GoldSim Dynamic-Link Library (DLL) Interface for Cementitious Barriers Partnership (CBP) Code Integration – 11444

Kevin G. Brown*, Frank Smith** and Gregory Flach**
*Vanderbilt University, School of Engineering, CRESP III, Nashville, TN 37235
**Savannah River National Laboratory, Aiken, SC 29808

Keywords: simulation, dynamic-link library, DLL, code integration, Cementitious Barriers Partnership

ABSTRACT

The Cementitious Barriers Partnership (CBP) Project is a multi-disciplinary, multi-institutional collaboration supported by the United States Department of Energy (US DOE) Office of Waste Processing. The objective of the CBP project is to develop a set of tools to improve understanding and prediction of the long-term structural, hydraulic, and chemical performance of cementitious barriers used in nuclear applications. The tools developed under this project have been used to evaluate and predict the behavior of cementitious barriers used in near-surface engineered waste disposal systems, e.g., waste forms, containment structures, entombments, and environmental remediation, including decontamination and decommissioning analysis of structural concrete components of nuclear facilities. The periods of cementitious performance being evaluated are up to or longer than 100 years for operating facilities and longer than 1000 years for waste management.

The CBP project is focused on reducing the uncertainties of current methodologies for assessing cementitious barrier performance and increasing the consistency and transparency of the assessment process. To better characterize the uncertainties in the models used to predict barrier performance, GoldSim is used as a probabilistic framework with interfaces to external codes for specific calculations. A general dynamic-link library (DLL) interface has been developed to link GoldSim with external codes. The DLL that performs the linking function is designed to take a list of code inputs from GoldSim, create an input file for the external application, run the external code, and return a list of outputs, read from files created by the external application, back to GoldSim for analysis. Although currently used by CBP, the DLL is generic and can be used for a wide variety of external codes that need to be examined probabilistically. Use of the DLL to couple external codes to GoldSim helps enable improved risk-informed, performance-based decision-making and supports several of the strategic initiatives in the DOE Office of Environmental Management Engineering & Technology Roadmap.

INTRODUCTION

The GoldSim Monte Carlo simulation program can interface with external functions in three ways: a) indirectly through lookup tables, b) values passed to or through Excel spreadsheets, and c) using dynamic-link libraries (DLLs). A lookup table is designed to represent the output of an external function over a prescribed domain, but does not provide a direct link to the GoldSim model. The user instead generates tabular values (of inputs and corresponding outputs) external to GoldSim and uses the results to populate a data table element within GoldSim. At run-time GoldSim interpolates between these values as an approximation of the external function. Lookup tables are easy to implement in GoldSim but are constrained to well-behaved functions involving a small number of inputs.

GoldSim also provides a built-in capability to access cells within an Excel spreadsheet. Values in GoldSim can be extracted from or placed into cells. GoldSim can also initiate spreadsheet calculations, including initiation of Visual Basic for Applications (VBA) function calls. The latter can launch Disk Operating System (DOS) batch files, for example, providing a general capability to parameterize and execute external codes. However, the GoldSim must link to the external code(s) through several processes that may introduce run-time vulnerabilities, e.g., opening and modifying a spreadsheet during GoldSim runtime.

The DLL interface is the more general, streamlined, and robust interface to external codes from GoldSim. However, the GoldSim user is responsible for creating the interface functionality through custom programming

that follows a GoldSim Application Program Interface (API). To minimize the burden placed on the CBP user, a generic DLL and accompanying "DLL instruction" language are provided by GoldSim. Low-level interface coding is hidden from the CBP user, who needs only specific knowledge as to where to place and retrieve values in external application-specific I/O (input/output) files, as described in the following section.

CBP DYNAMIC-LINK LIBRARY (DLL) DESIGN

The DLL code described in this paper was written in Fortran 90 (compiled using the g95 compiler available at http://www.gnu.org/) and consists of five files and 20 subroutines. A more detailed explanation for each subroutine is provided elsewhere by Smith et al. [1]. Operation of the DLL interface is controlled by instructions provided by the user in a simple text file, typically denoted `DLL.dat`. An example instructions file is shown in Fig. 1. This instructions file is tab delimited (required format) into 13 fields. The first field starts in the first column. Continuation lines are marked by an "`&`" in the first column; comments are indicated by an "`!`". In Fig. 1 parameters are passed to the executable file (STADIUM® in this case). The following keywords entered in Field 1 initiate actions in the DLL:

- `PUT` – Put the data specified within the block into the named file.
- `GET` – Get the data specified within the block from the named file.
- `EXE` – Perform the system calls specified within the block.
- `RPL` – Replace complete lines in the named file.
- `SUP` – Create a "super" file containing the commands or file names listed within the block.
- `LOG` – Write a log file (XML format) containing all input and output data.

A command block is terminated by the `END` statement.

Actions are processed in the order that they appear in the instructions file, and each action can be used multiple times. Typically, the user would want to change parameters in an input file (using the `PUT` command) before running the external program, run the external application that reads from the input file (using the `EXE` command), and retrieve results from the external application (using the `GET` command). The DLL performs error checking as it processes the instructions it receives.

DLL Design Assumptions

The DLL design conforms to the guidance provided by GoldSim [2] that generally specifies how the external interface must be constructed to be compatible with GoldSim's calling conventions. Within these constrictions, the user adds desired functionality using the commands listed above.

The design of the DLL assumes that the external application reads text files that supply data and control the calculations performed by the program and writes results to text output files. The formatting of the input and output files does not matter as long as the location of information in each file can be uniquely identified. Selected inputs can be modified as needed for desired calculations, and the results can be subsequently processed. The DLL interface can call external programs that can be run automatically as either an executable or through a batch file (that calls the executable) without additional user interaction.

```
!
!    #2    #3    #4    #5    #6    #7    #8    #9    #10   #11   #12   #13 (comment)
!-------------------------------------------------------------------------------
PUT   Stadium\stad09d-cbp-task7-template.inp  white
      14    row   123                       col   3                   11    1     inputs 014-024
      25    row   123                       col   4                   11    1     inputs 025-035
      45    row   138                       col   3                   9     1     inputs 045-053
      54    row   138                       col   4                   9     1     inputs 054-062
      80    row   27                        col   3                   17    1     inputs 080-096
      97    row   27                        col   4                   17    1     inputs 097-113
      114   row   12                        col   3                   2     1     inputs 114-115
      116   row   19                        col   3                   3     1     inputs 116-118
      119   row   96                        col   2                   1     1     input  119
      119   row   119                       col   2                   1     1     input  119
END
!-------------------------------------------------------------------------------
RPL   stad09d-cbp-task7-template.inp
      2     ..\..\Stadium\20cm-50cm-mesh01.cor
      4     ..\..\Stadium\20cm-50cm-mesh01.ele
END
!-------------------------------------------------------------------------------
EXE
      ..\..\..\Codes\Stadium\stadium_2009d_CBP
      &
      GUI=YES
      &
      stad09d-cbp-task7-template.inp
      &
      CBP002BATCH
      &
      stad09d-cbp-task7-template.out
END
!-------------------------------------------------------------------------------
GET   stad09d-cbp-task7-template.out.xls    space  ignore
      1     value  1.0   1     -0.1  col   4                   101   11    outputs 0001-1111
      3312  value  1.0   1     -0.1  col   18                  101   9     outputs 3312-3410
END
!-------------------------------------------------------------------------------
LOG   stadium_2layers.xml
END
!-------------------------------------------------------------------------------
```

Fig. 1. Example DLL instructions file (e.g., `DLL.dat`).

USE OF THE DYNAMIC-LINK LIBRARY

This section provides guidance to using the DLL by describing the commands that can be used in the instructions file (e.g., `DLL.dat`).

The `PUT` and `GET` Commands

The `PUT` and `GET` commands are functionally similar and are described together. As illustrated in Fig. 1, the name of the file to be processed is listed on the same line as the command in the column (next tab position) following the keyword. A keyword describing how to delimit fields in the file is provided in the column following the filename (next tab position). The following delimiter keywords (primarily corresponding to the delimiters) are recognized: `colon`, `comma`, `semicolon`, `space`, `tab`, and `white`. The keyword "`white`" describes any combination of tabs and/or spaces (appearing as "white space") used to delimit fields. The delimiter can appear only once following each field with the exception of spaces, which can be inserted multiple times. If the delimiter keyword is followed by the "`ignore`" tag, then any delimiter characters appearing before the first data entry are ignored.

Instructions are provided on the lines below that contain the filename and delimiter description. For each instruction, pertinent commands are defined using Fields 2 through 12 as described below. A command is terminated using the `END` statement.

Field 2: Input arguments are passed from GoldSim (Fig. 2) to the external interface in the `inargs()`array of double precision numbers for use by the `PUT` command. Similarly, output arguments are passed from the external interface back to GoldSim in the double precision `outargs()` array using the `GET` command. Following the Fortran 90 convention, the starting index of each array is `1`. The number in Field 2 indicates the position in the one-dimensional array representing the parameters in GoldSim (Fig. 2). Scalar inputs and outputs correspond to a single array argument, vector items are used in the order specified with one argument per element, and items in matrices are specified item-by-item, from the first row to the last [2].

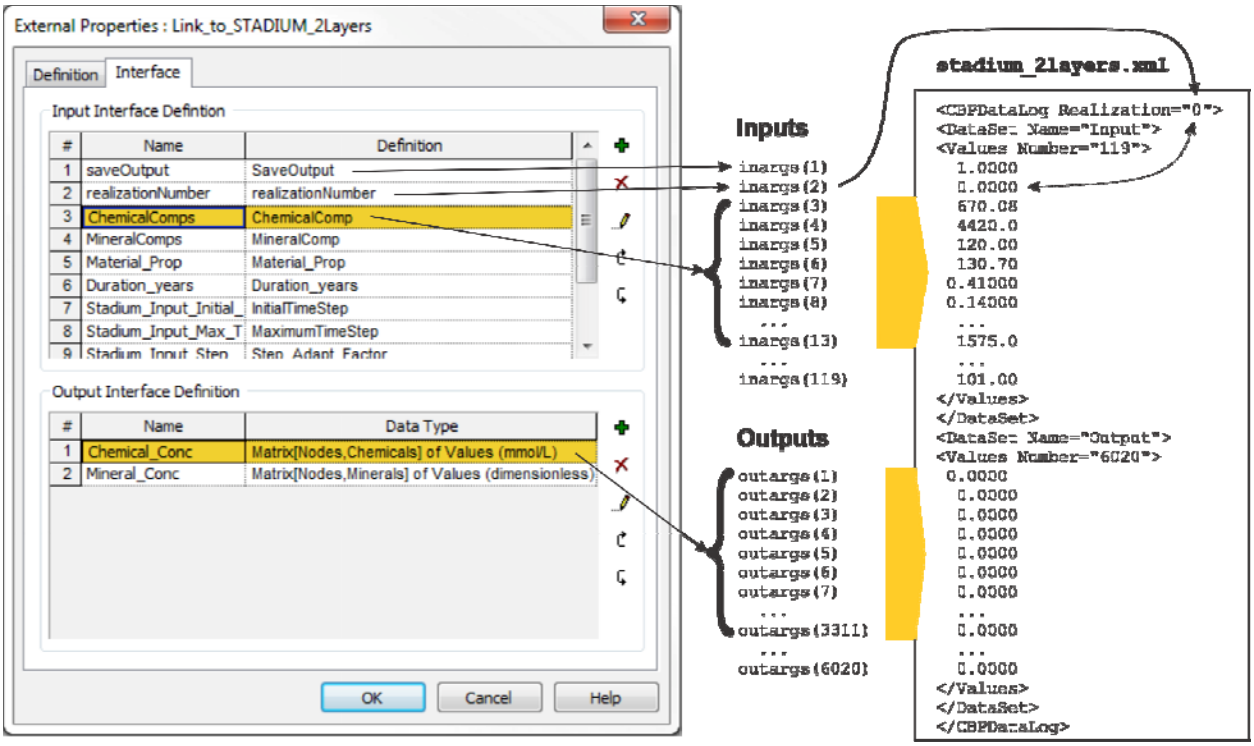


Fig. 2. Example of the GoldSim External Link Properties Interface and Corresponding Log File

GoldSim passes all values to and from the external program as double precision real numbers in the `inargs()` and `outargs()` arrays. Passing only numerical values can be restrictive because input files may contain text strings that may be desirable to change for different reasons and simulations. A simple work around to this limitation is the ability of the DLL interface to replace entire lines of input with text using the `RPL` command described below. The one exception to the GoldSim restriction of passing only double precision values is that the external function can return an error message to GoldSim in the `outargs()` array using a special GoldSim function [2].

For `PUT` commands, the first two entries in the `inargs()` array are reserved by GoldSim (as illustrated in Fig. 2) to pass two special parameters to the DLL interface:

1. The first parameter passed in `inargs(1)` is set by the user within GoldSim to indicate whether results from individual realizations are saved (`inargs(1)>0`) or not (`inargs(1)=0`). If the value in `inargs(1)` is 1, a subdirectory named `realization_0` is created, if necessary, for the results of running the external application. If GoldSim is run in probabilistic mode for multiple realizations, the results saved in the `realization_0` subdirectory would be overwritten and only results from the final realization would be saved. If `inargs(1)>0` and GoldSim is run in probabilistic mode with multiple realizations (`1` … `n`), subdirectories named `realization_1` through `realization_n+1` are created, if needed, to save results from the realizations.
2. The second parameter passed from GoldSim in `inargs(2)` is the number of the realization (i.e., the Run Property denoted *Realization* in GoldSim). GoldSim allows a single realization to also be run and analyzed.

The additional parameters used by `PUT` commands in the instructions file start with `inargs(3)`.

Field 3 – Field 6: The entries in these fields specify how the row in either the input or output file is identified. The keywords that may appear in Field 3 are listed in Table I. The information that must appear in Fields 4 – 6 depends on the keyword in Field 3 as explained in the table. The DLL reads record entries as text strings in the file starting with the first row until the conditions in Table I are satisfied. The data entry at the resulting location is then used in the `PUT` or `GET` command.

Table I. Row Specification in Field 3

Keyword	Function
`row`	A number representing the row (or record) number where the data are located is entered in Field 4. Fields 5 and 6 are left blank.
`record`	A number representing the record (or row) number where the data are located is entered in Field 4. Fields 5 and 6 are left blank.
`label`	An alphanumeric label identifying the row where the data are located is entered in Field 4, and a number representing the column where the label is to be read is entered in Field 5. Field 6 is skipped.
`value`	A number identifying the row where the data are located is entered in Field 4, the column where the label is to be read is entered in Field 5, and a numeric tolerance on the value is entered in Field 6. A positive tolerance in Field 6 specifies the absolute difference used to test if the numerical value in Field 5 has been found while a negative tolerance indicates a relative difference.
`string`	A character (alphanumeric) string identifying the row where the data is located is entered in Field 4. The string can appear in any column in the data file.

Field 7 – Field 10: These entries specify how the column in either the input or output files where the data values are located is identified. The key words that can appear in Field 7 are listed in Table II. The DLL reads entries in the specified row starting with the first column as text until the specified label is found. The data entry at this location is then used in the `PUT` or `GET` command.

Table II. Column Specification in Field 7

Keyword	Function
`col`	The number of the column (or field) where the data are located is entered in Field 8. Fields 9 and 10 are left blank.
`field`	The number of the field (or column) where the data are located is entered in Field 8. Fields 9 and 10 are left blank.
`heading`	An alphanumeric label identifying the column where the data are located is entered in Field 8, and the row where the label is located is entered in Field 9. Field 10 is skipped.
`value`	A numerical value identifying the column where the data are located is entered in Field 8, the row where the value is located is entered in Field 9, and a tolerance on the value is entered in Field 10 similar to that described in Table I for the `value` keyword.

Field 11: The number of rows to be used for data processing is entered in this field allowing entering or reading a row vector using a single command. For example, if a `PUT` command in the instructions file (e.g., `DLL.dat` in Fig. 1) has the number `n` in Field 2 and `m` in Field 11, the value in `inargs(n)` is written into the specified data file row and column identified by Fields 3 – 6 and Fields 7 – 10, respectively. The values in

`inargs(n+1)` through `inargs(n+m–1)` are also written into the next `(m–1)` rows in the same column in the specified data file.

Field 12: The number of columns to be used for data processing is specified in this field; this enables a single command to either enter a column vector or, in conjunction with Field 11 (described above), enter a matrix of values into the specified data file. For example, if a `GET` command has the number `n` in Field 2, the number `m` in Field 11, and the number `p` in Field 12, then the value in `outargs(n)` will be read from the specified data file row and column identified by Fields 3 – 6 and Fields 7 – 10, respectively. The values of `outargs(n+1)` through `outargs(n+p–1)` will be read from the next `(p–1)` columns in the same row, and the values of `outargs(n+p)` through `outargs(n+2p–1)` will be read from the same columns in the subsequent row. This process continues over `m` rows of data until `m×p` values from the data file have been read by the DLL interface.

Field 13: This field can be used to enter an optional comment that is not read or used by the DLL or by GoldSim.

The `RPL` Command

The `RPL` command can be used to replace entire lines in an existing input file. This command provides a simple work around to the GoldSim limitation of only passing only double precision numbers, the DLL interface was given the capability of replacing entire lines of input with text using the `RPL` command. Within the `RPL` command block, the entry in Field 2 identifies the line in the input file that will be replaced. The text starting in Field 3 is used to replace the current line in the input file.

The `EXE` Command

The `EXE` command block specifies DOS commands to be executed by the Windows operating system. Typically the name and relative location of the file (or files) that must be run to execute the external code are provided in this command. Any arguments that must be passed to the executable are included in the command.

The `EXE` command can also be used to execute a Windows batch file (i.e., one with the `.bat` extension) that contains the necessary executable file(s). Additional commands can be specified to perform other operations, such as copying, renaming, or moving input and output files as needed.

The `SUP` Command

The `SUP` command writes the given text to the specified file. The filename must be provided in the second column on the same line as the `SUP` command. The file is created if it is not found or overwritten if it already exists. For example, this command can be used to create a "super" file of instructions including necessary filenames that can then be accessed by the external application at runtime.

The `LOG` Command

The `LOG` command provides the name of the log file where arrays of the GoldSim input and output data used in the simulation will be written in XML format as shown in Fig. 2. The log filename must be provided in the second column on the same line as the `LOG` command. As also shown in the figure, the realization number is written to the log file followed by an array of the input data and an array of the output data.

EXAMPLE OF USING THE DYNAMIC LINK-LIBRARY (DLL)

A detailed example of calling an external code is provided to demonstrate the usefulness of the DLL developed by the CBP. SIMCO Technologies, Inc. developed the numerical model called STADIUM® (Software for Transport and Degradation in Unsaturated Materials) to predict the transport of ions and liquids in reactive porous media [3-5]. This state-of-the-art model has been successfully used to predict the degradation of

unsaturated concrete structures exposed to chemically aggressive environments [3, 6]. The results provided by STADIUM® have been validated using both laboratory test results and field exposure observations.

GoldSim is run using the instructions file in Fig.1 to produce three realizations of calculations using the STADIUM code, each for one year of simulation time [1]. Because the DLL replaces values in the input file specified by the `PUT` command, the user should supply a template input file for the DLL to manipulate. As shown previously, the DLL provides flexibility for locating parameters and data within the template file. The user must either know the locations of input parameters or data to be manipulated within the input file or know the basic input structure so that the methods described previously can be used.

The example instructions file (`DLL.dat` in Fig. 1) locates the parameters to be input using row and column specifications. The main portions of the STADIUM template input file (e.g., `RESO`, `PREL`, and `INIT` blocks) that are changed by the instructions file are shown in Fig.3. After the DLL executes the `RPL` and `PUT` commands shown in Fig. 1, the modified parts of the template file that was used as the input file for STADIUM are shown in Fig. 4 in yellow.

```
COOR
20cm-50cm-mesh01.cor
ELEM
20cm-50cm-mesh01.ele

RESO
    NUMBER_NUM_PARAM.   14
    integration_pts     2
    tolerance           1.0e-3
    itermax             30
    cartesian_axi       1.0
    Duration_years      10000.0
    Init_time_step_sec  5000.0
    f_sat               3.0
    Tangential_matrix   0.0
    damage              1.0
    physical_cl         0.0
    CO2_level_%         0.0
    Max_time_step_sec   4320000.0
    Step_Adapt_Factor   1.5
    Step_Adapt_Crit     5e-3

PREL
    N_PREL_GROUP        2
    N_PREL              18
    temperature         23.0            23.0
    W/B                 0.38            0.595
    Binder              405.0           930.0
    aggregates          1659.0  0.0
    Binder_density      2885.0  2603.5
    Porosity            0.135           0.65
    Permeability        18.0e-22        4000.0e-22
    oh_diff_coef        1.40e-11        7.5e-11
    Isotherm_b          -25.9280        -6.4651
    Isotherm_c          0.4285  1.7825
    Relative_perm       18.0            18.0
    init_hydrat         28.0            28.0
    tref_meas           28.0            28.0
    hydrat_a            0.8             0.3
    hydrat_alpha        0.015           0.003
    k_thermal           2.00            2.00
    spec_heat           1000.0  1000.0
    ex_rate_CO2         1.0e-5  1.0e-5

INIT
    external_file       0
    OH                  400.0   670.08
    Na                  282.1   4420.0
    K                   138.0   120.0
    SO4                 8.0     130.7
    Ca                  0.5     0.41
    Al(OH)4             0.1     0.14
    Cl                  5.0     9.0
    H2SiO4              0.0     9.7
    CO3                 0.0     2.9
    NO3                 0.0     2000.0
    NO2                 0.0     1575.0
    Rel_Humidity        1.0     1.0
    Potential           0.0     0.0
    Temperature         23.0    23.0

    Portlandite        13.6     41.9
    CaH2SiO4           37.9     103.3
    Ettringite          0.0     28.6
    Monosulfate        19.4     0.0
    C4AH13             14.8     0.0
    Thaumasite          0.0     0.0
    Calcite             0.0     4.8
    Monocarboaluminate  0.0     11.0
    Gypsum              0.0     0.0
```

Fig. 3. Pertinent data blocks from the STADIUM template file (i.e., the `RESO`, `PREL`, and `INIT` Blocks)

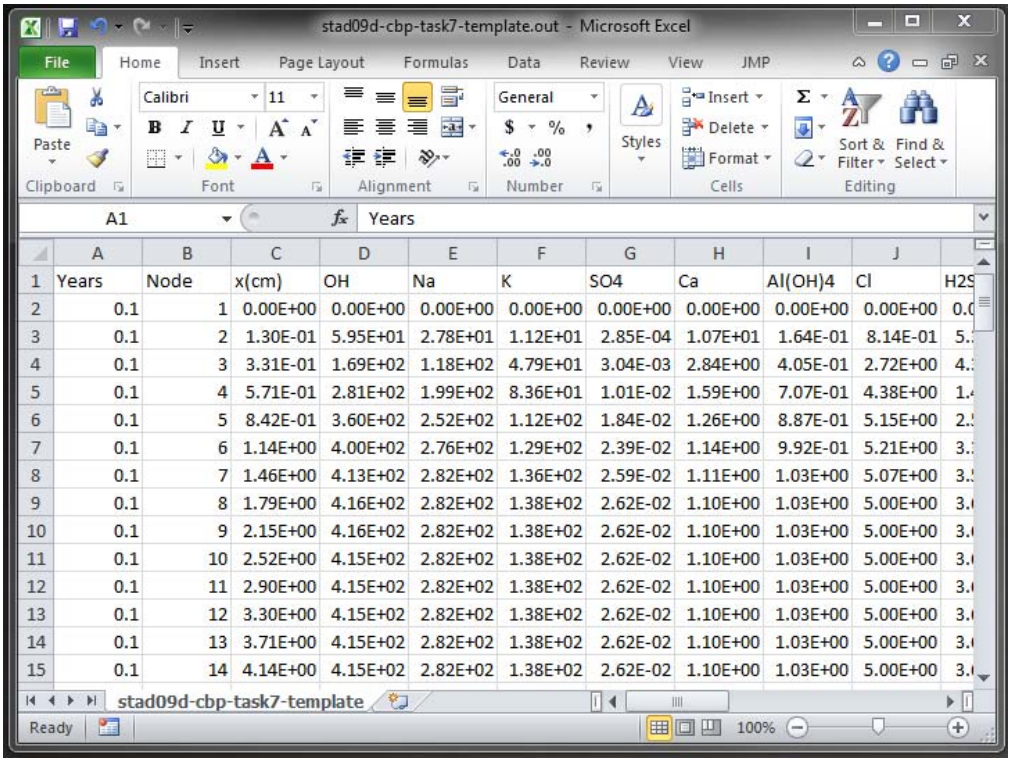
```
COOR
..\..\Stadium\20cm-50cm-mesh01.cor
ELEM
..\..\Stadium\20cm-50cm-mesh01.ele

RESO
    NUMBER_NUM_PARAM.   14
    integration_pts     2
    tolerance           1.0e-3
    itermax             30
    cartesian_axi       1.0
    Duration_years          1
    Init_time_step_sec       5000
    f_sat               3.0
    Tangential_matrix   0.0
    damage              1.0
    physical_cl         0.0
    CO2_level_%         0.0
    Max_time_step_sec    4320000
    Step_Adapt_Factor   1.50000E+00
    Step_Adapt_Crit   5.00000E-03

PREL
    N_PREL_GROUP        2
    N_PREL              18
    temperature         23          23
    W/B   3.80000E-01   5.95000E-01
    Binder        405         930
    aggregates       1659           0
    Binder_density       2885   2.60350E+03
    Porosity   1.35000E-01   6.50000E-01
    Permeability   1.80000E-21   4.00000E-19
    oh_diff_coef   1.40000E-11   7.50000E-11
    Isotherm_b  -2.59280E+01  -6.46510E+00
    Isotherm_c   4.28500E-01   1.78250E+00
    Relative_perm         18          18
    init_hydrat         28          28
    tref_meas         28          28
    hydrat_a   8.00000E-01   3.00000E-01
    hydrat_alpha   1.50000E-02   3.00000E-03
    k_thermal          2           2
    spec_heat       1000        1000
    ex_rate_CO2         1.0e-5          1.0e-5

INIT
    external_file       0
    OH        400   6.70080E+02
    Na   2.82100E+02       4420
    K        138         120
    SO4          8   1.30700E+02
    Ca   5.00000E-01   4.10000E-01
    Al(OH)4   1.00000E-01   1.40000E-01
    Cl           5           9
    H2SiO4           0   9.70000E+00
    CO3          0   2.90000E+00
    NO3          0       2000
    NO2          0       1575
    Rel_Humidity        1.0     1.0
    Potential           0.0     0.0
    Temperature         23.0    23.0

    Portlandite   1.36000E+01   4.19000E+01
    CaH2SiO4   3.79000E+01   1.03300E+02
    Ettringite           0   2.86000E+01
    Monosulfate   1.94000E+01          0
    C4AH13   1.48000E+01          0
    Thaumasite          0           0
    Calcite          0   4.80000E+00
    Monocarboaluminate          0         11
    Gypsum          0          0
```

Fig. 4. STADIUM input blocks after execution of the `RPL` and `PUT` commands in `DLL.dat` (Fig. 1)

After running STADIUM using the `EXE` command as shown in Fig.1 for the inputs defined by the user (e.g., those in Fig. 4), the external code creates a set of output files where the simulation results are written. The main output file (Fig. 5) has a `.xls` file extension (note that the file is not in Microsoft Excel format) [4]. As illustrated in Fig. 5, the results are provided in columns making either graphing the results using an external program or reading the results using the DLL straightforward. The results from different time steps are separated by a blank line. The columns of interest here are:

- A: time increment for which the solution is saved to the output file, in years or seconds.
- D – K: concentration of each ionic species, in mmol/L.
- O – W: content of each of the nine solid phases considered in the calculations, in g/kg material.

The other columns in the STADIUM output file are described in detail elsewhere [4].



	A	B	C	D	E	F	G	H	I	J	
1	Years	Node	x(cm)	OH	Na	K	SO4	Ca	Al(OH)4	Cl	H2S
2	0.1	1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.0
3	0.1	2	1.30E-01	5.95E+01	2.78E+01	1.12E+01	2.85E-04	1.07E+01	1.64E-01	8.14E-01	5.
4	0.1	3	3.31E-01	1.69E+02	1.18E+02	4.79E+01	3.04E-03	2.84E+00	4.05E-01	2.72E+00	4.
5	0.1	4	5.71E-01	2.81E+02	1.99E+02	8.36E+01	1.01E-02	1.59E+00	7.07E-01	4.38E+00	1.
6	0.1	5	8.42E-01	3.60E+02	2.52E+02	1.12E+02	1.84E-02	1.26E+00	8.87E-01	5.15E+00	2.
7	0.1	6	1.14E+00	4.00E+02	2.76E+02	1.29E+02	2.39E-02	1.14E+00	9.92E-01	5.21E+00	3.
8	0.1	7	1.46E+00	4.13E+02	2.82E+02	1.36E+02	2.59E-02	1.11E+00	1.03E+00	5.07E+00	3.
9	0.1	8	1.79E+00	4.16E+02	2.82E+02	1.38E+02	2.62E-02	1.10E+00	1.03E+00	5.00E+00	3.
10	0.1	9	2.15E+00	4.16E+02	2.82E+02	1.38E+02	2.62E-02	1.10E+00	1.03E+00	5.00E+00	3.
11	0.1	10	2.52E+00	4.15E+02	2.82E+02	1.38E+02	2.62E-02	1.10E+00	1.03E+00	5.00E+00	3.
12	0.1	11	2.90E+00	4.15E+02	2.82E+02	1.38E+02	2.62E-02	1.10E+00	1.03E+00	5.00E+00	3.
13	0.1	12	3.30E+00	4.15E+02	2.82E+02	1.38E+02	2.62E-02	1.10E+00	1.03E+00	5.00E+00	3.
14	0.1	13	3.71E+00	4.15E+02	2.82E+02	1.38E+02	2.62E-02	1.10E+00	1.03E+00	5.00E+00	3.
15	0.1	14	4.14E+00	4.15E+02	2.82E+02	1.38E+02	2.62E-02	1.10E+00	1.03E+00	5.00E+00	3.

Fig. 5. An example of the main STADIUM output file read into Microsoft Excel

The `GET` command was used (via the DLL) to retrieve concentrations of the 11 chemicals and nine minerals used by STADIUM at each node location at time one year. The `outargs()` array that returns results to GoldSim is sufficiently large to return the necessary concentrations at each time step. The GoldSim results for one selected mineral (Ettringite) and one selected chemical (SO_4) are illustrated in Fig. 6.

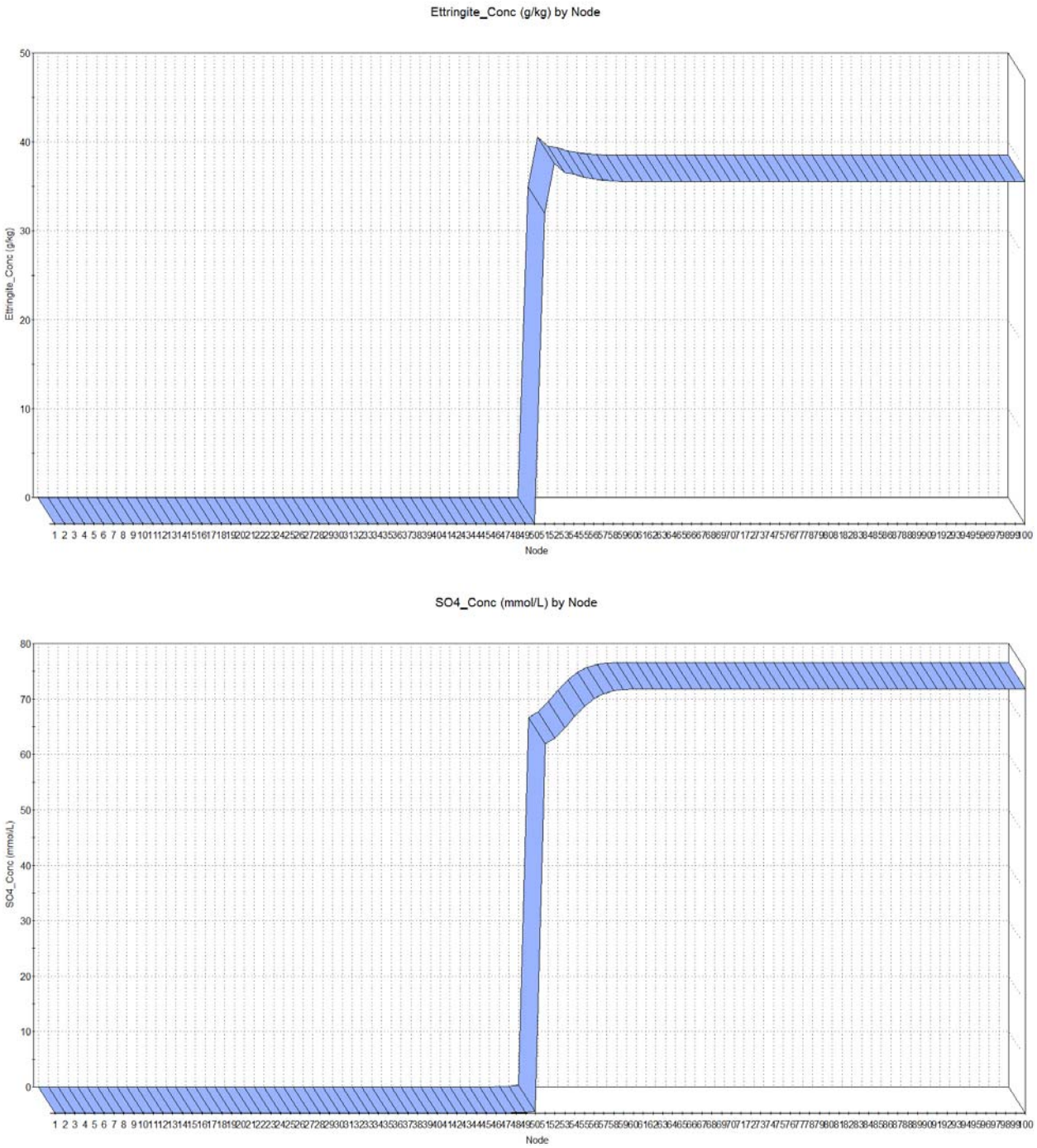


Fig. 6. GoldSim plot of STADIUM results after one year for one mineral (Ettringite) and one chemical (SO_4)

CONCLUSIONS

The CBP Project is focused on characterizing and reducing uncertainties in the current methodologies for assessing cementitious barrier performance as well as increasing the consistency and transparency of the overall assessment processes. To better represent uncertainties in the models used to predict barrier performance, GoldSim was selected as a probabilistic framework with interfaces to external computer codes for specific cementitious barrier calculations. A general dynamic-link library (DLL) interface has been

developed by the CBP to link GoldSim to external codes during runtime. The DLL is designed to take a list of code inputs from GoldSim, create an input file for the external application, run the external code, and return a list of outputs that is read from the text files created by the external application, back to GoldSim for analysis. Although currently used by CBP, the DLL can be used for a wide variety of external codes to be examined probabilistically.

The process of GoldSim calling a state-of-the-art external code (in this case, the STADIUM® model) is demonstrated to highlight the usefulness of the DLL developed by the CBP. SIMCO Technologies, Inc. developed the STADIUM model to predict the transport of ions and liquids in reactive porous media. Use of the DLL to couple external codes to GoldSim helps enable improved risk-informed, performance-based decision-making and supports several of the strategic initiatives in the DOE Office of Environmental Management Engineering & Technology Roadmap.

ACKNOWLEDGEMENTS AND DISCLAIMER

This report was prepared for the United States Department of Energy under Interagency Agreement No. DE-AI09-09SR22667 and is an account of work performed under that contract. Reference herein to any specific commercial product, process, or service by trademark, name, manufacturer, or otherwise does not necessarily constitute or imply endorsement, recommendation, or favoring of same by Savannah River Nuclear Solutions or by the United States Government or any agency thereof. The views and opinions of the authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. This report is part of a larger multi-investigator project supported by the U. S. Department of Energy entitled the Cementitious Barriers Partnership. The opinions, findings, conclusions, or recommendations expressed herein are those of the authors and do not necessarily represent the views of the U.S. Department of Energy. This work was also partially supported by the National Institute of Standards and Technology Sustainable Concrete Materials program.

and

This report is based on work supported by the U. S. Department of Energy, under Cooperative Agreement Number DE-FC01-06EW07053 entitled ‘The Consortium for Risk Evaluation with Stakeholder Participation III’ awarded to Vanderbilt University. The opinions, findings, conclusions, or recommendations expressed herein are those of the author(s) and do not necessarily represent the views of the Department of Energy or Vanderbilt University.

REFERENCES

1. F.G. SMITH III, G. FLACH, AND K.G. BROWN, "CBP Code Integration GoldSim DLL Interface”, CBP-TR-2010-009-2, Rev. 0. Savannah River National Laboratory and Vanderbilt University/CRESP; Cementitious Barriers Partnership: Aiken, SC and Nashville, TN (2010).
2. GTG, “GoldSim User's Guide: Probabilistic Simulation Environment (Volume 1 of 2; Version 10.0)”, GoldSim Technology Group (2009).
3. J. MARCHAND, E. SAMSON, Y. MALTAIS, R. LEE, AND S. SAHU, “Predicting the performance of concrete structures exposed to chemically aggressive environment—Field validation”, Materials and Structures, **35**(10): pp. 623-631 (2002).

4. SIMCO, “Software for Transport And Degradation in Unsaturated Materials (STADIUM) Version 2.8 User Guide”, SIMCO Technologies, Inc. (2008).
5. CBP, “Description of the Software and Integrating Platform (Contains 4 Chapters),” CBP-TR-2009-003, Rev. 0, Cementitious Barriers Partnership, Available from: http://cementbarriers.org/reports.html (2009).
6. E. SAMSON AND J. MARCHAND, “Modeling the transport of ions in unsaturated cement-based materials”, Computers & Structures, **85**(23-24): pp. 1740-1756 (2007).