550689

Sandia National Laboratories

Operated for the U.S. Department of Energy by
Sandia Corporation

Carlsbad Programs Group

date: 13 January 2009
to: Records
from: Ahmed E. Ismail
Haoran Deng
Je-Hun Jang
Thomas J. Wolery (LLNL)
subject: Verification of FMT database and conversion to EQ3/6 format

As part of the work for AP-140, *Analysis Plan for EQ3/6 Analytical Studies* (Wolery 2008), it was necessary to convert the existing Pitzer parameter database used in FMT to a format suitable for use with EQ3/6. In addition, it was decided to verify all Pitzer parameters in the database against the original literature sources to ensure the accuracy of the database that would be converted to EQ3/6 format.

Database Conversion

The conversion of the database involved multiple steps.

1. The first step in the process was the construction of an Excel spreadsheet, *DBConversion050405_Rev3.xls*. The spreadsheet takes for each species listed in the FMT_050405.CHEMDAT database the species name (as listed in FMT) and its chemical potential. The species were then divided into aqueous and solid species. Then, given the set of strict basis species and auxiliary basis species used by EQ3/6, reactions were constructed for the formation of each species from the appropriate set of basis species. Then, using the chemical potential information available for each species, a log K was determined. (The value of log K for each species is determined, whether or not is needed; where the value is not appropriate for use, it has been so indicated in the Excel spreadsheet.) The final output of this process is contained on the "Summary" spreadsheet, which was also exported as a CSV file, *DbConversion050405Summary.csv*.

   The conversion process included all of the required solid and aqueous species. For consistency, the fictitious species PosIon and NegIon were included. In addition, data from the literature (Shock et al. 1989), as reported in the *data.ymp* database included with EQ3/6, was used to provide equilibrium data for O_2(aq) and H_2(aq), which were needed for handling the species balances.

2. To complete the conversion process for species, a Perl script, *read_db.pl*, was run on the *DBConversion050405Summary.csv* file. This script (a code listing of which is shown in the Appendix) performs a different set of operations for each group of species present in the summary database:

   a. For strict basis species, the script uses the formula of each species to determine the charge of the species as well as how many atoms of each element are present.

b. For auxiliary basis species and complexes, in addition to the information for strict basis species, the stoichiometry of the aqueous dissociation reaction is reported. The log *K* value determined is reported where available; otherwise "No Data" is reported.
c. For solid species, all of the information except charge (which is zero) is reported. An additional blank, the "V0PrTr" header, is included but left equal to zero as it is not needed for WIPP calculations.

The result of executing this script is a text file, *species.dat*, which can be entered into an EQ3/6 database file. The accuracy of the conversion procedure was determined by visual inspection of the input and output files.

3. A second script, *chemdat_convert.pl*, was written to extract the Pitzer parameters from the FMT database. For each type of Pitzer parameter, the *FMT_050405.CHEMDAT* database was read to locate all nonzero parameters. For each nonzero parameter, a data block for the appropriate type of ion was constructed. Although EQ3/6 uses a temperature-dependent model for its parameters, the WIPP geochemistry model assumes a constant temperature, and thus the "a2," "a3," and "a4" entries for each parameter were set to zero. The output of this script is a text file, *fmt_convert.dat*, whose entries can be cut and pasted into an EQ3/6 database file. The accuracy of the conversion procedure was determined by visual inspection of the input and output files.

   The only conversion of parameters that was inserted into the script was the handling of the α_1 and α_2 parameters describing the interaction between 1:1, 1:2, and higher electrolytes. The FMT database contains a field indicating "1" for a 1:1 or 2:1 electrolyte ratio (for ion pairs in which at least one ion has charge 1, and the other has charge 1 or 2), a "2" for a 2:2 electrolyte pair (both ions have charge 2), and a "3" for higher ratios (one or more ions has a formal charge of 3 or larger). EQ3/6 uses the explicit values for these parameters; the field is read by *chemdat_convert.pl*, and the appropriate values, which are given in Pitzer (1991), are inserted into the output file.

4. To construct the output EQ3/6 database, data0.wipp, an existing database (data0.ymw) which comes as part of the installation distribution of EQ3/6 was used as a template. The new information was copied from the *species.dat* and *fmt_convert.dat* files into the new file, replacing any data in the original file. (Each section was copied individually, as the order in which FMT and EQ3/6 database files are processed is different.) Following this, some minor corrections and adjustments were made to the database, and source information for the various parameters was entered manually. These changes are described in the following section. The final output file created by this process is *data0.wipp*.

Code listings for the scripts *read_db.pl* and *chemdat_convert.pl* are given in the Appendix. All of the files created during the conversion process (*read_db.pl*, *chemdat_convert.pl*, *DBConversion050405_Rev3.xls*, *DBConversion050405Summary.csv*, *species.dat*, *fmt_convert.dat*, *SpeciesMap.dat*, and *data0.wipp*) will be placed in a ZIP archive file, *fmt_convert.zip*, which will be stored in library LIBEQ36, class CONVERT in the WIPP CMS repository. The file SpeciesMap.dat was manually generated using the information provided in the CHEMDAT database.

The work was performed on a MacBook Pro running OS X, version 10.4.11, with Perl version 5.8.6. The scripts should be compatible with any Perl installation version 5.8 or higher. To run the scripts on a Unix- or Linux-based system, change to the directory containing the Perl script and execute the command "./read_db.pl" or "./chemdat_convert.pl" at a prompt. Note that the supporting

files (*DBConversion050405Summary.csv* for *read_db.pl*, and *FMT_050405.CHEMDAT* and *SpeciesMap.dat* for *chemdat_convert.pl*) must be included in the same directory in order for the scripts to execute.

Database Verification

All Pitzer parameters were verified with the original literature citations provided in the FMT database. All necessary values for the WIPP geochemistry model—including interaction parameters for the major brine component species, as well as interactions involving the Am(III), Th(IV), and Np(V) actinide species which are used as the basis for determining the solubilities of all An(III), An(IV), and An(V) species present in the WIPP—were matched against the original literature citations.

All values matched those found in the original literature sources. During the matching process, however, several typographical errors were found in the original sources:

1. For parameters involving 2:1 ionic charge ratios taken from Pitzer (1991), there is an error in the column defining the C^{ϕ} parameter. According to the definition of C^{ϕ} provided in Harvie et al. (1984), the correct relationship between C^{ϕ} and C should be $\frac{2^{5/2}}{3}C^{\phi} = \frac{16}{3}C$ instead of the listed relationship $2 \times \frac{2^{5/2}}{3}C^{\phi} = \frac{16}{3}C$. No values in the FMT database required correction based on this problem.

2. In a report prepared for Sandia by Choppin et al. (1999), a number of cation-anion pairs were incorrectly listed as anion-anion pairs in Table 44 (that is, the anions in question are listed as being paired with Cl^- instead of Na^+). However, comparison with the original source literature from which the table is distilled shows the pairs correctly listed as having Na^+ as a cation instead of Cl^- as a second anion. Again, no values in the FMT database required correction as a result of this problem.

3. In addition to the above issues, it was also observed that the database contained entries for the U(IV) and Pu(IV) speciation models. However, such models are not currently included in WIPP calculations, which are based on models for Am(III), Th(IV), and Np(V). Consequently, the uranium and plutonium species were removed from the database following the completion of the conversion process.

4. During reviews of this document, it was discovered that there was an error in the conversion process for handling the conversion of the neutral-cation-anion Pitzer parameters. The row indicator was improperly incremented in the original version of the file, leading to incorrect identification of the cation. This has been corrected in the code listing shown below; because only two parameters were affected, and the mistake was detected in the review process, the changes ($MgOH^+$ to Na^+ and $AmCO_3^+$ to H^+) were made manually.

References for all of the Pitzer parameters were manually entered into the data0.wipp file after the verification process was complete.

REFERENCES

Choppin, G. R., A. H. Bond, M. Borkowski, M. G. Bronikowski, J. F. Chen, S. Lis, J. Mizera, O. Pokrovsky, N. A. Wall, Y.-X. Xia, and R. C. Moore. 2001. Waste Isolation Pilot Plant Actinide Source Term Test Program: Solubility Studies and Development of Modeling Parameters. SAND99-0943. Carlsbad, NM: Sandia National Laboratories.

Harvie, C. E., N. Møller, and J. H. Weare. 1984. The prediction of mineral solubilities in natural waters: The Na-K-Mg-Ca-H-Cl-SO_4-OH-HCO_3-CO_3-CO_2-H_2O system to high ionic strengths at 25°C. *Geochim. et Cosmochim. Acta*, **48**, 723.

Pitzer, K. S. 1991. *Activity Coefficients in Electrolyte Solutions*, 2nd ed. Boca Raton, FL: CRC Press.

Shock, E. L., H. C. Helgeson, and Sverjensky, D. A. 1989. Calculation of the thermodynamic and transport properties of aqueous species at high pressurs and temperatures: Standard partial molal properties of inorganic neutral species. *Geochim. et Cosmochim. Acta*, 53, 2157.

Wolery, T. B. 2008. *Analysis Plan for EQ3/6 Analytical Studies*. Analysis Plan AP-140. Carlsbad, NM: Sandia National Laboratories.

Appendix: Code Listings

read_db.pl

```
#!/usr/bin/perl

# A. E. Ismail
# 25 July 2008

# read_db.pl

# This script takes a CSV file containing thermodynamic information on
# the compounds to be used in the EQ3/6 database. For each species in
# the file, the number of atoms of each element is determined,
# and the log K information is constructed. The results are returned
# in the form required for an EQ3/6 data file.
#
# The output of this program is manually edited in conjunction with
# the results of chemdat_convert.pl to create the overall file to be
# processed by EQ3/6.

use strict;

my $spacer =
  "+------------------------------------------------------------------\n";
my $item = "****\n";
my $pad = "                          ";

open IN, "<DbConversion050405Summary.csv" or
die "Could not open database file.\n";
open OUT, ">species.dat" or die "Could not open output file.\n";

# Throw away strict basis species header
<IN> for 1 .. 5;

print OUT $spacer;
print OUT "basis species\n";

# Now parse basis species
while (<IN>) {
  print OUT $spacer;
  my @entries = split /,/;
  last unless ($entries[0] =~ /[A-Za-z0-9]/);
  print OUT $entries[0]."\n";

  &get_charge($entries[1]);
  &parse_elem($entries[1]);
}

# Throw away auxiliary basis header
<IN> for 1 .. 2;

print OUT "auxiliary basis species\n";

while (<IN>) {
  print OUT $spacer;
  my @entries = split /,/;
  last unless ($entries[0] =~ /[A-Za-z0-9]/);
  printf OUT "%.24s  %.24s\n", $entries[0].$pad, $entries[1];

  &get_charge($entries[1]);
  &parse_elem($entries[1]);
  &reaction($entries[3]);
  &logK($entries[2]);
}

# Throw away complexes header
<IN> for 1 .. 2;

print OUT "complexes\n";

while (<IN>) {
```

```
  print OUT $spacer;
  my @entries = split /,/;
  last unless ($entries[0] =~ /[A-Za-z0-9]/);
  printf OUT "%.24s  %.24s\n", $entries[0].$pad, $entries[1];

  &get_charge($entries[1]);
  &parse_elem($entries[1]);
  &reaction($entries[3]);
  &logK($entries[2]);
}

# Throw away solids header
<IN> for 1 .. 2;

print OUT "solids\n";

while (<IN>) {
  print OUT $spacer;
  my @entries = split /,/;
  last unless ($entries[0] =~ /[A-Za-z0-9]/);
  printf OUT "%.24s  %.24s\n", $entries[0].$pad, $entries[1];

  &get_volume();
  &parse_elem($entries[1]);
  &reaction($entries[3]);
  &logK($entries[2]);
}

print OUT $spacer;

close IN;
close OUT;


sub get_charge {

  my $entry = $_[0];

  # Need to determine charge

  if (substr($entry, -1) =~ /[+-]/) {
    # There's a charge, find it.
    my $charge = 1;
    while () {
      ++$charge;
      last unless (substr($entry, -$charge) =~ /[+-]{$charge}/);
    }
    if (substr($entry, -1) =~ /-/) {
      printf OUT "     charge  =  %4.1f\n", -(--$charge);
    } else {
      printf OUT "     charge  =  %4.1f\n", --$charge;
    }
  } else {
    printf OUT "     charge  =   0.0\n";
  }
  printf OUT $item;
}


sub get_volume {

  print OUT "     V0PrTr  =   000.000 cm**3/mol [source:         ]\n";
  print OUT $item;

}


sub parse_elem
{
  my $mult = 1.0;

  # First, make a copy of the string to be examined.
  my $work = $_[0];
```

```
# The next step is to eliminate strings that will not contribute
# to the element count.
my @elim_list = ("(5-)", "(6-)", "(aq)", "(g)", "+", "-");
for my $item (@elim_list) {
  $work =~ s/\Q$item\E//g;
}

# Now build the list of elements present in the species.
my %elem_list;
for my $j (0 .. length($work) - 1) {
  my $elem = substr $work, $j, 1;
  next unless ($elem =~ /[A-Z]/);
  my $second = substr $work, $j + 1, 1;
  $elem = substr $work, $j, 2 if ($second =~ /[a-z]/);
  $elem_list{$elem} = 0;
}

# Check if the whole compound is surrounded by brackets.
# If so, then the prefactor has to be taken into account.
if ($work =~ /\[/) {
  my @array = split ("\Q\[\E", $work);
  $mult = $array[0];
  $array[1] =~ s/\]//;
  $work = $array[1];
}

# The next step is to break up the formula into pieces.
# The different pieces are separated by periods.
my @pieces;
if ($work =~ /\./) {
  @pieces = split(/\./, $work);
} else {
  @pieces = ($work);
}

# Now we work on the individual pieces.
for my $block (@pieces) {
  my $factor = 1.0;

  # Check blocks for the presence of prefactors.
  if (substr($block, 0, 1) =~ /[1-9]/) {
    my @numbers = split(/[A-Za-z]/, $block);
    $factor = $numbers[0];
    $block = substr $block, length($factor);
    if ($factor =~ /\//) {
      my @terms = split(/\//, $factor);
      $factor = $terms[0]/$terms[1];
    }
  }

  # Now we need to analyze the remaining chunks and check
  # for parentheses. We'll begin by splitting on left
  # parentheses.
  my $full = "";
  if ($block =~ /\(/) {
    my @crumbs = split(/\(/, $block);
    # Now check the crumbs for right parentheses.
    for my $l (@crumbs) {
      my $string;
      if ($l =~ /\)/) {
        my @grains = split(/\)/, $l);
        my $reps = substr $grains[1], 0, 1;
        $grains[1] = substr $grains[1], 1;
        push @grains, $grains[0] for (2 .. $reps);
        $string = join("", @grains);
      } else {
        $string = $l;
      }
      $full = $full.$string;
    }
  } else {
    $full = $block;
  }
```

```
    for my $n (0 .. length($full) - 1) {
      # Not an element if it's a number or a lowercase
      next if (substr($full, $n, 1) =~ /[a-z0-9]/);
      # But it could be a two-letter element
      my $elem;
      if (substr($full, $n + 1, 1) =~ /[a-z]/) {
        $elem = substr $full, $n, 2;
      } else {
        $elem = substr $full, $n, 1;
      }
      if (substr($full, $n + length($elem), 1) =~ /[2-9]/) {
        my $count = substr($full, $n + length($elem), 1);
        $elem_list{$elem} += $factor * $count;
      } elsif (substr($full, $n + length($elem), 1) =~ /1/) {
        # If the digit's a 1, it must be a two-digit number
        my $count = substr($full, $n + length($elem), 2);
        $elem_list{$elem} += $factor * $count;
      } else {
        $elem_list{$elem} += $factor;
      }
    }
  }

  $elem_list{$_} *= $mult for (keys %elem_list);

  # We can proceed without problems if there are no parentheses
  # in the remaining formula. Check for this first.

  my $elems = scalar keys %elem_list;

  # Output results to file
  printf OUT "     %1d element(s): \n", $elems;
  my $elem_count = 0;
  print OUT "     ";
  for my $k (sort keys %elem_list) {
    printf OUT "%7.4f %-14s", $elem_list{$k}, $k;
    print OUT "\n     " if (++$elem_count % 3 == 0 && $elem_count != $elems);
  }
  print OUT "\n";
  print OUT $item;
}


sub reaction {

  my $reaction = $_[0];
  my $ns = 0;
  my (@names, @moles);

  # The reaction is split according to the equal sign.
  my @parts = split / = /, $reaction;

  # Start with reactants
  my @reactants = split / \+ /, $parts[0];
  for my $i (@reactants) {
    ++$ns;
    my @comps = split / /, $i;

    $names[$ns] = (@comps == 2) ? $comps[1] : $comps[0];
    $moles[$ns] = (@comps == 2) ? -$comps[0] : -1;
  }

  my @products = split / \+ /, $parts[1];
  for my $i (@products) {
    ++$ns;
    my @comps = split / /, $i;

    $names[$ns] = (@comps == 2) ? $comps[1] : $comps[0];
    $moles[$ns] = (@comps == 2) ? $comps[0] : 1;
  }

  printf OUT "    %2d species in aqueous dissociation reaction:\n", $ns;
  for my $i (1 .. $ns) {
    printf OUT "   %8.4f  %.21s", $moles[$i], $names[$i].$pad;
```

```
    print OUT "\n" if (($i % 2 == 0) && ($i != $ns));
  }

  print OUT "\n";
  print OUT "*\n";
}


sub logK {
  my $logK = $_[0];
  my $nd = "No_Data";

  print OUT
    "**** logK grid [0-25-60-100C \@1.0132bar; 150-200-250-300C \@Psat-H2O]:\n";
  if ($logK != "No_Data" && $logK != "99999") {
    printf OUT "       %8s  %8.4f  %8s  %8s\n", $nd, $logK, $nd, $nd;
  } else {
    printf OUT "       %8s  %8s  %8s  %8s\n", $nd, $nd, $nd, $nd;
  }
  printf OUT "       %8s  %8s  %8s  %8s\n", $nd, $nd, $nd, $nd;
  print OUT "* Source: \n";
}
```

chemdat_convert.pl

```
#!/usr/bin/perl

# A. E. Ismail
# 25 July 2008

# chemdat_convert.pl

# This script takes a slightly modified version of the CHEMDAT FMT
# database and exports it to a format suitable for use in EQ3/6. The
# only changes made to the CHEMDAT file are the insertion of headers
# for the different parameter sections in the file, as well as
# removing line breaks found in the middle of long data entries.

use strict;

my $spacer =
  "+------------------------------------------------------------------------\n";
my $cspacer =
  "*------------------------------------------------------------------------\n";
my $item = "****\n";
my $pad = "                          ";

my %orgs = ("Cit" => "Citrate", "Ac" => "Acetate", "Lac" => "Lactate",
            "Ox" => "Oxalate");

my @alpha1 = (0.0, 2.0, 1.4, 1.4);
my @alpha2 = (0.0, 12.0, 12.0, 50.0);
my (@cations, @anions, @neutrals);

# Determine species, and count the number of each type
&read_species();
my $nc = $#cations;
my $na = $#anions;
my $nn = $#neutrals;

# We also need variables to keep track of array positions
my ($block, $row, $col);

open IN, "<FMT_050405.txt" or die "Could not open CHEMDAT file.\n";
open OUT, ">fmt_convert.dat" or die "Could not open output file.\n";

print  OUT $spacer;
print  OUT
  "ca combinations: beta(n)(ca) and Cphi(ca) [optional: alpha(n)(ca)]\n";

while (<IN>) {
  (<IN>, last) if /cation-anion/i;
}


# Handle cation-anion interactions
while (<IN>) {

  last if /cation-cation/i;

  my @array = split;
  next if (!@array);
  next if ($array[1] == 0 && $array[2] == 0 &&
           $array[3] == 0 && $array[4] == 0);

  for my $org (keys %orgs) {
    $array[5] =~ s/$org/$orgs{$org}/;
    $array[6] =~ s/$org/$orgs{$org}/;

    # Also change the handling of the negative charges from
    # CHEMDAT form to EQ3/6 form.
    $array[6] =~ s/===/\(6-\)/;
    $array[6] =~ s/==-/\(5-\)/;
    $array[6] =~ s/=/--/;
  }
```

```
  my $id1 = $array[5].$pad;
  my $id2 = $array[6].$pad;
  # We will define the @beta array as (beta(0), beta(1), beta(2),
  # cphi) to make life easier.
  my @beta;
  my @bstr = ("  beta(0)\n", "  beta(1)\n", "  beta(2)\n", "  Cphi:\n");
  $beta[$_ - 1] = $array[$_] for 1 .. 4;

  print  OUT $spacer;

  printf OUT "%.24s  %.24s\n", $id1, $id2;
  printf OUT "  alpha(1) = %-.1f\n", $alpha1[$array[0]];
  printf OUT "  alpha(2) = %-.1f\n", $alpha2[$array[0]];
  for (0 .. 3) {
    printf OUT $bstr[$_];
    printf OUT "     a1 = %.6g\n", $beta[$_];
    print  OUT "     a2 = 0.\n";
    print  OUT "     a3 = 0.\n";
    print  OUT "     a4 = 0.\n";
  }
  print  OUT "* Source:\n";
}

# Handle cation-cation interactions next
print  OUT $spacer;
print  OUT "cc' and aa' combinations: theta(cc') and theta(aa')\n";
print  OUT $cspacer;
print  OUT "* cation-cation\n";

$row = 0;
while (<IN>) {

  last if /anion-anion/i;
  my @array = split;
  next if (!@array);

  ++$row;
  my $col = $row;
  while () {
    ++$col;
    my $term = shift @array;
    last if ($term =~ /[A-Za-z]/);
    next if ($term == 0.0);

    &print_block("theta", $cations[$row], $cations[$col], "", $term);
  }
}


# Handle anion-anion interactions next
print  OUT $cspacer;
print  OUT "* anion-anion\n";

$row = 0;
while (<IN>) {

  last if /cation-cation-anion/i;
  my @array = split;
  next if (!@array);

  ++$row;
  my $col = $row;
  while () {
    ++$col;
    my $term = shift @array;
    last if ($term =~ /[A-Za-z]/);
    next if ($term == 0.0);

    &print_block("theta", $anions[$row], $anions[$col], "", $term);
  }
}

# Now comes the tricky part: ternary parameters!
# We need to keep track of block, row, and column!
```

```
# Start with the cation-cation-anion parameters
print  OUT $spacer;
print  OUT "cation-cation-anion\n";

$block = 0;
while (<IN>) {

  last if /cation-anion-anion/i;
  my @array = split;
  if (!@array) {
    ++$block;
    $row = $block;
    next;
  }

  ++$row;
  $col = 0;
  while () {
    ++$col;
    my $term = shift @array;
    last if ($term =~ /[A-Za-z]/);
    next if ($term == 0.0);

    &print_block("psi", $cations[$block], $cations[$row], $anions[$col],
                 $term);
  }
}

# Next, tackle the cation-anion-anion parameters
print  OUT $spacer;
print  OUT "cation-anion-anion\n";

$block = 0;
while (<IN>) {

  last if /neutral-cation/i;
  my @array = split;
  if (!@array) {
    ++$block;
    $row = $block;
    next;
  }

  ++$row;
  $col = 0;
  while () {
    ++$col;
    my $term = shift @array;
    last if ($term =~ /[A-Za-z]/);
    next if ($term == 0.0);

    &print_block("psi", $anions[$block], $anions[$row], $cations[$col],
                 $term);
  }
}


# Neutral binary parameters are next. They're a little simpler
# since there's only one block per section.

# Neutral-cation first
print  OUT $spacer;
print  OUT "nc and na combinations: lambda(nc) and lambda(na)\n";
print  OUT $cspacer;
print  OUT "* neutral-cation\n";

$row = 0;
while (<IN>) {

  last if /neutral-anion/i;
  my @array = split;
  next if (!@array);
```

```
    ++$row;
    $col = 0;
    while () {
      ++$col;
      my $term = shift @array;
      last if ($term =~ /[A-Za-z]/);
      next if ($term == 0.0);

      &print_block("lambda", $neutrals[$row], $cations[$col], "", $term);
    }
  }

  # Neutral-anion next
  print  OUT $cspacer;
  print  OUT "* neutral-anion\n";

  $row = 0;
  while (<IN>) {

    last if /neutral-cation-anion/i;
    my @array = split;
    next if (!@array);

    ++$row;
    $col = 0;
    while () {
      ++$col;
      my $term = shift @array;
      last if ($term =~ /[A-Za-z]/);
      next if ($term == 0.0);

      &print_block("lambda", $neutrals[$row], $anions[$col], "", $term);
    }
  }


  # Time for neutral-cation-anion parameters now
  print  OUT $spacer;
  print  OUT "nca combinations: zeta(nca)\n";

  $block = 0;
  while (<IN>) {

    last if /cation map/;
    my @array = split;
    if (!@array) {
      ++$block;
      next;
    }

    ++$row;
    $col = 0;
    while () {
      last if (++$col > $nc);
      my $term = shift @array;
      last if ($term =~ /[A-Za-z]/);
      next if ($term == 0.0);

      &print_block("zeta", $neutrals[$block], $cations[$row], $anions[$col],
                   $term);
    }
  }

  close IN;
  close OUT;


  sub read_species {

    # Open species map

    open SPEC, "<SpeciesMap.dat" or die "Could not open species map.\n";

    # Parse cations first
```

```
  <SPEC> for 1 .. 2;
  my $nc = 0;
  while (<SPEC>) {
    my @line = split;
    last if (!@line);

    while (@line) {
      $cations[++$nc] = shift (@line);
    }
  }


  # Parse anions next

  <SPEC> for 1 .. 2;
  my $na = 0;
  while (<SPEC>) {
    my @line = split;
    last if (!@line);

    while (@line) {
      $anions[++$na] = shift (@line);
    }
  }


  # Parse neutral species last

  <SPEC> for 1 .. 2;
  my $nn = 0;
  while (<SPEC>) {
    my @line = split;
    last if (!@line);

    while (@line) {
      $neutrals[++$nn] = shift (@line);
    }
  }

  close SPEC;

}


sub print_block {

  my $type = $_[0];
  my $id1  = $_[1].$pad;
  my $id2  = $_[2].$pad;
  my $id3  = $_[3].$pad;
  my $term = $_[4];

  print  OUT $spacer;

  printf OUT "%.24s  %.24s  %.24s\n", $id1, $id2, $id3;
  print  OUT "  $type:\n";
  printf OUT "    a1 = %.6g\n", $term;
  print  OUT "    a2 = 0.\n";
  print  OUT "    a3 = 0.\n";
  print  OUT "    a4 = 0.\n";
  print  OUT "* Source:\n";
}
```