#undef GETID_SZ
}    /* END GetIDName() */
/*-------------------------------------------------------------*/
```

```
/* This file is newpost.c 6-23-98 HW Stockman */
/*********************************************/
/* Derived from PostprocP.c, latter written by P. Gottlieb, modified by P. Cloke.  */
/* This version designed allow greater flexibility by passing lists of minerals.  */
/* NOTE the columns are now TAB-DELIMITED, so HEADERS MAY NOT */
/*  APPEAR TO LINE UP OVER COLUMNS, but it is now trivially easy to paste tabbed */
/*  blocks of data into Excel.... */
/* NOTE: As of 7-29-98, the option to read in the minerals list is implemented, */
/*  but the option to read in an elements list IS NOT YET IMPLEMENTED.  */
/* 7-10-98 tab the reactants, too... */
/* 7-7-98 hws got working, tested pp40-41 YMP#2 */
/* NOTE to get the minerals list file option to work for solid solutions, */
/*  you should put 3 spaces in front of name, as in PG's original mnrlstr[][] */
/*  array below.  To make sure species like NpO2 are sufficiently unique, you */
/*  should end the name with a space, and verify the space is present in ascii file, */
/*  else species like NpO2SO4- may get picked up, when the actual mineral is NOT */
/*  present!! */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <malloc.h>

#define FLAG_HELP   -1
struct flags_struct {
     char *elemName;  /* list of elements to postproP, -e elements.txt */
     int use_elemName;
     char *minName;  /* list of minerals to postproc, -m minerals.txt */
     int use_minName;
     } FLAG;


#define DAYS_1000_YRS  365248.6  /* hws 7-7-98 */
#define MAX_OUTS_SZ 4000 /* hws 7-7-98 */

double getfloat(char*,int,int);
int locate(char*,char*),getreacts(),numreacts;
void msgerr(char*,int,int),trimb(char*),getelements();
/* hws adds args to getmnrls(), below */

int finished=0;
char dummy[150],reactstrs[20][20];
FILE *fout,*ferr,*fin,*fout,*fallyrs1,*fallyrs2,*fchgyrs1,
  *fchgyrs2,*froot;
float ph,is,mos,mas,hpluss,time,b,gd,ps,pu,u,j13,reactvals[20];

/* hws funcs */
#define MAXMINERALS 100
#define MINERAL_NAME_SZ 33
int ReadMinNames(char *fName, char mName[][MINERAL_NAME_SZ], int maxmin);
void getmnrls(char minNames[][MINERAL_NAME_SZ], float minMoles[], int nMinerals);
int SetFlags(int argc, char *argv[], struct flags_struct *fp);

/* this line below replaces old explicit floats for mineral amounts */
float FMINERAL_MOLE[MAXMINERALS];
/**
     puo2,npo2,amoh,sodd,haiw,rhabdo,gdpo4,ndpo4,smpo4,gdoh,ndoh,euoh,smco3,laf,
     gdf,ndf,smf,agcl,rh2o3,ruo2,dias,hema,goet,trev,nisi,pyro,smec,nonca,nonk,
     nonmg,nonna,cauo,uo3;
**/
/* 6-23-98 we could allocate below, but at 3300 bytes, who cares? */
/* I'll make my var globals all caps */
char MINERAL_NAME[MAXMINERALS][MINERAL_NAME_SZ];
```

```
/* 7-7-98 need to keep the old mineral name list as default */
#define NMIN_PG 33
#define MIN_SZ_PG 20
char MNRL_STRS[NMIN_PG][MIN_SZ_PG]={" AmOHCO3 "," CaUO4 "," Chlorargyrite"," Diaspore","
Eu(OH)CO3(s)",
    " GdOHCO3 "," Goethite"," Haiweeite"," Hematite"," NdOHCO3 "," Ni2SiO4 ",
    " NpO2 "," PuO2 "," Pyrolusite"," Rh2O3 "," RuO2 "," Sm2(CO3)3 "," Soddyite",
    " Trevorite"," UO3:2H2O"," Smectite-di","  Nontronite-Ca","  Nontronite-K",
    "  Nontronite-Mg","  Nontronite-Na"," Rhabdophane-ss","  NdPO4:H2O",
    "  GdPO4:H2O","  SmPO4:H2O","  LaF3:0.5H2O","  NdF3:0.5H2O",
    "  GdF3:0.5H2O","  SmF3:0.5H2O"};

struct OUTREC
   {struct OUTREC *next;
   char data[1000];};
/********************** MAIN ***************************/
void main(int argc, char *argv[])
{
int i,j,k,bcount=0,lcount=0,endblock,firstall=1,firstchg=1,
      firsttime=1,newblock=1,fileflag=0;
struct OUTREC *pallyrs1,*pallyrs2,*pallyrs3,
     *pchgyrs1,*pchgyrs2,*pchgyrs3,*p,
     *pfallyrs1,*pfallyrs2,*pfallyrs3,
        *pfchgyrs1,*pfchgyrs2,*pfchgyrs3;
char outs[4][MAX_OUTS_SZ], fstr[50],rootstr[50];
int nMinerals, nElements, ii, ilen; /* hws; ii for safety, in case PG depends on i */

/* hws 7-7-98: look for minerals list name, and attempt to read; if can't read, */
/*  fill in default minerals list */
/*^^^^^^^^^^^^^^^^ Translate flags to local vars ^^^^^^^^^^^^^^^*/
if(SetFlags(argc, argv, &FLAG)==FLAG_HELP){
     exit(0);  /* if we displayed help, quit */
     }


nMinerals = 0;
if(FLAG.use_minName){
     nMinerals = ReadMinNames(FLAG.minName, MINERAL_NAME, MAXMINERALS);
     if(nMinerals>0)
        printf("==> FOUND %d Minerals in %s\n", nMinerals, argv[1]);
     }
if(!nMinerals){ /* copy PG's old list; can't just associate pointers, */
    /* because we might depend on having full MINERAL_NAME_SZ some day */
    nMinerals = NMIN_PG;
    for(i=0; i<NMIN_PG; i++) strncpy(MINERAL_NAME[i],MNRL_STRS[i],MINERAL_NAME_SZ);
    printf("defaulted to standard PostprocP minerals list\n");
    }

if((froot=fopen("rootname","r"))!=NULL)
  {fscanf(froot,"%s",rootstr);
  fclose(froot);
  strcpy(fstr,rootstr);
  strcat(fstr,".allout");
  if((fin=fopen(fstr,"r"))!=NULL)fileflag=1;}
if(fileflag==0)
 if((fin=fopen("allout","r"))==NULL)
  {printf("Cant open input file\n");
  exit(0);}
if(fileflag==0)fout=fopen("postproc.out","w");
else
  {strcat(rootstr,".postproc");
  fout=fopen(rootstr,"w");}
```

```
printf("filename=%s  fileflag=%d\n",fstr,fileflag);
while(finished==0)
  {if((k=locate("                          Time = "," J-13 water"))==1)
    {fgets(dummy,100,fin);
       time=getfloat(dummy,29,11);}
  else if(k==0) msgerr("Missed time",bcount,lcount);
  else break; /*proper end of file */
  if((k=locate("     Reactant                    Moles",
       "      --- Element Totals"))==1)
    {if(firsttime==1)
          {numreacts=getreacts(1);
             firsttime=0;}
        else getreacts(0);}
  else if(k==0) msgerr("Missed reactants",bcount,lcount);
  else msgerr("Unexpected end of file",bcount,lcount);
  getelements();
  if((k=locate("     modified NBS pH scale"," H+"))==1)
     ph=getfloat(dummy,37,8);  /* pH */
  else if(k==0) msgerr("Missed pH",bcount,lcount);
  else msgerr("Unexpected end of file",bcount,lcount);
  if((k=locate("                        Ionic strength"," H+"))==1)
     is=getfloat(dummy,38,13);  /* Ionic strength */
  else if(k==0) msgerr("Missed ionic str",bcount,lcount);
  else msgerr("Unexpected end of file",bcount,lcount);
  if((k=locate("                        Moles of solvent"," H+"))==1)
     mos=getfloat(dummy,44,13);  /* Moles solvent water */
  else if(k==0) msgerr("Missed moles water",bcount,lcount);
  else msgerr("Unexpected end of file",bcount,lcount);
  if((k=locate("                        Mass of solvent"," H+"))==1)
     mas=getfloat(dummy,44,13);  /* Mass solvent water */
  else if(k==0) msgerr("Missed mass water",bcount,lcount);
  else msgerr("Unexpected end of file",bcount,lcount);
  if((k=locate(" H+","          --- Summary of Solid Product Phases---"))==1)
     hpluss=-getfloat(dummy,68,9);  /* H+ */
  else if(k==0) msgerr("Missed H+",bcount,lcount);
  else msgerr("Unexpected end of file",bcount,lcount);
  if(fabs(ph-hpluss)>1.e-4)
    {printf("%f %f\n",ph,hpluss);
       msgerr("pH mismatch",bcount,lcount);}
  getmnrls(MINERAL_NAME, FMINERAL_MOLE, nMinerals);

  if((k=locate("      Time increased from",
     "                    Reaction progress"))==1)
        {endblock=1;
        bcount++;}
  else if (k==0) endblock=0;
  else finished=1;
  lcount++;
  printf("%d %d\n",bcount,lcount);
  if ((endblock==0)||(finished==1))
  {sprintf(outs[0],"%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e\n",
     time/365.2486/1000,ph,b,gd,ps,pu,u,is,mos);
/** hws comments out old code 7-7-98 **
  sprintf(outs[1],

"%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.
3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11.3e%11
.3e%11.3e%11.3e\n",
     time/365.2486/1000,puo2,npo2,amoh,sodd,haiw,cauo,uo3,rhabdo,gdpo4,ndpo4,smpo4,gdoh,
     ndoh,euoh,smco3,laf,gdf,ndf,smf,agcl,rh2o3,ruo2,dias,hema,goet,trev,nisi,
     pyro,smec,nonca,nonk,nonmg,nonna);
** end hws comment out 7-7-98 **/
/** hws replacement code 7-7-98: **/
```

```
sprintf(outs[1],"%11.3e", time/DAYS_1000_YRS);
for(ii=0; ii<nMinerals; ii++){
      ilen = strlen(outs[1]);
      if(ilen>MAX_OUTS_SZ-13) break;
         sprintf(outs[1]+ilen,"\t%11.3e", FMINERAL_MOLE[ii]);
      }
strcat(outs[1],"\n");
/** END hws replacement code 7-7-98 */
/* hws 7-10-98 convert time to 12.3 for consistency with tab separator below */
  sprintf(outs[2],"%12.3e",time/365.2486/1000); /* 7-10-98 hws adds tab below */
  /* 7-10-98 also requires converting 11 to 12 */
  for(i=0;i<numreacts;i++)sprintf(outs[2]+12*(i+1),"\t%11.3e",reactvals[i]);
  outs[2][12*(numreacts+1)]='\n';
  outs[2][12*(numreacts+1)+1]='\0'; /* 7-10-98 paranoia */
  if(firstall==1)
    {firstall=0;
       pfallyrs1=malloc(sizeof(struct OUTREC));
       pfallyrs2=malloc(sizeof(struct OUTREC));
       pfallyrs3=malloc(sizeof(struct OUTREC));
       pallyrs1=pfallyrs1;
       pallyrs2=pfallyrs2;
       pallyrs3=pfallyrs3;}
  else
    {if((pallyrs1->next=malloc(sizeof(struct OUTREC)))==NULL)
          msgerr("malloc",bcount,lcount);
     if((pallyrs2->next=malloc(sizeof(struct OUTREC)))==NULL)
          msgerr("malloc",bcount,lcount);
     if((pallyrs3->next=malloc(sizeof(struct OUTREC)))==NULL)
          msgerr("malloc",bcount,lcount);
       pallyrs1=pallyrs1->next;
       pallyrs2=pallyrs2->next;
       pallyrs3=pallyrs3->next;}
  strcpy(pallyrs1->data,outs[0]);
  strcpy(pallyrs2->data,outs[1]);
  strcpy(pallyrs3->data,outs[2]);
  pallyrs1->next=NULL;
  pallyrs2->next=NULL;
  pallyrs3->next=NULL;
  if(newblock==1)
    {if(firstchg==1)
         {firstchg=0;
      pfchgyrs1=malloc(sizeof(struct OUTREC));
          pfchgyrs2=malloc(sizeof(struct OUTREC));
          pfchgyrs3=malloc(sizeof(struct OUTREC));
          pchgyrs1=pfchgyrs1;
          pchgyrs2=pfchgyrs2;
          pchgyrs3=pfchgyrs3;}
    else
      {if((pchgyrs1->next=malloc(sizeof(struct OUTREC)))==NULL)
         msgerr("malloc",bcount,lcount);
          pchgyrs2->next=malloc(sizeof(struct OUTREC));
          pchgyrs3->next=malloc(sizeof(struct OUTREC));
          pchgyrs1=pchgyrs1->next;
          pchgyrs2=pchgyrs2->next;
          pchgyrs3=pchgyrs3->next;}
    strcpy(pchgyrs1->data,outs[0]);
    strcpy(pchgyrs2->data,outs[1]);
    strcpy(pchgyrs3->data,outs[2]);
    pchgyrs1->next=NULL;
    pchgyrs2->next=NULL;
    pchgyrs3->next=NULL;
        newblock=0;}}
 if(endblock==1) newblock=1;}
```

```
fprintf(fout,"\n\nDATA FOR EACH TIMESTEP Elements\n\n");
fprintf(fout,"%11s%11s%11s%11s%11s%11s%11s%11s%11s\n",
   "1000yr","pH","MolesB","MolesGd","MolesP","MolesPu",
      "MolesU","IonicStr","MlsH20");
p=pfallyrs1;
fputs(p->data,fout);
while((p=p->next)!=NULL) fputs(p->data,fout);
fprintf(fout,"\n\nDATA FOR EACH TIMESTEP Minerals\n\n");
/** hws 7-7-98 comment out old code **
fprintf(fout,"%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s\n",

"1000yr","MlPuO2","MlNpO2","MlAmOHCO3","MlSodd","MlHaiwee","MlCaUO4","MlUO3:2H2O","MlRhabdo",
   "MlGdPO4","MlNdPO4","MlSmPO4","MlGdOHCO3","MlNdOHCO3","MlEuOHCO3",
   "MlSm2(CO3)3","MlLaF3","MlGdF3","MlNdF3","MlSmF3","MlChlorarg","MlRh2O3",
   "MlRuO2","MlDiaspore","MlHematite","MlGoethite","MlTrevorite","MlNi2SiO4",
   "MlPyrolusi","MlSmectite","MlNontro-Ca","MlNontro-K","MlNontro-Mg",
   "MlNontro-Na");
** END hws 7-7-98 comment out old code **/
/** 7-7-98 hws start replacement code **/
fprintf(fout,"1000yr");
for(ii=0; ii<nMinerals; ii++){
     fprintf(fout,"\t%s",MINERAL_NAME[ii]);
     }
fprintf(fout,"\n");
/* &&&&&&&&&&& */
/** 7-7-98 hws END replacement code */

p=pfallyrs2;
fputs(p->data,fout);
while((p=p->next)!=NULL) fputs(p->data,fout);
fprintf(fout,"\n\nDATA FOR EACH TIMESTEP Reactants\n\n");
  /* 7-10-98 11s to 12s for time for consistency with tabs */
  sprintf(outs[3],"%12s","1000yr"); /* hws 7-10-98 adds tab below; 11* to 12* */
  for(i=0;i<numreacts;i++)sprintf(outs[3]+12*(i+1),"\t%11s",reactstrs[i]);
fprintf(fout,"%s\n",outs[3]);
p=pfallyrs3;
fputs(p->data,fout);
while((p=p->next)!=NULL) fputs(p->data,fout);
fprintf(fout,"\n\nDATA FOR CHANGING TIMESTEPS Elements\n\n");
fprintf(fout,"%11s%11s%11s%11s%11s%11s%11s%11s%11s\n",
   "1000yr","pH","MolesB","MolesGd","MolesP","MolesPu",
      "MolesU","IonicStr","MlsH20");
p=pfchgyrs1;
fputs(p->data,fout);
while((p=p->next)!=NULL) fputs(p->data,fout);
fprintf(fout,"\n\nDATA FOR CHANGING TIMESTEPS Minerals\n\n");
/** 7-7-98 hws comment out old code ** .
fprintf(fout,"%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s%11s\n",

"1000yr","MlPuO2","MlNpO2","MlAmOHCO3","MlSodd","MlHaiwee","MlCaUO4","MlUO3:2H2O","MlRhabdo",
   "MlGdPO4","MlNdPO4","MlSmPO4","MlGdOHCO3","MlNdOHCO3","MlEuOHCO3",
   "MlSm2(CO3)3","MlLaF3","MlGdF3","MlNdF3","MlSmF3","MlChlorarg","MlRh2O3",
   "MlRuO2","MlDiaspore","MlHematite","MlGoethite","MlTrevorite","MlNi2SiO4",
   "MlPyrolusi","MlSmectite","MlNontro-Ca","MlNontro-K","MlNontro-Mg",
   "MlNontro-Na");
** END hws 7-7-98 comment out old code **/
/** 7-7-98 hws start replacement code **/
fprintf(fout,"1000yr");
for(ii=0; ii<nMinerals; ii++){
     fprintf(fout,"\t%s",MINERAL_NAME[ii]);
     }
```

```
fprintf(fout,"\n");
/* &&&&&&&&&& */
/** 7-7-98 hws END replacement code */
p=pfchgyrs2;
fputs(p->data,fout);
while((p=p->next)!=NULL) fputs(p->data,fout);
fprintf(fout,"\n\nDATA FOR CHANGING TIMESTEPS Reactants\n\n");
fprintf(fout,"%s\n",outs[3]);
p=pfchgyrs3;
fputs(p->data,fout);
while((p=p->next)!=NULL) fputs(p->data,fout);
/* hws 6-22-98 */
if(fout) fclose(fout);
if(fin) fclose(fin);
}
/********************** END MAIN ****************************/


void msgerr(char msgstr[50],int i,int j)
{fprintf(fout,"%s  block count = %d  line count = %d\n",msgstr,i,j);
printf("%s  block count = %d  line count = %d\n",msgstr,i,j);
/*exit(0);*/}

int locate(char sstring[60],char estring[50])
{int i,j;
i=strlen(sstring);
j=strlen(estring);
while(fgets(dummy,100,fin)!=NULL)
  if (strncmp(dummy,sstring,i)==0)return 1;
  else if (strncmp(dummy,estring,j)==0)return 0;
return -1;}

double getfloat(string,start,len)
char string[100];
int start,len;
{char temp[30];
strncpy(temp,string+start,len);
temp[len]='\0';
return atof(temp);}
/*-----------------------------------------------------------*/
/* 7-7-98 hws edits getmnrls() heavily */
void getmnrls(char minNames[][MINERAL_NAME_SZ], float minMoles[], int nMinerals)
{
int i,k,founds[MAXMINERALS],finished=0,slens[MAXMINERALS];
char ss[]="            --- Summary of Pure Mineral Saturation States ---";
for(i=0;i<nMinerals;i++){
      slens[i]=strlen(minNames[i]);
      founds[i] = 0;
      }
k=strlen(ss);
while((fgets(dummy,100,fin)!=NULL)&&(finished==0)){
     if(strncmp(dummy,ss,k)==0) finished=1;
     else {
          for(i=0;i<nMinerals;i++){
               if(strncmp(dummy,minNames[i],slens[i])==0){
                    founds[i]=1;
                    minMoles[i] = getfloat(dummy,40,12);
                    }
               }
          }
     }

for(i=0; i<nMinerals; i++){
```

```
      if(founds[i]==0) minMoles[i] = 0.0f;
      }
} /* END getmnrls() */
/*-----------------------------------------*/

void getelements()
{int i,k,num=5,founds[5]={0},finished=0,slens[10];
char elstrs[5][20]={"      B ","      Gd","      P ","      Pu",
    "      U "},ss[]={"   Single ion"};
for(i=0;i<num;i++) slens[i]=strlen(elstrs[i]);
k=strlen(ss);
while((fgets(dummy,100,fin)!=NULL)&&(finished==0))
  {if(strncmp(dummy,ss,k)==0)finished=1;
  else
    for(i=0;i<num;i++)
         if(strncmp(dummy,elstrs[i],slens[i])==0)
              {founds[i]=1;
              switch(i)
                {case 0:b=getfloat(dummy,57,13);break;
                    case 1: gd=getfloat(dummy,57,13);break;
                    case 2: ps=getfloat(dummy,57,13);break;
                    case 3: pu=getfloat(dummy,57,13);break;
                    case 4: u=getfloat(dummy,57,13);break;}}}
for(i=0;i<num;i++)
  if(founds[i]==0)
   switch(i)
    {case 0:b=0;break;
        case 1: gd=0;break;
        case 2: ps=0;break;
        case 3: pu=0;break;
        case 4: u=0;break;}}


int getreacts(int k)
{int i;
char temps[30];
fgets(dummy,100,fin); /*skip blank line*/
i=0;
fgets(dummy,100,fin); /*now read first line of reactants*/
while(dummy[0]!='\n')
  {if(k==1)
    {strncpy(temps,dummy,25);
    temps[25]='\0';
    trimb(temps);
    strcpy(reactstrs[i],temps);}                    /* name of reactant */
  reactvals[i]=getfloat(dummy,29,11); /* moles of reactant */
  i++;
  fgets(dummy,100,fin);}
return(i);}

void trimb(char string[30])
{int i=0,j,k;
while(string[i]==' ')i++;
j=strlen(string)-1;
while(string[j]==' ')j--;
for(k=0;k<j-i+1;k++) string[k]=string[k+i];
if(j-i+1<9)string[j-i+1]='\0';
else string[9]='\0';} /*no reactant string name greater than 9chars*/



/*                    Time =
```

```
                    = 3.329E+06 days
  J-13 water              3.9080E+03    .0000E+00   2.3471E+04    .0000E+00
      B          1.405415E-01          1.316626E-05          1.316629E-05
      Gd         1.531378E-03          9.863147E-09          9.863168E-09
      P          1.742931E-04          5.699148E-09          5.699160E-09
      Pu         5.260290E-07          2.183452E-12          2.183456E-12
      U          2.582874E-03          1.098999E-08          1.099001E-08
    modified NBS pH scale             6.6651
                  Ionic strength =  2.596699E-01 molal
                  Moles of solvent H2O =  5.55085E+01
                  Mass of solvent H2O  =  1.00000E+00 kg
 H+                          2.7749E-07     -6.5567      -.1083      -6.6651
 PuO2                         -5.5313   2.9425E-06   8.1213E-04   7.0120E-05
 Soddyite                     -1.5865   2.5912E-02   1.7314E+01   3.4015E+00
 Rhabdophane-ss               -2.1365   7.3031E-03   1.9141E+00    .0000E+00
   GdPO4:H2O                  -2.6521   2.2280E-03   6.0208E-01    .0000E+00
*/
/*-------------------- HWS functions ----------------------*/
int ReadMinNames(char *fName, char mName[][MINERAL_NAME_SZ], int maxmin)
{
#define NBUF 512
FILE *fp;
int i,n;
char buf[NBUF]; /* to make sure we get entire line */

/* first see if we can open the file; if we can't, pass back 0 */
fp = fopen(fName,"r"); /* assume ascii, one name per line */
if(!fp) return(0);

for(n=0; n<maxmin; n++){
     /* printf("in loop, index %d\n", n); */
     if(!fgets(buf, NBUF, fp)) break;
     buf[MINERAL_NAME_SZ-1]=0; /* I'm paranoid */
     i = (buf[0] != ' ')? 1 : 0;
     strncpy(mName[n]+i,buf,MINERAL_NAME_SZ-i);
     if(i) mName[n][0] = ' ';
     /* strip off any carriage returns or newlines */
     for(i=0; i<MINERAL_NAME_SZ; i++){
          if(mName[n][i]<=15) mName[n][i] = 0;
          }
     }

if(fp) fclose(fp);
return(n);
#undef NBUF
}   /* END ReadMinNames() */
/*-------------------------------------------------------*/
/* 7-8-98 modify allp_bat.c function to set some inportant params based */
/*  on flags on command line... */
/* NOTE here fp is NOT a file pointer! */
/* RETURNS: FLAG_HELP !=1 if finds a "help" character; 1 otherwise. */
/*  The -1 condition should be used to terminate the program... */
/* SEE flags_struct for other notes */
/*****************************************/
int SetFlags(int argc, char *argv[], struct flags_struct *fp)
{
int i,flag,uflag, len;
char *argptr;

static char *str=
" FLAG     ARGUMENT      WHAT IT SETS                          IMPLEMENTED?\n\
 ----     --------      ------------                          ------------\n\
  -e       string       filename with Elements to postproP          N\n\
```

```
  -m        string      filename with Minerals to postproP          y\n\
-h | -?     <none>      display this list                           y";

fp->use_elemName=0; fp->use_minName=0;

for(i=1; i<argc; i++){
     /* find next '-' switch */
/* NOTE i<argc MUST come first to avoid core dump at end of list */
     for(;i<argc && argv[i][0]!='-'; i++);
     if(i>=argc) break;
     flag = argv[i][1];
     uflag = toupper(flag);
     len = strlen(argv[i]);
     if(len<=1) continue;
     if(len==2 && (uflag=='H' || flag=='?')){
          puts(str); /* 3-7-97 correct */
          return(FLAG_HELP);
          }
     argptr = argv[i]+2; /* normal case, e.g. -fmyfile */
     if(len==2){    /* has form -f; check if next token is argument */
          if(i<argc-1 && argv[i+1][0] != '-'){
               argptr = argv[i+1];
               }
          else continue;
          }
     else argptr = argv[i] + 2;

     /* set flags according to switch */
     switch(uflag){
          case 'E':
               fp->elemName = argptr;
               fp->use_elemName = 1;
               break;
          case 'M':
               fp->minName = argptr;
               fp->use_minName = 1;
               break;
          }
     }
return(1);
}    /* END SetFlags() */
/* This file is lastpost.c */
/*****************************/
/* lastpost.c  processes a file named allpost, which is the result of
   concatenating the results of a sequence of runs of postproc.c representing
        consecutive timesteps which have been sliced into blocks so that the
        output files do not grow too large to handle.  The result of the concatenation
        is a sequence of six table groups, with the groups representing sequential
        timesteps.  This program merges the individual tables accros all the groups,
        resulting in a set of six tables, each covering the entire timespan.
        The present version is also set to print only every tenth line to reduce
        the size of the output file so that it can be easily graphed from a
        spreadsheet.*/
#define LASTPOST_LINE_SZ 4000
#include <stdio.h>  /* LASTPOST_LINE_SZ replaces value of 400... (hws) */
#include <string.h>
#include <stdlib.h>
#include <malloc.h>

FILE *fin, *fout;

struct OUTREC  /* for linked list of output records */
   {struct OUTREC *next;
```

```
   char data[LASTPOST_LINE_SZ];};

void main()
{int i, j, count=0,finished=0;
struct OUTREC *pyrs[6],/*used for constructing one linked list for each table*/
    *pfyrs[6],/*used for the start of each linked list*/
            *p;/*used for traversing the linked list to write the output file*/
char outs[LASTPOST_LINE_SZ], /*for output line*/
   recstrs[6][100]={"DATA FOR EACH TIMESTEP Elements",
   "DATA FOR EACH TIMESTEP Minerals","DATA FOR EACH TIMESTEP Reactants",
   "DATA FOR CHANGING TIMESTEPS Elements","DATA FOR CHANGING TIMESTEPS Minerals",
   "DATA FOR CHANGING TIMESTEPS Reactants"}, /*headings for input file tables*/
       dummy[LASTPOST_LINE_SZ],/*for reading a line of input data*/
   headstrs[6][LASTPOST_LINE_SZ];/*will be used for column headings for each output table.*/
fin=fopen("allpost","r");/*input data  file*/
fout=fopen("lastpost.out","w");/*output file*/

/* 6-24-98 hws adds the following: */
if(!fin){
     puts("Can't open allpost file-- do you have to rename another file allpost?");
     if(fout) fclose(fout);
     exit(1);
     }
if(!fout){
     puts("Can't open output file lastpost.out -- out of disk space?");
     if(fin) fclose(fin);
     exit(2);
     }
/* end 6-24-98 hws addition for file errors */

for(i=0;i<6;i++) /*allocate memory for start of each linked list*/
  {pfyrs[i]=malloc(sizeof(struct OUTREC));
   pyrs[i]=pfyrs[i];
   pyrs[i]->next=malloc(sizeof(struct OUTREC));}/*next rec for the first data*/
while((finished==0)&&(fgets(dummy,LASTPOST_LINE_SZ,fin)!=NULL))/*outer loop to read all data*/
  for(i=0;i<6;i++)/*inner loop to read each group of six*/
/*starting with the first line read in the above while statement, read through
  lines until the first table heading is reached.  On subsequent passes, it
       will read through the blank lines before the next table*/
          {while((finished==0)&&(strncmp(dummy,recstrs[i],strlen(recstrs[i]))!=0))
       if(fgets(dummy,LASTPOST_LINE_SZ,fin)==NULL)/*EOF if we run out of lines*/
         {finished=1;
         break;}
    fgets(dummy,LASTPOST_LINE_SZ,fin);/*readthrough a blank line following the table heading*/
    fgets(dummy,LASTPOST_LINE_SZ,fin);
    strcpy(headstrs[i],dummy);/*copy the column headings for use in the output*/
    fgets(dummy,LASTPOST_LINE_SZ,fin);/*now get the first data line*/
            /*the following test includes whether the input line is blank, which would
            indicate the end of the input table.*/
    while((finished==0)&&(strncmp(dummy,"      ",6)!=0)&&(dummy[0]!='\n'))
      {pyrs[i]=pyrs[i]->next;
      strcpy(pyrs[i]->data,dummy);/*if not blank, copy it to the linked list*/
      pyrs[i]->next=malloc(sizeof(struct OUTREC));/*allocate for the next line*/
      if(i==0)count++;
      if(fgets(dummy,LASTPOST_LINE_SZ,fin)==NULL)finished=1;}}/*get the line for the next*/
                                          /*iteration and test for EOF*/
for(i=0;i<6;i++)
  {free(pyrs[i]->next);/*free the last allocation which won't be needed*/
  pyrs[i]->next=NULL;}/*now tag the last link*/
for(i=0;i<6;i++)
  {count=0;
  fprintf(fout,"\n\n%s\n\n",recstrs[i]);/*print table heading*/
  fprintf(fout,"%s\n",headstrs[i]);/*print column headings*/
```

```
  p=pfyrs[i];/*point to start of linked list*/
  while((p=p->next)!=NULL) /*skip the first record which has no data*/
    {if (count%10==0) fprintf(fout,"%s",p->data);/*print every tenth line*/
    count++;}}
/* hws 6-19-98 */
if(fin) fclose(fin); if(fout) fclose(fout);
}
```

Attachment II. Files Archived on QIC-80 DT-350 Tapes (Ref. 16).

Each file listed in table 5.3-1 has been archived along with its associated output files, and stored on a QIC-80 DT-350 mini-cartridge tape (170 MB native capacity, 350 MB compressed; tapes were written with Colorado Backup for Windows 95, version 1.70). For each EQ6 run in table 5.3-1, the following files will be found on the tapes:

`filename.6i`	(the original EQ6 starting input file)
`filename.lastpost`	(the output of lastpost.exe; a condensed version of allpost for plotting)
`filename.allpost`	(the output of `alln_bat.exe`, tabular format for plotting)
`filename.allin`	(all input files used `alln_bat.exe`, concatenated)
`filename.alltab`	(all tab files generated by EQ6 in the run, concatenated)

The filenames listed in table 5.3-1 can be used to infer the run conditions. Each file name consists of up to six pieces before the `6i` extension:

`Cer dn WorN driprate part extrainfo`

The "`Cer`" prefix indicates ceramic, and is on all files. The only other mandatory piece is the `driprate`, which has format `x_xxxx`; when the underscore is replaced with a decimal point, `x.xxxx` represents the drip rate in m^3/yr (see section 5.1.1.3). In table 5.3-1, `x_xxxx` takes values `0_5`, `0_015` and `0_0015`. (note "`0`" in the driprate is "zero", not "oh"). The `dn` designates series "d", and the "n" indicates a set of rates used for the calculations, taking values `1,2,3,4` and `5`, as indicated in table 5.3-1. The optional `WorN` takes values `W` or `N`, respectively, if the waste form (the Pu-ceramic) is present or not present in the run. The `part` number is a Roman numeral, which is `I` for all files described in this study, since each stage consisted of only one part. The `extrainfo` designates extra information about the run, in this case the use of a comparatively low CO_2 pressure ($10^{-3.5}$ atm, the surface ambient, in place of the YMP ambient of ~$10^{-2.5}$ atm), or suppression of hematite (`nH`).

The comment lines of the EQ6 inputs file (6i files) contain descriptions of the runs, with run histories. Some of the comment lines may contain inaccuracies in descriptions, which are retained to maintain the traceability to prior versions. The molar amounts and rates used in the calculations are given in the uncommented (active) parts of the input files, and these values take precedence over any values that might be inferred from the comment lines.

A complete listing of the files on the tapes is given below. The tapes also contain the spreadsheets used in the calculations, and the EQ6 database data0.nuc.R8.

Listing of files on the electronic media (from MD-DOS command: dir > dir.txt):

```
 tremain  xls          72,704  08-13-98 12:04p tremain.xls
 327ceram doc         625,152  08-29-98  1:06p 327ceram.doc
 CER0_0~1 6I           40,892  08-05-98  5:45p Cer0_0015I!.6i
 CER0_0~1 ALL      91,010,554  08-05-98  9:57p cer0_0015I!.allin
 CER0_0~2 ALL       3,021,322  08-05-98  9:58p cer0_0015I!.allpost
 CER0_0~3 ALL      67,586,610  08-05-98  9:57p cer0_0015I!.alltab
 CER0_0~1 LAS         301,811  08-05-98  9:58p cer0_0015I!.lastpost
 CER0_0~1 XLS         584,704  08-19-98  2:17p Cer0_0015InMnC.xls
 CER0_5~1 XLS       1,702,400  08-19-98  3:20p Cer0_5&0_0015CO2_Lo.xls
 CER0_5~2 XLS       1,736,192  08-19-98  2:25p Cer0_5&0_0015D.xls
 CERD1N~1 6I           40,516  07-22-98  3:49p Cerd1N0_5I.6i
 CERD1N~2 ALL     111,103,772  07-22-98  9:12p Cerd1N0_5I.allin
 CERD1N~1 ALL       4,836,849  07-22-98  9:12p Cerd1N0_5I.allpost
 CERD1N~3 ALL      84,904,950  07-22-98  9:13p Cerd1N0_5I.alltab
 CERD1N~1 LAS         405,271  07-23-98  4:12p Cerd1N0_5I.lastpost
 CERD1N~2 6I           40,886  08-21-98  2:28p Cerd1N0_5I_nH.6i
 CERD1N~4 ALL     112,776,840  08-23-98  4:16p Cerd1N0_5I_nH.allin
 CE93E2~1 ALL       3,433,336  08-23-98  4:16p Cerd1N0_5I_nH.allpost
 CE4666~1 ALL      86,152,950  08-23-98  4:16p Cerd1N0_5I_nH.alltab
 CERD1W~1 6I           47,839  07-23-98  9:05a Cerd1W0_0015I.6i
 CERD1N~1 ALL      58,482,722  07-23-98 12:25p Cerd1W0_0015I.allin
 CERD1W~2 ALL       2,118,331  07-23-98 12:25p Cerd1W0_0015I.allpost
 CERD1W~3 ALL      43,821,700  07-23-98 12:25p Cerd1W0_0015I.alltab
 CERD1N~1 LAS         215,058  07-23-98 12:25p Cerd1W0_0015I.lastpost
 CERD2N~1 6I           40,590  07-24-98  4:40p Cerd2N0_5I.6i
 CERD2N~2 ALL     112,117,444  07-24-98  9:13p Cerd2N0_5I.allin
 CERD2N~1 ALL       4,132,816  07-24-98  9:15p Cerd2N0_5I.allpost
 CERD2N~3 ALL      86,544,900  07-24-98  9:13p Cerd2N0_5I.alltab
 CERD2N~1 LAS         412,167  07-24-98  9:15p Cerd2N0_5I.lastpost
 CERD2W~1 6I           48,726  08-27-98  5:32p cerd2W0_0015I.6i
 CERD2W~1 ALL      98,157,856  08-27-98  9:02p Cerd2W0_0015I.allin
 CERD2W~2 ALL       3,012,456  08-27-98  9:04p Cerd2W0_0015I.allpost
 CERD2W~3 ALL      73,923,330  08-27-98  9:02p Cerd2W0_0015I.alltab
 CERD2W~1 LAS         302,705  08-27-98  9:04p Cerd2W0_0015I.lastpost
 CERD3W~2 6I           48,061  07-28-98  5:37p Cerd3W0_0015I.6i
 CERD3W~4 ALL      59,184,542  07-28-98  7:47p Cerd3W0_0015I.allin
 CE5C55~1 ALL       2,119,637  07-28-98  7:47p Cerd3W0_0015I.allpost
 CEC75B~1 ALL      44,809,960  07-28-98  7:47p Cerd3W0_0015I.alltab
 CERD3W~2 LAS         215,850  07-28-98  7:47p Cerd3W0_0015I.lastpost
 CERD3W~1 6I           48,209  07-29-98  5:25p Cerd3W0_0015I_CO2_LO.6i
 CERD3W~2 ALL      59,531,882  07-29-98  7:28p Cerd3W0_0015I_CO2_LO.allin
 CERD3W~1 ALL       2,118,647  07-29-98  7:29p Cerd3W0_0015I_CO2_LO.allpost
 CERD3W~3 ALL      45,115,720  07-29-98  7:29p Cerd3W0_0015I_CO2_LO.alltab
 CERD3W~1 LAS         215,432  07-29-98  7:29p Cerd3W0_0015I_CO2_LO.lastpost
 CERD3W~3 6I           48,759  08-23-98  4:37p Cerd3W0_0015I_CO2_LO_nH.6i
 CEB66C~1 ALL      99,986,282  08-24-98  6:22a Cerd3W0_0015I_CO2_LO_nH.allin
 CE0F71~1 ALL       3,005,516  08-24-98  6:23a Cerd3W0_0015I_CO2_LO_nH.allpost
 CE2D93~1 ALL      75,698,870  08-24-98  6:22a Cerd3W0_0015I_CO2_LO_nH.alltab
 CERD3W~3 LAS         302,597  08-24-98  6:23a Cerd3W0_0015I_CO2_LO_nH.lastpost
 CERD4W~1 6I           48,948  08-20-98  6:02p cerd4W0_015I.6i
 CERD4W~3 ALL      98,547,654  08-21-98  8:05a Cerd4W0_015I.allin
 CERD4W~1 ALL       3,080,410  08-21-98  8:06a Cerd4W0_015I.allpost
 CERD4W~2 ALL      72,938,580  08-21-98  8:05a Cerd4W0_015I.alltab
```

```
CERD4W~1 LAS        309,691  08-21-98  8:06a Cerd4W0_015I.lastpost
CERD5W~1 6I          48,431  08-04-98  7:38p Cerd5W0_0015I_CO2_LO.6i
CERD5W~1 ALL     99,513,142  08-04-98 11:13p Cerd5W0_0015I_CO2_LO.allin
CERD5W~2 ALL      3,006,564  08-04-98 11:14p Cerd5W0_0015I_CO2_LO.allpost
CERD5W~3 ALL     73,664,630  08-04-98 11:13p Cerd5W0_0015I_CO2_LO.alltab
CERD5W~1 LAS        302,669  08-04-98 11:14p Cerd5W0_0015I_CO2_LO.lastpost
DATA0N~1 R8       2,298,907  05-28-98 11:26a data0.nuc.R8
dir      txt          4,816  08-29-98  2:04p dir.txt
GD%REM~1 XLS         15,872  08-06-98  2:28p Gd%remaining.xls
GD_LOS~1 XLS         17,408  08-28-98  5:01p Gd_loss_from_peak.xls
GD_RAI~1 XLS         17,408  08-24-98  4:01p Gd_Rai_recalc.xls
HFPYRO~1 XLS         27,136  04-20-98  9:35a Hfpyromfr5.xls
Masses5  xls         75,264  08-25-98 10:49a Masses5.xls
readme   txt          1,810  08-29-98  2:01p readme.txt
run6     xls         65,536  08-20-98  4:14p run6.xls
run7     xls        221,696  08-28-98  3:53p run7.xls
VOLMAS~1 XLS        973,824  06-22-98  1:04p volmas21a.xls
        70 file(s)  1,801,381,895 bytes
         2 dir(s)   2,147,450,880 bytes free
```