CBP CODE INTEGRATION GOLDIM INTEGRATION INTERFACE

Cementitious Barriers Partnership

June 2010

CBP-TR-2010-009-2

CBP CODE INTEGRATION GOLDSIM DLL INTERFACE

Frank Smith
Savannah River National Laboratory
Aiken, SC 29808

Greg Flach
Savannah River National Laboratory
Aiken, SC 29808

Kevin G. Brown
Vanderbilt University, School of Engineering
Consortium for Risk Evaluation with Stakeholder Participation III
Nashville, TN 37235

June 2010

CBP-TR-2010-009-2, Rev. 0

ACKNOWLEDGEMENTS

This report was prepared for the United States Department of Energy in part under Contract No. DE-AC09-08SR22470 and is an account of work performed in part 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.

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 U.S. Department of Energy or Vanderbilt University.

DISCLAIMER

Printed in the United States of America

United States Department of Energy
Office of Environmental Management
Washington, DC

FOREWORD

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.

A multi-disciplinary partnership of federal, academic, private sector, and international expertise has been formed to accomplish the project objective. In addition to the US DOE, the CBP partners are the Savannah River National Laboratory (SRNL), Vanderbilt University (VU) / Consortium for Risk Evaluation with Stakeholder Participation (CRESP), Energy Research Center of the Netherlands (ECN), and SIMCO Technologies, Inc. The Nuclear Regulatory Commission (NRC) is providing support under a Memorandum of Understanding. The National Institute of Standards and Technology (NIST) is providing research under an Interagency Agreement. Neither the NRC nor NIST are signatories to the Cooperative Research and Development Agreement (CRADA).

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 set of simulation tools and data developed under this project will be 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 (spent-fuel pools, dry spent-fuel storage units, and recycling facilities such as fuel fabrication, separations processes). Simulation parameters will be obtained from prior literature and will be experimentally measured under this project, as necessary, to demonstrate application of the simulation tools for three prototype applications (waste form in concrete vault, high-level waste tank grouting, and spent-fuel pool). Test methods and data needs to support use of the simulation tools for future applications will be defined.

The CBP project is a five-year effort focused on reducing the uncertainties of current methodologies for assessing cementitious barrier performance and increasing the consistency and transparency of the assessment process. The results of this project will enable improved risk-informed, performance-based decision-making and support several of the strategic initiatives in the DOE Office of Environmental Management Engineering & Technology Roadmap. Those strategic initiatives include 1) enhanced tank closure processes; 2) enhanced stabilization technologies; 3) advanced predictive capabilities; 4) enhanced remediation methods; 5) adapted technologies for site-specific and complex-wide D&D applications; 6) improved SNF storage, stabilization and disposal preparation; 7) enhanced storage, monitoring and stabilization systems; and 8) enhanced long-term performance evaluation and monitoring.

Christine A. Langton, PhD
Savannah River National Laboratory

David S. Kosson, PhD
Vanderbilt University / CRESP

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

CBP	Cementitious Barriers Partnership
CRESP	Consortium for Risk Evaluation with Stakeholder Participation
DLL	Dynamic-link library
DOE	Department of Energy
ECN	Energy Research Centre of the Netherlands
GTG	GoldSim Technology Group
NIST	National Institute of Standards and Technology
NRC	Nuclear Regulatory Commission
SRNL	Savannah River National Laboratory
STADIUM	Software for Transport and Degradation in Unsaturated Materials
XML	eXtensible Markup Language

CBP Code Integration GoldSim DLL Interface

Frank Smith
Savannah River National Laboratory
Aiken, SC 29808

Greg Flach
Savannah River National Laboratory
Aiken, SC 29808

Kevin G. Brown
Vanderbilt University, School of Engineering
Consortium for Risk Evaluation with Stakeholder Participation III
Nashville, TN 37235

1 INTRODUCTION

A general dynamic-link library (DLL) interface has been developed to link GoldSim with external codes[1]. The overall concept behind this development is to use GoldSim as top level modeling software with interfaces to external codes for specific calculations. 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. Instructions for creating the input file, running the external code, and reading the output are contained in an instructions file (`DLL.dat`) that is read and interpreted by the DLL. As an example, a prototype model linking GoldSim with the STADIUM® code used to predict concrete service life has been developed and successfully run. While the example is for an interface between GoldSim and STADIUM (Brown & Flach 2009), the DLL is designed to be general and should be readily adaptable to interfacing other codes to GoldSim.

2 DLL DESIGN

Design of the DLL assumes that the external application reads an input file that controls the calculations performed and writes an output file of results. The exact formatting of the input and output does not matter as long as the location of information in each file can be uniquely identified. The input can then be modified as needed to make different calculations and the results of these calculations can be processed for further use. The DLL also assumes that the external code can be automatically run as either an executable or through a

[1] According to Microsoft (http://msdn.microsoft.com/en-us/library/ms682589(VS.85).aspx), a DLL is a module that contain functions and data that can be used by another module (application or DLL).

batch file (calling the executable) without user intervention. Minimal user intervention could be tolerated but defeats the purpose of code automation especially in the context of probabilistic analysis.

To some extent, the DLL design was based on information provided in Appendix C of the GoldSim User's Guide (Volume 2 GoldSim Technology Group, Version 10.0 February 2009 (GTG 2009)). The User's Guide specifies, in general, how the external interface must be constructed to be compatible with GoldSim's calling conventions and gives a list of parameters that must be used. Within these constraints, the user can add additional functionality as desired. It is this additional functionality included in the DLL interface that is primarily the subject of this document.

The DLL code was written in Fortran 90 and is modularized into five files and 20 subroutines. A brief explanation of the function of each subroutine is provided in Appendix A. Operation of the DLL interface is controlled by instructions provided by the user in a `DLL.dat` file. An example instruction file is shown in Figure 1. This 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. The following key words initiate specific actions in the DLL interface:

1. `PUT` – Put data specified within the block into the named file.
2. `GET` – Get data specified within the block from the named file.
3. `EXE` – Perform the system calls specified within the block.
4. `RPL` – Replace complete lines in the named file.
5. `SUP` – Create a "super" file containing the commands or file names listed within the block.
6. `LOG` – Write a log file in XML format containing the input and output data.
7. `END` – Termination of a command block.

The actions are processed in the order that they appear in the `DLL.dat` file and each of the actions can be used more than once. Typically, the user would want to enter data into an input file using the `PUT` command, run an external application that reads from the input file using the `EXE` command, and retrieve output from the external application using the `GET` command. The DLL performs some error checking as it processes the instructions it receives. For example, if a `GET` or `PUT` instruction points to a file location that does not exist, the code will trap this error, send an error message back to GoldSim, and cleanly exit from the DLL interface. However, this error checking is not as extensive as it could be and will be expanded in future releases.

```
!
!      #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.xml
END
!-----------------------------------------------------------------------------------------------------
```

Figure 1. Example DLL.dat file

3 USER GUIDE

This section provides a guide to using the DLL by describing the commands that can be entered into the `DLL.dat` file.

3.1 PUT and GET

The `PUT` and `GET` commands are functionally similar and will be described together. The name of the file to be processed by the `PUT` or `GET` command is listed on the same line as the command in the column (next tab position) immediately following the key word. Following the file name, the delimiter used to separate data fields in the file is given in the next column. Recognized delimiter names are: `colon`, `comma`, `semicolon`, `space`, `tab`, and `white`. The delimiter "`white`" indicates any combination of tabs and/or spaces which will appear as "`white`" space in the file. When identifying delimiters in the files, it is assumed that the delimiter appears only one time following each field with the exception of spaces which can be inserted multiple times. For "`white`" space, tabs are replaced with spaces during file processing. If the delimiter name is followed by the key word "`ignore`," any delimiters appearing before the first data entry are stripped out and ignored.

Between the `PUT` or `GET` command and the `END` statement, that indicates the termination of this command block, the instructions on each of the lines that direct the command operations are entered in Fields 2 – 12. These fields contain the information listed below.

Field 2: Input arguments are passed from GoldSim to the external interface in the array `inargs()`, and output arguments are passed from the external interface back to GoldSim in the array `outargs()`. Following Fortran 90 convention, the starting index of each array is 1 (rather than 0 in the C language). The number in Field 2 indicates the position in the one-dimensional array that begins the command. `PUT` commands use the `inargs()` array while `GET` commands use the `outargs()` array.

`PUT` commands must start with `inargs(3)`. The first two entries in the `inargs()` array have been reserved to pass two special parameters from GoldSim to the DLL interface. The first parameter, `isave` passed in `inargs(1)`, is set by the user within GoldSim to indicate whether results from individual GoldSim Monte Carlo realizations are to be saved (`isave>0`) or not (`isave=0`). If `isave=0`, a subdirectory named `realization_0` is setup to hold results from the GoldSim run. If GoldSim is then run in Monte Carlo mode with multiple realizations, results saved in the subdirectory would be overwritten with each realization and only results from the final realization will be saved. If `isave>0` and GoldSim is run with multiple realizations 1 … n, individual subdirectories `realization_1` through `realization_n` are setup to save results from each realization. The second parameter passed from GoldSim in `inargs(2)` is the realization number.

Field 3 – Field 6: These entries specify how the row in either the input or output files where the data values are located is identified. The key words that can appear in Field 3 are listed in Table 1. The information that must appear in Fields 4 – 6 depends on the key word in Field 3 as explained in the Table 1.

Table 1. Row Specification in Field 3

Key Word	Function
row	The row (record) number where the data is located is entered in Field 4.
record	The record (row) number where the data is located is entered in Field 4.
label	An alphanumeric label identifying the row where the data is located is entered in Field 4 and the column where the label is to be read is entered in Field 5.
value	A numerical value that identifies the row where the data is located is entered in Field 4, the column where the label is to be read is entered in Field 5, and a tolerance on the value is entered in Field 6. A positive tolerance in Field 6 specifies that an absolute difference is used to test if the numerical value in Field 5 has been found while a negative tolerance specifies a relative difference. The DLL reads numerical values in the specified column and in all rows starting with the first row of data until the specified value is found. The data entry at this location is then used in the `PUT` or `GET` command.
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. The DLL reads record entries as text strings in the file starting with the first row until the specified label is found. The data entry at this location is then used in the `PUT` or `GET` command.

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 2.

Table 2. Column Specification in Field 7

Key Word	Function
col	The column (field) number where the data is located is entered in Field 8.
field	The field (column) number where the data is located is entered in Field 8.
heading	An alphanumeric label identifying the column where the data is located is entered in Field 8 and the row where the label is located is entered in Field 9. 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.
value	A numerical value that identifies the column where the data is 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. A positive tolerance in Field 10 specifies that an absolute difference is used to test if the numerical value in Field 8 has been found while a negative tolerance specifies a relative difference. The DLL reads numerical values in the specified row and in all columns starting with the first column of data until the specified value is found. The data entry at this location is then used in the `PUT` or `GET` command.

Field 11: This field contains the number of rows to be used for data processing; this allows entering or reading a row vector with a single command. For example, if a `PUT` command has the number `n` entered in Field 2 and the number `m` in Field 11, `inargs(n)` will be written into the data file row and column identified by the information in Fields 3 – 10. The values `inargs(n+1)` through `inargs(n+m-1)` will be written into the next (`m-1`) rows in the same column.

Field 12: This field contains the number of columns to be used for data processing; this allows entering a column vector with a single command or, in conjunction with Field 11, entering a matrix of values with a single command. 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, `outargs(n)` will be read from the data file row and column identified by the information in Fields 3 – 10. 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 next row down. This process will continue over `m` rows of data until a total of `m×p` values have been read.

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

3.2 RPL

The `RPL` command can be used to replace entire lines in an existing input file. GoldSim passes double precision numerical values in the `inargs()` and `outargs()` arrays. The only exception to this is that the external function can return an error message to GoldSim in the `outargs()` array using a special function supplied by GoldSim Technology Group (GTG). Passing only numerical values can be restrictive because input files may contain text strings with, for example, file names, and it may be desirable to change these file names for different simulations. As a simple work around to this limitation, 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 of input that will be replaced. The text starting in Field 3 will be used to replace the current entry in the line.

3.3 EXE

The `EXE` command block specifies DOS commands to be executed by the Windows operating system. The feature is primarily used to give the name and relative location of the file (or files) that must be run to execute the external code. This command could be used to run a batch file or an executable. Any arguments that must be passed to the executable are included in the command. Additional commands can be specified to perform other operations, such as copying or renaming output files.

3.4 SUP

The `SUP` command writes the specified text to the specified file. The file name must be provided in the second column on the same line as the `SUP` command. The file is created if not found, or overwritten if existing. The keyword can be used to create a "super" file of file names and instructions that can then be accessed by the external application.

3.5 LOG

The `LOG` command is used to give the name of a file where arrays of the GoldSim input and output data used in the simulation will be written in XML format. The log file name must be provided in the second column on the same line as the `LOG` command. The realization number is written to the log file followed by an array of the input data and an array of the output data.

4 EXAMPLES

4.1 PUT/GET Example

To test whether the various options available to `PUT` and `GET` values from the input and output arrays function correctly, a simple test run was made using the `DLL.dat` file shown in Figure 2 with the starting `test.inp` file shown in Figure 3. The `PUT` instructions should replace the values in `test.inp` along a diagonal from value (2,2) to value (6,6) with zeros. The test was made by running the GoldSim model intended for use with the STADIUM code with this `DLL.dat` instruction file. As the modified `test.inp` file in Figure 4 shows the PUT operations worked correctly. The GET instructions were designed to read these zero values. Examination of the test.xml file (not provided) showed that the first six output arguments returned from the DLL to GoldSim were zero as intended.

```
!
!      #2     #3     #4     #5     #6     #7     #8     #9     #10    #11    #12    #13 (comment)
!---------------------------------------------------------------------------------------------------------------
PUT    test.inp      white  ignore
       2      row    2                    field  2                    1      1      "row" + "field" option
       2      record 3                    col    3                    1      1      "record" + "col" option
       2      label  line3  1             headingfield4 1             1      1      "label" + "heading" option
       2      value  123    2      +0.01  value  -444   2      +0.01  1      1      "value" option
       2      value  456    2      -0.01  value  -555   2      -0.01  1      1      "value" option
       2      string myStr                col    7                    1      1      "string" option
END
!---------------------------------------------------------------------------------------------------------------
GET    test.inp      white  ignore
       1      row    2                    field  2                    1      1      "row" + "field" option
       2      record 3                    col    3                    1      1      "record" + "col" option
       3      label  line3  1             headingfield4 1             1      1      "label" + "heading" option
       4      value  123    2      +0.01  value  -444   2      +0.01  1      1      "value" option
       5      value  456    2      -0.01  value  -555   2      -0.01  1      1      "value" option
       6      string myStr                col    7                    1      1      "string" option
END
!---------------------------------------------------------------------------------------------------------------
LOG    test.xml
END
!---------------------------------------------------------------------------------------------------------------
```

Figure 2. DLL.dat file used to test DLL PUT and GET options

```
!field1 field2  field3  field4  field5  field6  field7
line1   -111    -222    -333    -444    -555    -666
line2   -11     -22     -33     -44     -55     -66
line3   -999    -999    -999    -999    -999    -999
line4   123     -999    -999    -999    -999    -999
line5   456     -999    -999    -999    -999    -999
line6   -999    -999    -999    myStr   -999    -999
line7   -999    -999    -999    -999    -999    -999
line8   -999    -999    -999    -999    -999    -999
```

Figure 3. Input file used to test DLL PUT and GET options

```
!field1 field2  field3  field4  field5  field6  field7
line1   0       -222    -333    -444    -555    -666
line2   -11     0       -33     -44     -55     -66
line3   -999    -999    0       -999    -999    -999
line4   123     -999    -999    0       -999    -999
line5   456     -999    -999    -999    0       -999
line6   -999    -999    -999    myStr   -999    0
line7   -999    -999    -999    -999    -999    -999
line8   -999    -999    -999    -999    -999    -999
```

Figure 4. Modified input file after running GoldSim

4.2 STADIUM Example

This example uses the `DLL.dat` file shown in Figure 1 to make a one year run of the STADIUM code with three realizations. Because the DLL replaces values in the input file, the user must supply a template file for the DLL to manipulate. As shown in Section 3.1, the DLL provides some flexibility for locating data within the template file. However, the user must either know the exact location of input parameters within the input file or know the basic structure of the input so that the methods described in Section 3.1 can be applied. The example `DLL.dat` file in Figure 1 locates data by giving specific row and column positions. The template file used for the STADIUM code is shown in Figure 5. After the DLL executes the `RPL` and `PUT` commands shown in Figure 1, the modified template file that was used as the actual input file for STADIUM is shown in Figure 6. The current version of the model did not try to replace the entire input file; the values actually replaced by the DLL are highlighted in Figure 6.

The `GET` command shown in Figure 1 was used to retrieve concentrations of the 11 chemicals and nine minerals used by the STADIUM code at each of the 101 node locations at time one year. The `GET` command can be interpreted as follows:

> Look for a value of 1.0 in the first column of output data with a relative tolerance of 0.1 year. When this value is found, starting in column 4, retrieve 101 rows of data over 11 columns, then, starting in column 18, retrieve 101 rows of data over 9 columns. The first block of data retrieved is placed in the `outargs()` array starting with element one. The second block of data is placed in the `outargs()` array starting with element 3312.

The `outargs()` array that returns output to GoldSim has been dimensioned large enough to return concentrations at one time step for a mesh as large as 301 nodes. The gap between the two blocks of output data retrieved in this example allows blank spaces for the data that would have appeared in nodes 102 through 301.

```
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

CHIM
    NUMBER_CHEM_PARAM.     3
    m_max               5
    print_level         1
    iter_max        1000


    Nions     11
    Nsolides  9
```

Figure 5. Template input file for STADIUM

```
    Database_file   CHM-DB-STADIUM.txt

    OH
    Na
    K
    SO4
    Ca
    Al(OH)4
    Cl
    H2SiO4
    CO3
    NO3
    NO2

    Portlandite
    CaH2SiO4
    Ettringite
    Monosulfate
    C4AH13
    Thaumasite
    Calcite
    Monocarboaluminate
    Gypsum

COND
    Nb_cycles           1
    Sequences_days      0  365.0

    OH                  1    0.0       1         0.0        0.0      0.0
    Na                  1    0.0       1         0.0        0.0      0.0
    K                   1    0.0       1         0.0        0.0      0.0
    SO4                 1    0.0       1         0.0        0.0      0.0
    Ca                  1    0.0       1         0.0        0.0      0.0
    Al(OH)4             1    0.0       1         0.0        0.0      0.0
    Cl                  1    0.0       1         0.0        0.0      0.0
    H2SiO4              1    0.0       1         0.0        0.0      0.0
    CO3                 1    0.0       1         0.0        0.0      0.0
    NO3                 1    0.0       1         0.0        0.0      0.0
    NO2                 1    0.0       1         0.0        0.0      0.0
    Humidity            2    0.0       1         0.0        0.0      1.0
                                 101    0.0       0.0      1.0
    Potential           1    0.0       1         0.0        0.0      0.0
    Temperature         0


CONV
    Nb_cycles           1
    Sequences_days      0    365

    OH                  0
    Na                  0
    K                   0
    SO4                 0
    Ca                  0
    Al(OH)4             0
```

Figure 5. Template input file for STADIUM (Cont'd)

```
    Cl                  0
    H2SiO4          0
    CO3                 0
    NO3                 0
    NO2                 0
    Humidity            0
    Potential           0
    Temperature         2    0.0        1         365.0     0.0       5.0  15.0
                                        101       365.0     0.0       5.0  15.0
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
IMPR
    number_print_times     31
    print_times
       1.0
       10.0
       20.0
       50.0
       75.0
       100.0
       200.0
       300.0
       400.0
       500.0
       600.0
       700.0
       800.0
       900.0
       1000.0
       1500.0
```

Figure 5.   Template input file for STADIUM (Cont'd)

```
      2000.0
      2500.0
      3000.0
      4500.0
      5000.0
      5500.0
      6000.0
      6500.0
      7000.0
      7500.0
      8000.0
      8500.0
      9000.0
      9500.0
      10000.0
    print_before_chm       0
    level_1_2       1
    imp_flux_0_1_2  0
STOP
```

Figure 5. Template input file for STADIUM (Cont'd)

```
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

CHIM
    NUMBER_CHEM_PARAM.     3
    m_max               5
    print_level         1
    iter_max        1000


    Nions     11
    Nsolides  9
```

Figure 6. STADIUM input file after execution of PUT and RPL commands

```
    Database_file   CHM-DB-STADIUM.txt

    OH
    Na
    K
    SO4
    Ca
    Al(OH)4
    Cl
    H2SiO4
    CO3
    NO3
    NO2

    Portlandite
    CaH2SiO4
    Ettringite
    Monosulfate
    C4AH13
    Thaumasite
    Calcite
    Monocarboaluminate
    Gypsum

COND
    Nb_cycles           1
    Sequences_days      0  365.0

    OH                  1    0.0       1         0.0        0.0      0.0
    Na                  1    0.0       1         0.0        0.0      0.0
    K                   1    0.0       1         0.0        0.0      0.0
    SO4                 1    0.0       1         0.0        0.0      0.0
    Ca                  1    0.0       1         0.0        0.0      0.0
    Al(OH)4             1    0.0       1         0.0        0.0      0.0
    Cl                  1    0.0       1         0.0        0.0      0.0
    H2SiO4              1    0.0       1         0.0        0.0      0.0
    CO3                 1    0.0       1         0.0        0.0      0.0
    NO3                 1    0.0       1         0.0        0.0      0.0
    NO2                 1    0.0       1         0.0        0.0      0.0
    Humidity            2    0.0       1         0.0        0.0      1.0
        101  0.0     0.0     1.0
    Potential           1    0.0       1         0.0        0.0      0.0
    Temperature         0


CONV
    Nb_cycles           1
    Sequences_days      0    365

    OH                  0
    Na                  0
    K                   0
    SO4                 0
    Ca                  0
    Al(OH)4             0
```

Figure 6. STADIUM input file after execution of PUT and RPL commands (Cont'd)

```
    Cl                    0
    H2SiO4            0
    CO3                      0
    NO3                   0
    NO2                   0
    Humidity              0
    Potential             0
    Temperature           2     0.0         1         365.0      0.0        5.0   15.0
        101        365.0      0.0        5.0        15.0
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
IMPR
    number_print_times     31
    print_times
       1.0
       10.0
       20.0
       50.0
       75.0
       100.0
       200.0
       300.0
       400.0
       500.0
       600.0
       700.0
       800.0
       900.0
       1000.0
       1500.0
```

Figure 6. STADIUM input file after execution of PUT and RPL commands (Cont'd)

```
      2000.0
      2500.0
      3000.0
      4500.0
      5000.0
      5500.0
      6000.0
      6500.0
      7000.0
      7500.0
      8000.0
      8500.0
      9000.0
      9500.0
      10000.0
    print_before_chm      0
    level_1_2       1
    imp_flux_0_1_2  0
STOP
```

Figure 6. STADIUM input file after execution of PUT and RPL commands (Cont'd)

5 REFERENCES

Brown, KG & Flach, GP 2009, *CBP Software Summaries for LeachXS™/ORCHESTRA, STADIUM®, THAMES, and GoldSim*, CBP-TR-2009-003, Rev. 0, in *Description of the Software and Integrating Platform*, Vanderbilt University/CRESP and Savannah River National Laboratory; Cementitious Barriers Partnership, Nashville, TN and Aiken, SC. Available from: http://cementbarriers.org/reports.html.

GTG 2009, *GoldSim User's Guide: Probabilistic Simulation Environment (Volume 2 of 2)*, Version 10.0 (February 2009) edn, 2 vols, GoldSim Technology Group, Issaquah, WA. Available from: www.goldsim.com (license required) [September 1, 2009].

GTG 2010, *GoldSim Dashboard Authoring Module User's Guide*, Version 10.1 (January 2010) edn, GoldSim Technology Group, Issaquah, WA. Available from: www.goldsim.com (license required).

APPENDIX A

LISTING AND EXPLANATION OF SUBROUTINES IN GOLDSIM DLL INTERFACE

The DLL interface code was written in Fortran 90 and is modularized into the five files and 19 subroutines listed below.

`DllExternalCode_g95.f90`

```
DllExternalCode
```

`Filework.f90`

```
ReadSup
ReadRpl
RunExe
WriteLog
GetOrPut
ReadRow
ReadCol
ReadRecords
setDelim
checkField
Continuation
```

`GSUtilities.f90`

```
gs_parameters
copy_msg_to_outputs
```

`Params.f90`

`putget.f90`

```
putField
getField
FindDelim
IostatCheck
logstring
```

Module `Params` contains a list of parameters common to all of the other modules. `Params` also contains the two parameters `NINPUTS` and `NOUTPUTS` that give the number of inputs passed from GoldSim to the DLL and the number of outputs passed back from GoldSim to the DLL. These values must agree with the number of inputs and outputs defined in the GoldSim interface. The current version of the DLL has `NINPUTS = 119` and `NOUTPUTS = 6020`. The number of inputs is exactly the number of values that GoldSim uses to set up a STADIUM input file. The number of outputs is set large enough to hold data for 20 chemical and mineral concentrations at up to 301 nodes. If fewer nodes are present, the output array can still be organized correctly so that individual chemical and mineral species can be separated. However, it may be necessary to manually change these parameter values and recompile the DLL for different simulations. `Params` also contains names for the file containing the DLL instructions (generically referred to as `DLL.dat` in this document) and a log file where information primarily of interest for debugging purposes is written. The user may wish to change the default names of these files and recompile the DLL for specific simulations.

DLL SUBROUTINES

`gs_parameters`

This module specifies parameters used by GoldSim as a part of its DLL external interface. These parameters indicate the phase of the simulation currently in progress and provide return codes to GoldSim indicating the completion status of the external calculation. This subroutine was provided in Appendix C of the GoldSim User's Guide (Volume 2 GoldSim Technology Group, Version 10.0 February 2009 (GTG 2009)) and was used without change.

Parameters used to indicate the phase of the simulation are:

`INITIALIZE` – Called after DLL is loaded and before each realization.

`REPORT_VERSION` – Called after DLL load to report the external function version number.

`REPORT_ARGUMENTS` – Called after DLL load to report the number of input and output arguments.

`CALCULATE` – Called during the simulation, each time the inputs change.

Parameters providing return codes to GoldSim are:

`CLEANUP` – Called before the DLL is unloaded.

`SUCCESS` – Call was completed successfully.

`CLEANUP_NOW` – Call was successful, but GoldSim should clean up and unload the DLL immediately.

`FAILURE` – Failure (no error information returned).

`FAILURE_WITH_MSG` – Failure, with DLL supplied error message available. Address of error message is returned in the first element of the output arguments array.

`INCREASE_MEMORY` – Failed because the memory allocated for output arguments is too small. GoldSim will increase the size of the output argument array and try again.

Subroutine to pass error message to GoldSim:

`copy_msg_to_outputs (smsg, outargs)`

`smsg`..................String variable containing output message
`outargs`...........Array of output arguments returned to GoldSim

To help GoldSim users debug problems with DLL external functions, GoldSim allows users to send an error message from the DLL back to GoldSim through the external element interface when the call to an external function fails. The error message is displayed to the user in a pop-up dialog. The subroutine that performs this task was provided by GoldSim in Appendix C of the User's Guide (GTG 2009) and was used without change. An example of the use of this subroutine is shown in Figure A-1.

```
subroutine DllExternalCode (method_id, status, inargs, outargs) BIND (C)

!Variable typing statements

select case (method_id)
  case (INITIALIZE)
    outargs(1) = NINPUTS
    outargs(2) = NOUTPUTS
    status = SUCCESS

  case (REPORT_VERSION)
    outargs(1) = VERSION
    status = SUCCESS

  case (REPORT_ARGUMENTS)
    outargs(1) = NINPUTS
    outargs(2) = NOUTPUTS
    status = SUCCESS

  case (CALCULATE)
    !Set working directory
    !Open file for logging processing of DLL instructions
    !Make subdirectory for run(s)
    !Make subdirectory for realization

    !Read in instructions
    call ReadRecords (datfile, instructions, nInstructions, msg)

    !Put inargs(*) into input file(s)
    call GetOrPut (streamlined, nStreamlined, subdir, "PUT", inargs, msg)

    !Create superfile per instructions
    call ReadSup (streamlined, nStreamlined)

    !Replace lines in input file per instructions
    call ReadRpl (streamlined, nStreamlined)

    !Execute command per instructions
    call RunExe (streamlined, nStreamlined)

    !Get outargs(*) from output file(s)
    call GetOrPut (streamlined, nStreamlined, subdir, "GET", outargs, msg)

    !Create logfile
    call WriteLog (streamlined, nStreamlined, inargs, outargs, irealization)

    !Report success
    status = SUCCESS

  case (CLEANUP)
    status = SUCCESS

  case default
    msg = "Unknown method ID requested"
    call copy_msg_to_outputs(msg, outargs)
    status = FAILURE_WITH_MSG

end select

end subroutine DllExternalCode
```

Figure A-1. Outline of Subroutine DllExternalCode

DllExternalCode (method_id, status, inargs, outargs)

`method_id`......Parameter indicating phase of simulation
`status`............Return code to GoldSim indicating status of external calculation
`inargs` ....Array of input arguments received from GoldSim
`outargs`...........Array of output arguments returned to GoldSim

This is the main subroutine called by the external link and, as such, controls the flow of the file processing. The basic structure that must be followed for this subroutine was given in Appendix C of the GoldSim User's Guide (GTG 2009). An outline version of this subroutine with many of the detailed statements removed is shown in Figure A-1. This outline illustrates the use of the parameters listed above and calls to some of the primary subroutines in module `Filework`.

ReadSup (records, nRecords)

`records`...........Array of records in `SUP` block
`nRecords` ....Number of records in `SUP` block

Subroutine `ReadSup` reads the instructions in the `SUP` block and writes them into a "super" file that can then be used by external applications. This feature would typically be used to provide a list of files and their relative locations required by the external application.

ReadRpl (records, nRecords)

`records`...........Array of records in `RPL` block
`nRecords` ....Number of records in `RPL` block

Subroutine `ReadRpl` reads the instructions in the `RPL` block and replaces the identified lines in the input file with the text supplied in the records. The first entry in each record is the number of the line in the file that is to be replaced and the second entry in each record is the text string that will be placed in the file.

RunExe (records, nRecords)

`records`...........Array of records in `EXE` block
`nRecords` ....Number of records in `EXE` block

Subroutine `ReadExe` reads the instructions in the `EXE` block and makes system calls to execute each of the commands listed within the block. More than one external application can be executed. Any command-line arguments required by the executables must be included in the commands.

`WriteLog (records, nRecords, inargs, outargs, irealization)`

`records`..................Array of records in `LOG` block
`nRecords` ....Number of records in `LOG` block
`inargs` ....Array of input arguments received from GoldSim
`outargs`..................Array of output arguments returned to GoldSim
`irealization` ......Number of the realization being run

Subroutine `WriteLog` writes an XML formatted log file containing the realization number and the input and output data used for this particular calculation.

`GetOrPut (DLLdat, nDLLdat, subdir, action, args, msg)`

`DLLdat`.............Array of records in `GET` or `PUT` block
`nDLLdat`...........Number of records in `GET` or `PUT` block
`subdir` ....For a `PUT` command, the name of the subdirectory where the input file is to be written
`action`.............Key word `GET` or `PUT`
`args`..................For `PUT` command `inargs()` array, for `GET` command `outargs()` array
`msg`....................Text containing error message

Subroutine `GetOrPut` implements the `GET` or `PUT` commands entered into the `DLL.dat` file. These commands are described in Section 3.1.

`ReadRow (vctr, iloc, array, nRecords, delim, msg)`

`vctr`..................Vector containing row specifications from Fields 3 – 6 in the `PUT` or `GET` command
`iloc`..................Row number where `PUT` or `GET` command should start operating
`array` ....Array of records in file where `PUT` or `GET` command is operating
`nRecords` ....Number of records in file where `PUT` or `GET` command is operating
`delim` ....Delimiter used in file where `PUT` or `GET` command is operating
`msg`....................Text string containing error message

Subroutine `ReadRow` reads the row specifications in a `PUT` or `GET` command (see Section 3.1, Field 3 – Field 6) and uses the specifications to find the row within the data file where the command should start operating.

`ReadCol (vctr, iloc, array, nRecords, delim, msg)`

`vctr`..................Vector containing column specifications from Fields 7 – 10 in the `PUT` or `GET` command
`iloc`..................Column number where `PUT` or `GET` command should start operating
`array` ....Array of records in file where `PUT` or `GET` command is operating
`nRecords` ....Number of records in file where `PUT` or `GET` command is operating
`delim` ....Delimiter used in file where `PUT` or `GET` command is operating
`msg`....................Text string containing error message

Subroutine `ReadCol` reads the column specifications in a `PUT` or `GET` command (see Section 3.1, Field 7 – Field 10) and uses the specifications to find the column within the data file where the command should start operating.

ReadRecords (file, array, nRecords, msg)

`file`.................Text string with name of file to read
`array`...............Array of records in file
`nRecords`........Number of records in file
`msg`...................Text string containing error message

Subroutine `ReadRecords` reads the data records in the specified file and returns an array containing the records and the number of records found. If a read error is encountered, an error message is returned in text string `msg`.

setDelim (field, delim, msg)

`field`...............Text string with name of file delimiter
`delim`...............Text character representing delimiter used in file
`msg`...................Text string containing error message

Subroutine `setDelim` reads the name of the delimiter specified in the `DLL.dat` file and returns the delimiter character. If an incorrect delimiter name is encountered, an error message is returned in text string `msg`. Valid delimiters are: colon, comma, semicolon, space, tab, and white (a combination of tabs and spaces).

Strip (records, nRecords, delim)

`records`..........Array of records
`nRecords`........Number of records
`delim`...............Text character representing delimiter used in file

Subroutine `Strip` removes leading blank spaces from the set of records passed to the subroutine.

checkField (field, check, toler, found)

`field`...............Text string holding data that is to be checked
`check`...............Value that is being searched for
`toler`...............Tolerance allowed in identifying value
`found`...............Logical variable indicating whether the numerical value of the data in the field is within the tolerance of the value of check.

Subroutine `checkField` implements the option to search for a numerical value to identify either the row or column in a data file that a `PUT` or `GET` command is manipulating.

`Continuation (nInput, input, continuationCharacter, nOutput, output)`

`nInput` .... Number of records in original `DLL.dat` file
`input` .... Array of records in original `DLL.dat` file
`continuationCharacter` ...... Text character used to identify a continuation line
`nOutput` .... Number of records in processed `DLL.dat` file
`output` .... Array of records in processed `DLL.dat` file

Subroutine `Continuation` collapses continued lines in the `DLL.dat` file into a single record, removes comment lines, and returns an array of processed `DLL.dat` command lines.

`putField (line, lineLength, nField, field, fieldLength, delim, new_line, lerr)`

`line`..................... A text string containing the record to be modified
`linelength`....... Maximum length of the record
`nField`............... The field in the record to be replaced
`field` .... A text string containing the value to be inserted into the record
`fieldlength`..... Maximum length of the field
`delim` .... The delimiter used in the record
`new_line` .... Logical variable indicating if the record is a new line of the data file
`lerr`..................... Logical variable indicating an error in the PUT operation

Subroutine `putField` is called to insert a data value into a data record replacing the old value at the same position. Only new lines need to have delimiter positions located. That is, when the subroutine is inserting another value into the same line used previously it already knows the delimiter positions.

`getField (line, lineLength, nField, field, fieldLength, delim, new_line)`

`line`..................... A text string containing the record to be used
`linelength`....... Maximum length of the record
`nField`............... The field in the record to be extracted
`field` .... A text string containing the value extracted from the record
`fieldlength`..... Maximum length of the field
`delim` .... The delimiter used in the record
`new_line` .... Logical variable indicating if the record is a new line of the data file

Subroutine `getField` is called to extract a data value from a data record. Only new lines need to have delimiter positions located. That is, when the subroutine is reading another value from the same line used previously it already knows the delimiter positions.

`FindDelim (line, linelength, delim, position)`

`line`..................... A text string containing the record to be searched
`linelength`....... Maximum length of the record

`delim` .... The delimiter to search for
`position` .... An array of delimiter positions

Subroutine `FindDelim` is called to locate delimiter positions in a data record and return a list of the positions to the calling routine.

`IostatCheck (iostat_flag, exit_flag, msg)`

`iostat_flag`..... Integer variable indicating status of read
`exit_flag` .... Logical variable signaling exit from read
`msg`....................... Text string containing error message

Subroutine `IostatCheck` is called to check the status of an attempted read statement. If an error or end of file is detected, `exit_flag` is set to false. If an error is detected, the error message "`***READ ERROR ***`" is returned to the calling program.

`logstring (iunt, label, value)`

`iunit` ....Output unit where label and value will be written
`label` ....Text string containing label to be output
`value` ....Text string containing value to be written

Subroutine `logstring` is called to write a labeled value to an output file. This subroutine is used to write messages to file `DLL.log` indicating the status of operations the DLL is performing to assist the user in debugging applications.

U.S. DEPARTMENT OF ENERGY

SRNL
SAVANNAH RIVER NATIONAL LABORATORY

CRESP

VANDERBILT UNIVERSITY

UNITED STATES NUCLEAR REGULATORY COMMISSION

NIST
National Institute of Standards and Technology

ECN

SIMCO Technologies inc.