

---

# **The AREST Code: User's Guide for the Analytical Repository Source-Term Model**

**D. W. Engel  
A. M. Liebetau  
G. C. Nakamura  
B. M. Thornton  
M. J. Apted**

---

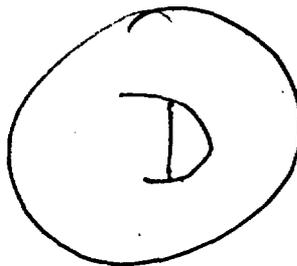
**May 1989**

**Prepared for the U.S. Department of Energy  
under Contract DE-AC06-76RLO 1830**

**Pacific Northwest Laboratory  
Operated for the U.S. Department of Energy  
by Battelle Memorial Institute**

**PNL-6645**

8907170358 890710  
PDR WASTE  
WM-11 PDC



102.2

## DISCLAIMER

These programs were prepared as an account of work sponsored by an agency of the United States Government. Neither the contractor, nor the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

## NOTICE

This document was produced with Government support under Contract DE-AC06-76RLO 1830 awarded by the United States Department of Energy. The Government has certain rights in the information and data contained herein.

PACIFIC NORTHWEST LABORATORY  
*operated by*  
BATTELLE MEMORIAL INSTITUTE  
*for the*  
UNITED STATES DEPARTMENT OF ENERGY  
*under Contract DE-AC06-76RLO 1830*

Copyright © 1989 Battelle Memorial Institute  
All Rights Reserved

**THE AREST CODE: USER'S GUIDE  
FOR THE ANALYTICAL REPOSITORY  
SOURCE-TERM MODEL**

**D. W. Engel  
A. M. Liebetrau  
G. C. Nakamura  
B. M. Thornton  
M. J. Apted**

**May 1989**

**Prepared for  
the U.S. Department of Energy  
under Contract DE-AC06-76RLO 1830**

**Pacific Northwest Laboratory  
Richland, Washington 99352**

**Copyright © 1989: Battelle Memorial Institute**

## EXECUTIVE SUMMARY

This document is a user's guide for the Analytical REpository Source-Term (AREST) code that has been developed by the Performance Assessment Scientific Support (PASS) program at the Pacific Northwest Laboratory(a). This development was sponsored by the Office of Civilian Radioactive Waste Management (OCRWM) of the U.S. Department of Energy. The purpose of the code is to provide OCRWM with a tool for making preliminary, quantitative performance assessments of the engineered barrier and near-field systems of a geologic repository for high-level radioactive waste. These analyses will provide program guidance and help to identify technical design issues that require resolution. This effort is complementary to other U.S. (O'Connell and Drach 1986; Pigford and Chambre' 1987) and foreign (KBS 1983; LeNeveu 1986; Sharland, Tasker, and Tweed 1987) efforts for developing models and codes for engineered-barrier and near-field performance assessment.

The AREST code is divided into three parts: the Waste Package Containment (WPC), the Waste Package Release (WPR), and the Engineered System Release (ESR) components. This user's guide describes the code's organization, as well as the execution procedure. Configuration of the present version of the AREST code is based on the PASS computing network at Pacific Northwest Laboratory (PNL); implementation on other computer systems may necessitate additional modifications.

To assess waste package performance, the AREST code relies on input arrays of environmental parameters. These are typically supplied by chemical, hydrological, thermal, mechanical, and radiological support codes that are separate from the formal AREST code. There are several available, well-documented codes in each of these areas; the flexibility to choose among these codes is left to the user. An example set of default values for these parameters, taken for a postulated configuration of a waste package in an unsaturated tuff geologic setting, are tabulated in this guide and within this version of the AREST code.

---

(a) Pacific Northwest Laboratory is operated for the U.S. Department of Energy by Battelle Memorial Institute under Contract DE-AC06-76RLO 1830.

A modular structure was designed into the AREST code in order to permit updating and revision to the code as new data or performance models become available. Specific instructions for modifying the code are provided in this guide. Expectation of and accommodation for future revisions is vital, especially for a code such as AREST, which will evolve from its preliminary stage of development. As new barrier materials or repository environments are proposed and studied, code modification must follow. In addition, continuing peer review may reveal the need to abandon or modify existing performance models. It is hoped, therefore, that this initial, documented version of the AREST code will serve as a focus for expanded participation by groups with constructive comments regarding realistic and defensible models that can be incorporated for waste package performance analysis.

## ACKNOWLEDGMENTS

The authors wish to acknowledge N. C. Garisto and D. M. LeNeveu, of Atomic Energy of Canada Limited, for their assistance on the AREST code and comparison of results from the SYVAC Vault Model (LeNeveu 1986) with the AREST code, which gave us a better understanding of the use of the decay-chain model, precipitation model, and  $UO_2$  solubility function. We would also like to thank T. H. Pigford, P. L. Chambre' and their students at the University of California at Berkeley for the mass transfer equations used in the AREST code and for their assistance on the AREST code. Finally, we would like to thank P. W. Reimus for reviewing a draft of this report, T. L. Gilbride for editorial assistance, and S. L. Cammann for typing.

## CONTENTS

EXECUTIVE SUMMARY . . . . .	iii
ACKNOWLEDGMENTS . . . . .	v
1.0 INTRODUCTION . . . . .	1.1
2.0 OVERVIEW OF THE AREST CODE . . . . .	2.1
2.1 CODE INPUT . . . . .	2.1
2.2 WASTE PACKAGE CONTAINMENT . . . . .	2.1
2.3 WASTE PACKAGE RELEASE . . . . .	2.1
2.4 ENGINEERED SYSTEM RELEASE . . . . .	2.2
3.0 LOGICAL STRUCTURE OF THE AREST CODE . . . . .	3.1
3.1 GENERAL STRUCTURE . . . . .	3.1
3.2 WASTE PACKAGE CONTAINMENT . . . . .	3.1
3.3 WASTE PACKAGE RELEASE . . . . .	3.2
3.4 ENGINEERED SYSTEM RELEASE . . . . .	3.3
4.0 INPUT . . . . .	4.1
4.1 DATA FILE FORMAT . . . . .	4.1
4.2 THE DATA EDITOR . . . . .	4.3
5.0 OUTPUT . . . . .	5.1
6.0 FUTURE DEVELOPMENTS . . . . .	6.1
7.0 REFERENCES . . . . .	7.1
APPENDIX A - DEFAULT INPUT VALUES (TUFF) . . . . .	A.1
APPENDIX B - SAMPLE RUNS (TUFF) . . . . .	B.1
APPENDIX C - ARESTIN MODULE . . . . .	C.1
APPENDIX D - ENVIR MODULE . . . . .	D.1
APPENDIX E - GRDH20 MODULE . . . . .	E.1
APPENDIX F - CORRODE MODULE . . . . .	F.1

CONTENTS (Continued)

APPENDIX G - PREFAIL MODULE . . . . .	G.1
APPENDIX H - DISTRIB MODULE . . . . .	H.1
APPENDIX I - MODEL MODULE . . . . .	I.1
APPENDIX J - PITDPH MODULE . . . . .	J.1
APPENDIX K - CTINV MODULE . . . . .	K.1
APPENDIX L - RPARAM MODULE . . . . .	L.1
APPENDIX M - SOLUB MODULE . . . . .	M.1
APPENDIX N - RMATRIX MODULE . . . . .	N.1
APPENDIX O - RGAP MODULE . . . . .	O.1
APPENDIX P - RCRUD MODULE . . . . .	P.1
APPENDIX Q - RCLAD MODULE . . . . .	Q.1
APPENDIX R - RSUM MODULE . . . . .	R.1
APPENDIX S - CUMREL MODULE . . . . .	S.1
APPENDIX T - USER-SUPPLIED ROUTINES . . . . .	T.1

TABLES

3.1	Logical Flow of the AREST Code . . . . .	3.1
3.2	Logical Flow of the WPC Component . . . . .	3.2
3.3	Logical Flow of the WPR Component . . . . .	3.2
3.4	Logical Flow of the ESR Component . . . . .	3.3
4.1	Summary of the Input Commands for the Data Editor of AREST . . . .	4.5
A.1	Sample Primary Input Data File . . . . .	A.2
A.2	Input Time-Temperature Data File . . . . .	A.7
A.3	Input Heat Loading Probability Function . . . . .	A.7
B.1	Waste Package Containment . . . . .	B.2
B.2	Individual Waste Package Release . . . . .	B.3
B.3	Average Release Rate from the Repository . . . . .	B.4
B.4	Total (Cumulative) Release from the Repository . . . . .	B.5
E.1	Sample Input Data File (ENVSOL.DAT) . . . . .	E.2
R.1	Example of Different Failure Times . . . . .	R.2

## 1.0 INTRODUCTION

The AREST code, which implements the Analytical REpository Source-Term (AREST) model for the evaluation of radionuclide release from an engineered barrier system containing spent nuclear fuel, has been developed for the Office of Civilian Radioactive Waste Management (OCRWM) by the Performance Assessment Scientific Support (PASS) program. The performance models and assumptions for the AREST code have already been described in detail (Liebetrau et al. 1987a; Liebetrau et al. 1987b). This document will deal only with the implementation, user modifications, testing, and running of the AREST code.

The purpose of this user's guide is to show the reader how to use the AREST code. Specific instructions for modifying and running the AREST code on the PASS program MicroVAX system at the Pacific Northwest Laboratory (PNL) are given. The AREST code is currently running on a MicroVAX II, from Digital Equipment Corporation, and is written in Fortran 77. The AREST code was written using most of the Fortran 77 standards, with non-standard features described in Appendix T. Sample runs, argument lists and test plans for major modules of the code are also provided. The AREST code has been designed in a modular fashion in order to make changes and updates to the code relatively easy.

This report consists of six sections which describe the AREST model and its implementation and several appendices which describe major modules of the code and sample runs. An overview of the AREST code is contained in Section 2.0. The logic flow of the AREST code and its three major components are described in Section 3.0. Section 4.0 discusses the input process of the AREST code. Output from the AREST code is described in Section 5.0. Future developments to the AREST code are described in Section 6.0.

Appendix A contains default data files and a description of how to run the AREST code. Appendix B shows the results of a run using the default data files from Appendix A. The default data have been described in other reports (Apted, Liebetrau, and Engel 1987; Apted and Engel 1988). The remaining appendices describe the major modules that make up the AREST code. Each appendix contains a description of the module, the arguments that are passed to and from it, the routines it calls, the logic flow, and a test plan. The

appendices provide an in-depth description of the AREST code and can be used, along with the actual code, to make modifications to the AREST code. The final appendix (Appendix T) describes the user-supplied routines that are set up in the AREST code. They are designed to be easily implemented and used for modeling corrosion and release rates.

## 2.0 OVERVIEW OF THE AREST CODE

The AREST code is broken into three major components plus an input module. The three components consist of the Waste Package Containment (WPC), the Waste Package Release (WPR), and the Engineered System Release (ESR).

### 2.1 CODE INPUT

The input to the AREST code includes the values assigned to spent fuel waste package, and repository design variables, and to variables that describe the physical and chemical environments of the repository near-field and waste package. The actual input occurs in the module ARESTIN (Appendix C).

Input values can be supplied to the AREST code by means of input data files or interactively by the user. The input process is described in more detail in Section 4.0; sample input data files are supplied in Appendix A.

### 2.2 WASTE PACKAGE CONTAINMENT

The WPC component describes the containment performance of an individual waste package from the time of repository closure to the time of containment failure. Containment performance is simulated by using mechanistic models of corrosion and degradation processes. Alternatively, users can supply a hypothetical containment failure distribution of their choice.

The mechanistic models of corrosion and degradation processes may be supplied by the user or they may be chosen from a list of preprogrammed models that are supplied with the AREST code. The WPC component contains preliminary models for uniform corrosion, stress corrosion, pitting corrosion, fractures, and uniform corrosion of Zircaloy cladding of spent nuclear fuel.

### 2.3 WASTE PACKAGE RELEASE

Upon containment failure, the present WPR component describes the release of radionuclides from spent fuel and their migration outward through the waste package. The WPR component consists of separate modules for each of four radionuclide sources from spent fuel the  $UO_2$  matrix, the gap, the surface layer/crud, and cladding (Liebetrau et al. 1987a, Appendix D). Each of these

modules (RMATRIX, RGAP, RCRUD, and RCLAD) are described in one of the appendices. Modification of the WPR module for analysis of release from borosilicate glass waste forms is in progress.

#### 2.4 ENGINEERED SYSTEM RELEASE

The ESR component integrates (sums) the releases calculated from successive waste package simulations. Both the average release from the repository for each radionuclide and the total radionuclide release are calculated by the ESR component.

### 3.0 LOGICAL STRUCTURE OF THE AREST CODE

#### 3.1 GENERAL STRUCTURE

The AREST code has been configured in a modular fashion so that it is easy to modify. The logical flow of the AREST code is provided, Table 3.1. In Table 3.1 the name in capital letters in the column to the right is the name of the module that performs the task described. Major variables (e.g., iwp and TFA) and components (e.g., WPC, WPR, and ESR) occur in parentheses in the description.

TABLE 3.1. Overall Logic of the AREST Code

<u>Description</u>	<u>Module</u>
1. Input repository, waste package, environmental, and design parameters.	(ARESTIN)
2. Calculate performance for each waste package (iwp).	
a. Set up waste package parameters.	
1. Simulate Time - Temperature distribution.	(SENVIR)
2. Calculate time of resaturation.	(TRESAT)
b. Simulate containment failure (WPC).	(CORRODE)
c. Set up parameters for post-failure.	
1. Calculate temperatures for failure times.	(ENVIR)
2. Output containment failure information.	(OUTCOR)
3. Set times of failure.	
a. Containment failure after resaturation (TFA).	
b. Container failure for any environment steam or water (TFAC)	
c. Container failure after resaturation (TFACR).	
4. Calculate time-dependent inventories.	(CTINV)
5. Obtain groundwater composition.	(GRDH2O)
d. Calculate waste package release (WPR).	
3. Store and output repository release information (ESR).	

#### 3.2 WASTE PACKAGE CONTAINMENT

The logical flow of the WPC component of the AREST code is shown in Table 3.2. The WPC module is called from the AREST code to return the time of container (TFAC) and containment (TFA) failures. The term container failure refers to the failure, or breaching of the outside container wall. Where the term containment failure refers to the total loss of containment capability.

This refers to failure of the container wall plus the failure of the zircaloy cladding.

**TABLE 3.2. Logical Flow of the WPC Component**

Description	Module
1. Simulate prefailures.	(PREFAIL)
2. If container did not prefail, then simulate containment performances.	
a. Simulate via user-supplied distribution or	(DISTRIB)
b. Simulate via corrosion models.	(MODEL)
1. Determine initial environmental values and corrosion rates.	(PHASE, ENVIR, INITCUM)
2. For each time step,	
a. Update environmental values (e.g., temperature, groundwater, composition).	(PHASE, ENVIR, GRDH20)
b. If container failure has not occurred, then simulate modes of corrosion other than uniform corrosion.	(NONUNIF)
c. If container failure has occurred, then simulate cladding corrosion.	(CLADDNG)
d. Simulate uniform corrosion.	(UNIFORM)
e. If containment failed, failure is indicated, proceed to WPR component.	

### 3.3 WASTE PACKAGE RELEASE

Following containment failure, radionuclide release rates are calculated in the WPR component of the AREST code. The logical flow of the WPR component is shown in Table 3.3.

**TABLE 3.3. Logical Flow of the WPR Component**

Description	Module
1. Calculate release for each nuclide.	
a. Matrix Model	
1. Set parameters involved in release calculation.	(RPARAM)
2. Calculate UO <sub>2</sub> solubility.	(SOLUB)
3. Calculate precipitation of uranium.	(SXPPT, SCPPT, STPPT)
4. Calculate release.	(RMATRIX)
b. Gap/Grain Boundary Model	
1. Set release parameters for individual nuclide.	(RPARAM)
2. Calculate release.	(RGAP)
c. Surface Layer/Crud Model	
1. Set release parameters for individual nuclide.	(RPARAM)
2. Calculate release.	(RCRUD)
d. Cladding Model	
1. Set release parameters for individual nuclide	(RPARAM)
2. Calculate release.	(RCLAD)

### 3.4 ENGINEERED SYSTEM RELEASE

The logical structure of the ESR component is shown in Table 3.4. The ESR component is not a separate module like the other major components (WPC and WPR) but is contained in the main part of the AREST code. It takes the results from the WPC and WPR components to calculate the average fractional release rate (parts/year) for the repository and also the total release (grams/year) from the repository.

TABLE 3.4. Logical Flow of the ESR Component

<u>Description</u>	<u>Module</u>
1. For each waste package sum releases and store particular rates for distribution of releases.	(RSUM)
2. Calculate average repository release for each nuclide.	
3. Calculate total repository release amounts for each nuclide.	(CUMARR)
4. Output release (rates and amounts).	(OUTREL)

## 4.0 AREST CODE INPUT

The bulk of the AREST code input data is read from files. However, the AREST code allows for editing of data prior to use. Changes to the data may be saved for later use. This section describes the general format of the data files and the use of the AREST code's data editor.

### 4.1 DATA FILE FORMAT

A primary goal in designing the input file was to make it easy for the user, as well as computers, to read. For the user, the file should contain comments and the data should be in a relatively flexible format. For computers, however, the format should be fairly consistent so that data is easy to find and read.

The AREST code satisfies both requirements by using a line-oriented data format. By imposing only a few format restrictions, AREST can differentiate between data and comments, assuming that the calling routine provides information about the number and type of data items expected. Perhaps the clearest way to explain these restrictions is to describe what the AREST code looks for in an input file.

The AREST code distinguishes between two types of data lines, numeric and character, on the basis of the first data item of the line. When the AREST code searches for a numeric data line, it looks for a line which has a numeric character (digit, sign, or decimal point) in the first column. Any lines found before such a line are assumed to be comment lines.

Example:

```
_____
| This doesn't start with numeric data; it's treated as a comment.
| 3.1416 This is the data line.
```

When searching for a character data line, the program first looks for a line ending in a colon. The next line found with a non-whitespace (i.e., not a space or tab) character in the first column is assumed to be the data line.

Example:

When the program is looking for character data, it expects to first find one or more comment lines (four in this example). These comment lines can even go into the first column like this. The only requirement is that they end with a colon:

These comment lines and the blank lines around the comment are assumed not to be the desired data line, since the first character in each line is whitespace.

This is the desired character data line!

The AREST code can be used to read multiple lines (a block) of similar data. The first data line, either numeric or character, is found as described above. Reading continues until a line starting with whitespace is found (a blank line will do). All intervening lines are assumed to be data.

Example:

This is a comment line:

<u>Nuclide</u>	<u>Half Life</u>	(These lines are comments, but the ones that follow are data.)
C-14	5.73e+03	
Tc-99	2.15e+05	
Np-237	2.14e+06	

This line servers as a delimiter.

To summarize, the following rules apply:

- 1) All data begins in column 1.
- 2) Numeric data is assumed to start with a digit, sign, or decimal point.
- 3) Character data must be flagged by a line ending with a colon.

- 4) Multiple lines of similar data should be delimited by a line starting with whitespace.

The current version of the AREST code uses both primary and secondary data files; the above format rules apply to both types. The existing data files are in the correct format and may be edited using the AREST code's data editor or a text editor.

#### 4.2 THE DATA EDITOR

Although it is possible to use a text editor to edit data files, the AREST code provides a data editor which can be used for most changes to the input data set.

When running the AREST code, the user will be prompted for the name of the primary input file. Primary input is always assumed to be from a file. If the file is not found, the prompt will be repeated. Upon finding the specified file, the AREST code will ask if the data needs to be edited. If the answer is yes ('Y' or 'y'), the data editor is invoked. Otherwise, the AREST code simply reads in the data in preparation for corrosion and release simulations.

The data editor is menu-based, each menu holding one or more data lines, each line holding one or more data items (values). To edit items, in the AREST code, selections are made by indicating the row letter and the item number. The row letter is simply the letter at the start of the row. Rows are lettered consecutively, beginning with 'A'. The easiest way to define "item number" is by example.

Example:

```
C) U-238 4.51e+09 3.10e+06
```

In the above line

'C' is the row letter

'U-238' is the first item (item number = 1)

'4.51e+09' is the second item (item number = 2)

'3.10e+06' is the third item (item number = 3)

Thus, an item is any contiguous group of non-whitespace characters, and the item number is the relative position of that item within the line. To

select an item, specify the row letter followed immediately by the item number. The item number can be omitted for the first item in the line (i.e., the default item number is 1).

Example:

For the same data line as the last example  
to select 'U-238', enter 'C1' or 'C'  
to select '4.51e+09', enter 'C2'  
to select '3.10e+06', enter 'C3'

Once a selection is made, the AREST code will display the current value of the item, then prompt for the new value. If it is decided not to replace the item, press RETURN; otherwise, enter one or more items. Items in the selected line will be replaced by typed entries in a one-to-one correspondence. AREST will attempt to preserve the left-justification of the original items, but if necessary will move an item to the right to prevent it from running into the previous item. \*

In addition to item replacement, there are several commands which can be invoked. The most often used commands will probably be the MENU and NEXT MENU commands:

Enter '?' to have the current menu redisplayed.  
Press RETURN to go to the next menu.

**WARNING!** The AREST code processes the data file sequentially, and cannot backtrack to a previous menu. Be sure to finish with the current menu before going to the next one.

The other general purpose commands are GOTO, HELP, QUIT, and UNDO. In all cases, these commands can be shortened to their first two letters. (In fact, the AREST code does not check beyond the second letter.) The commands can be shortened to one letter provided there is no conflict with a menu row letter. A summary of these commands appear in Table 4.1.

Entering 'GOTO' in response to the menu prompt brings up the GOTO menu. This menu allows the user to jump forward in the data file to the next section

**TABLE 4.1.** Summary of the Input Commands for the Data Editor of AREST

<u>Command</u>	<u>Description</u>
?	Reprint Menu
Dn	Delete Menu Line n
In	Insert Before Line n
Goto	Go Forward to Another Menu
Quit	End Editing, Read the Rest of the Input from the File
Undo	Restore to Previous File
<Return>	Next Menu

which may require editing. There are a limited number of preset points (called "stops") to which the user may jump. If it is decided not to skip to another section, press RETURN. To select a stop, enter a menu row letter. The program still has to read the skipped data, so there may be a delay before the next menu appears.

Entering 'HELP' in response to the menu prompt causes the help file to be displayed. The help file contains a summary of item replacement and available editor commands.

Entering 'QUIT' in response to the menu prompt indicates that no further data will be edited. Again, there may be a delay while the program reads in the remaining values.

Entering 'UNDO' in response to the menu prompt causes AREST to restore the previous contents of the last edited line (of the current menu).

Two other commands, INSERT and DELETE, can be used when editing a block of data, i.e., multiple lines of similar data.

DELETE is used to remove a data line. The syntax for this command is 'D' followed immediately by the menu row letter. The items in the selected line will be replaced by minus signs. Note that these minus signs are treated as items and can be edited in the same manner as other items.

INSERT is used to add data lines. The syntax is 'I' followed by a row letter. Insertion will occur before that row. For inserting after the last row, use the next letter of the alphabet for the row letter. After indicating the insertion location, the user will be prompted for the number of lines to insert. (Enter 0 if it is decided not to insert any lines.) The AREST code places no limit on this number. However, only one set of insertions can be made before a given row, so care must be taken so that all necessary insertions are made at the same time.

After entering the number of insertions desired, the insertion menu will appear with dummy values in the data lines. The new values are entered by editing these dummy values. Editing is done in the same way as for the regular menus, and applicable commands can still be used.

When the data editor is exited (either by a RETURN on the last menu or by a QUIT), the AREST code will prompt the user for file names for storing the revised input data. Pressing RETURN in response to a prompt indicates that particular revision will not be saved.

## 5.0 AREST OUTPUT

The output from AREST is in the form of data files, see Appendix B for an example. The output is accomplished by writing to specific unit numbers (15, 16, 40, 41, 42, and 43). On the system that the AREST code is implemented (MicroVAX II), writing to unit 15 creates a data file "FOR015.DAT." The first data file, FOR015.DAT (Unit 15), contains the input information for the WPC component. The next data file, FOR016.DAT, contains the information for the WPR component. This includes the container dimensions, parameters that affect release, and information for each nuclide being used.

The third data file, FOR040.DAT, contains the information about individual waste package containment. It consists of the time, temperature, and mechanism by which the container and containment failed for each waste package simulated.

The last three files contain the release for each nuclide at each time step. The first file, FOR041.DAT, contains the fractional release rates (parts/year) for the first failed waste package. The time in this file is the time since containment failure. The next file, FOR042.DAT, contains the average repository fractional release rates from each release model (matrix, gap/grain boundary, surface layer/crud, and cladding). The last file, FOR043.DAT, contains the total release from the repository (in grams/year). The time in the last two files, is time after repository closure.

## 6.0 FUTURE DEVELOPMENTS

The version of the AREST code that is documented here corresponds to the AREST model that has been previously reported (Liebetrau et al. 1987a). Since the release of that document, some new features have been incorporated into the AREST code that were not previously described. These new features include uranium precipitation and a new release model for a partially saturated media. The effects of precipitation using the AREST code have been reported previously (Apted and Engel 1987). The new release model calculates steady-state release based on rates of matrix dissolution, diffusion, and convection (Reimus et al. 1988).

The modeling of the effects of decay chains and the graphical display of the results are performed external to the AREST code. These two features are currently being added to the AREST code.

Future developments for the AREST code consist of a spatial capability of the repository and additional models for corrosion and release.

## 7.0 REFERENCES

Apted, M. J., A. M. Liebetrau, and D. W. Engel. 1987. "Spent Fuel as a Waste Form: Analysis with AREST Performance Assessment Code." Waste Management '87, 2:545-554, ed. P. G. Post. University of Arizona, Tucson.

Apted, M. J., D. W. Engel. 1988. "Analysis of Congruent Matrix Release, Precipitation, and Time-Dependent Containment Failure on Spent Fuel Performance." Sci. Basis. Nucl. Waste Management VI, eds. M. J. Apted and R. F. Westerman, pp. 303-312. Materials Research Society, Pittsburgh.

KBS. 1983. Final Storage of Spent Nuclear Fuel - KBS-3. Karnbranslesakerhet Report SKBF/KBS, Stockholm, Sweden.

LeNeveu, D. M. 1986. Vault Submodel for the Second Interim Assessment of the Canadian Concept for Nuclear Fuel Waste Disposal: Post-Closure Phase. AECL-8383, Atomic Energy of Canada Limited, Whiteshell Nuclear Research Establishment, Pinawa, Manitoba, Canada.

Liebetrau, A. M., M. J. Apted, D. W. Engel, M. K. Altenhofen, D. M. Strachan, C. R. Reid, C. F. Windisch, R. L. Erikson, and K. I. Johnson. 1987a. The Analytical Repository Source-Term (AREST) Model: Description and Document. PNL-6346, Pacific Northwest Laboratory, Richland, Washington.

Liebetrau, A. M., M. J. Apted, D. W. Engel, M. K. Altenhofen, C. R. Reid, D. M. Strachan, R. L. Erikson, and D. H. Alexander. 1987b. "AREST: A Probabilistic Source-Term Code for Waste Package Performance Analysis." Waste Management '87, 2:535-544, ed. P. G. Post. University of Arizona, Tucson.

O'Connell, W. J. and R. S. Drach. 1986. Waste Package Performance Assessment: Deterministic System Model Program Scope and Specification. UCRL-53761, Lawrence Livermore National Laboratory, Livermore, California.

Pigford, T. H. and P. L. Chambre'. 1987. Verification, Validation, and Reliability of Predictions. LBL-23215, Lawrence Berkeley Laboratory, Berkeley, California.

Reimus, P. W., A. M. Liebetrau, M. J. Apted, and D. W. Engel. 1988. Performance Assessment for Spent Fuel Waste Packages in Hydrologically Unsaturated Geologic Media. Spectrum '88, American Nuclear Society, Chicago, Illinois.

Sharland, S. M., P. W. Tasker, and C. J. Tweed. 1987. The Coupling of Chemical and Transport Processes in Near-Field Modelling. AERE-R-12450, Harwell Laboratory, Oxfordshire, United Kingdom.

APPENDIX A

DEFAULT INPUT VALUES

## APPENDIX A

### DEFAULT INPUT VALUES

Input to the AREST code is primarily done through the use of data files. The standard practice is to edit a data file to reflect a specific run, or the data editor can be used to modify input, as described in Section 4.0. A sample of a primary data file is listed in Table A.1. Note that all of the data lines start in column one, and that the order in which the data is input must remain fixed, unless the ARESTIN module is modified, see Appendix C.

All of the variables in Table A.1 are described briefly either by the comment line before the variable or on the same line after the value. Many of the parameters do not vary from run to run or are simply not used for each run, so modification of the file is generally not necessary. For instance, the corrosion parameters, for modeling corrosion, are specific to the corrosion mechanism and models being used. An example is the molecular weight of the metal (the first parameter in the corrosion section) which is used only by one of the existing corrosion models in AREST, the PNL pitting corrosion model. If this model is not used, there is no need to worry about modifying this variable.

The release section of Table A.1 contains three tables of parameters describing the individual nuclides. A nuclide is used in the analysis if a (Y or y) is in the first column of the nuclide in all three tables. Otherwise the nuclide is not used.

The data file containing the time, reference temperature, and the minimum repository temperature is one of the input parameters in Table A.1 (Time-Temperature Data, TIMETEMP.DAT). The default time-temperature file is shown in Table A.2, where the data again must occupy the first column. Also, to act as a delimiter, a blank line must be present at the end of this data file. The heat loading values and their cumulative probabilities are shown in Table A.3. A blank line must also be the last entry in the heat loading data file.

The use of the files contained in Tables A.2 and A.3 are discussed briefly in Appendix D (ENVIR MODULE). The files are read into AREST using the INENVIR

routine. This routine is discussed in Appendixes C (ARESTIN MODULE) and T (USER-SUPPLIED ROUTINES). There is an additional file read into the AREST code, which is not listed in this Appendix. The file contains the time/temperature dependent ground water composition and is discussed in Appendix E (GRDH20 MODULE).

**TABLE A.1. Sample Primary Input Data File**

**Run Identification:**

TUFF

**Time-Temperature Data:**

timetemp          File name

**Heat Load and Cumulative Probability Data:**

heatl            File name

**Miscellaneous Time-Temperature and Heat Load Data:**

100              Temperature of resaturation (integer)  
 29               Ambient temperature (integer)  
 187.67          Heat load conversion factor (real)

**Corrosion Control Data:**

1 0.0 0          1. Type of simulation (0=model, 1=distribution)  
                  2. Prob. of prefailure  
                  3. Number of prefailures

**Corrosion Parameters (for modeling):**

55.8            Molecular weight of metal (g/mol) (real)  
 7.86            Density of metal (g/cm3) (real)  
 5E-6            Ionic diffusivity of metal cation (cm2/s) (real)  
 2                Proton charge on metal cation (equiv/mole) (integer)  
 -1               Proton charge on aggressive anion (equiv/mol) (integer)  
 3.3E-3          Empirical parameter A for PNL Pitting (real)  
 .5               Empirical parameter N for PNL Pitting (real)  
 19.1000        Chloride concentration (mg/l) (real)  
 -.02            Potential drop along pit length (V) (real)  
 6.8             pH (real)  
 1.0             Oxygen concentration (mg/l) (real)  
 12000           Average number of pits per container (integer)  
 0.0044          Surface area of each pit (cm2)  
 60000.0        Waste container surface area (cm2)  
 5.0             Waste container thickness (cm)  
 0.00016133    % total surface area that constitutes failure  
 4.0             Allowable corrosion depth (UNIFORM cm)  
 5.0             Allowable corrosion depth (STRESS cm)  
 2.362           Allowable corrosion depth (PITTING P[Z > KF])  
 5.0             Allowable corrosion depth (FRACTURE cm)  
 5.0             Allowable corrosion depth (CLADDING cm)  
 .0 4.0          a and b from oxide formula MaOb  
 2                Valency of metal cation  
 0.4             EH  
 -276.0E3       Free energy for formation of corrosion film (delta Gc) (J/mol)  
 -6.78E4        Free energy for dissolution of metal (delta Gd) (J/mol)  
 0.24            Empirical parameter alpha for PNL UNIFORM  
 0.005           Empirical parameter beta for PNL UNIFORM

0.156	Empirical parameter B1	for PNL UNIFORM	(cm-K/yr)
38000.0	Empirical parameter B2	for PNL UNIFORM	(joules/mole)
1.43E7	Empirical parameter C1	for PNL UNIFORM	(cm-deg K)/yr
-3.629E5	Empirical parameter C2	for PNL UNIFORM	(joules/mole)
0.1	Empirical parameter PH20	for PNL STEAM	(MPa)
1.7	Empirical parameter ALPA	for PNL STEAM	

Corrosion models (from list):

For each model:

0 = User supplied

1 = Not used

2+= Predefined

Steam:

2 = PNL

3 = BWIP

1

Uniform:

2 = BWIP

3 = DTP

4 = Westerman

5 = Brookhaven

6 = PNL

1

Stress:

1

Pitting:

2 = PNL

3 = DTP

1

Fracture:

1

Cladding (steam):

2 = PNL

1

Cladding (aqueous):

2 = PNL

1

Distribution and Parameters for Corrosion Data

1, Time, 0 --> Point

2, Min, Max --> Uniform

3, Mean, Standard Deviation --> Normal

4, Min, Lambda --> Exponential

Data:

3 1000 200

**Release Control:**

0 Simulating only corrosion? (0=no, 1=yes)

**Release Models:**

**Steam Release Model for Cladding from List:**

- 0 --> User supplied subroutine
- 1 --> Not using Steam Release
- 2 --> Instantaneous Release to outer barrier

1

**Matrix Release Model from List:**

- 0 --> User supplied subroutine
- 1 --> Not using matrix release
- 2 --> Pigford & Chambre'
- 3 --> PNL TUFF model

2

**Gap Release Module from List:**

- 0 --> User supplied subroutine
- 1 --> Not using gap release
- 2 --> Pigford & Chambre'
- 3 --> PNL TUFF model

2

**Grain Release Module from List:**

- 0 --> User supplied subroutine
- 1 --> Not using grain release

1

**Cladding Release from List:**

- 0 --> User supplied subroutine
- 1 --> Not using cladding release
- 2 --> Pigford & Chambre'
- 3 --> PNL TUFF model

2

**UO2 Solubility Function from List:**

- 0 --> Not being used
- 1 --> Use reference temperature
- 2 --> Use temperature at failure

1

**Assuming congruent release of UO2?**

- 0 --> No
- 1 --> Yes

1

Release Input Data:

5E-1 Diffusion coefficient in steam, (cm<sup>2</sup>/sec)  
 1E-5 Diffusion coefficient in water, (cm<sup>2</sup>/sec)  
 34.0 Cylindrical waste form radius, (cm)  
 400.0 Cylindrical waste form length, (cm)  
 3.0 Packing thickness, (cm)  
 1 0 --> into packing 1 --> into host rock  
 0.5 Porosity of packing  
 0.1 Porosity of host rock  
 10.3 Gap width, (cm)  
 92715.0 Gap surface area, (cm<sup>2</sup>)  
 957014.0 Gap volume, (cm<sup>3</sup>)  
 7 Number of PWR assemblies per waste package  
 1.29 Bulk density of packing (g/cm<sup>3</sup>)  
 2.33 Bulk density of host rock (g/cm<sup>3</sup>)  
 60.0 Percent saturated

Distribution Type and Parameters for Simulating Release:

0, 0, 0 --> No simulation  
 1, Min, Max --> Simulate Uniform  
 2, Mean, Standard Deviation --> Simulate Normal

0 0.2 0.5 Porosity of packing  
 0 0.002 0.02 Porosity of host rock  
 0 0 0 Retardation coefficient in packing  
 0 0 0 Retardation coefficient in host rock

0 Number of percentiles on output  
 1 Using vector approach (0 --> No 1 --> Yes)

Percentages of Radionuclides in Matrix, Gap, Grain, Cladding:

Use	<u>ELEM</u>	<u>MATRIX</u>	<u>GAP</u>	<u>GRAIN</u>	<u>CRUD</u>	<u>CLAD</u>
y	U-238	100.	0.	0.	0.	0.
n	U-236	100.	0.	0.	0.	0.
n	U-234	100.	0.	0.	0.	0.
n	Am-241	100.	0.	0.	0.	0.
n	Cs-135	98.	2.	0.	0.	0.
n	I-129	98.	2.	0.	0.	0.
n	Pu-240	100.	0.	0.	0.	0.
y	C-14	35.	1.	0.	1.	63.
y	Zr	100.	0.	0.	0.	0.

Matrix Release Data:

<u>Use</u>	<u>Elem</u>	<u>T-1/2</u> (yrs)	<u>INVEN</u> (Ci/1000MTU)	<u>SOLUB</u> (g/m**3)	<u>KD1</u> (ml/gm)	<u>KD2</u> (ml/gm)	<u>SP.ACT.</u> (Ci/g)
y	U-238	4.51E+09	3.20E+02	5.00E+01	1.80E+00	1.80E+00	3.33E-07
n	U-236	2.39E+07	2.30E+02	5.00E+01	1.80E+00	1.80E+00	6.34E-05
n	U-234	2.47E+05	7.80E+02	5.00E+01	1.80E+00	1.80E+00	6.18E-03
n	Am-241	4.58E+02	3.50E+05	1.45E-03	1.20E+03	1.20E+03	3.24E+00
n	Cs-135	3.00E+06	2.70E+02	8.10E+05	2.90E+02	2.90E+02	8.82E-04
n	I-129	1.59E+07	3.30E+01	7.74E+05	0.00E+00	0.00E+00	1.74E-04
n	Pu-240	6.58E+03	4.10E+05	1.20E-01	6.40E+01	6.40E+01	2.26E-01
y	C-14	5.73E+03	6.90E+02	1.40E+00	0.00E+00	0.00E+00	4.45E+00
y	Zr	0.00E+03	5.11E+09	4.60E-05	0.00E+00	0.00E+00	4.45E+00

Matrix Release Data (cont):

<u>Use</u>	<u>Elem</u>	<u>Diffusion</u> (cm**2/s)	<u>Shared</u> <u>Csat flag</u>
y	U-238	1.00E-05	0.0
n	U-236	1.00E-05	0.0
n	U-234	1.00E-05	0.0
n	Am-241	1.00E-05	0.0
n	Cs-135	1.00E-05	0.0
n	I-129	1.00E-05	0.0
n	Pu-240	1.00E-05	0.0
y	C-14	1.00E-05	0.0
y	Zr	1.00E-05	0.0

1000      Number of waste packages to simulate  
0          Trace?                    (0 --> No    1 --> Yes)  
1          Output release? (0 --> No    1 --> Yes)  
0          Output UO2 solubility? (Csat(UO2) (0-->No, 1-->Yes)

**TABLE A.2. Input Time-Temperature Data File**

<u>Time (years)</u>	<u>Ref Temp (°C)</u>	<u>Min Temp (°C)</u>
0.005	179.99	29.00
0.27	179.99	39.79
1	216.67	50.14
5	248.01	73.27
10	249.04	86.74
20	242.74	101.32
30	233.21	109.13
40	224.02	113.44
50	213.50	115.02
70	197.39	115.25
100	183.03	114.40
200	157.00	111.19
300	143.00	107.96
400	132.00	106.19
500	126.00	104.70
1000	110.00	98.09
2000	90.00	83.52
6000	58.00	54.23
10000	47.00	45.90
100000	47.00	45.90

**TABLE A.3. Input Heat Loading Probability Function**

<u>Heat Load (kw/MTU)</u>	<u>Cum Prob</u>
0.2	0.0
0.5	0.11
1.0	0.57
1.2	0.82
1.5	0.96
2.0	0.99
2.5	1.0

**APPENDIX B**

**SAMPLE RUN**

## APPENDIX B

### SAMPLE RUN

The results of running the AREST code using the data described in Appendix A are shown in Tables B.1, B.2, B.3, and B.4. These tables list a sample of the actual data files that the AREST code writes.

Table B.1 shows the containment information for the first ten simulated waste packages. This file, FOR040.DAT, was created by writing to unit 40.

The three data files created by the AREST code that contain the release rates (FOR041.dat, FOR042.DAT, and FOR043.DAT) look very much alike. For each nuclide being used, several release parameters are written to each file. The release parameters are 1) the half life,  $T_{1/2}$ , 2) the concentration at the waste form surface,  $C_{sat}$ , 3) the retardation coefficient in the packing,  $K_1$ , 4) the retardation coefficient in the host rock,  $K_2$ , 5) the porosity in the packing,  $E_1$ , 6) the porosity in the host rock,  $E_2$ , 7) the 1000-year inventory,  $INVEN$ , and 8) the distribution of the nuclide in the release models (matrix, gap, grain boundary, crud, and clad). The time steps in these three files are initialized in the the routine  $RTIME$ . To modify the time steps refer to Appendix T.

Table B.2 lists the release rates for C-14 from the first simulated failed waste package. The times in this file (FOR041.DAT) are the time after container failure for the crud model, the time after container failure after resaturation for the clad model, and the time after containment failure after resaturation for the remaining models (matrix and gap/grain boundary). These release rates are used to calculate the release for the other simulated waste packages if the vector approach is chosen (specify 1 in the using vector approach option Table A.1) in the input file. This means that the AREST code calculates the release for only one waste package and uses this release profile, augmented with the failure time, to get the release for the remaining waste packages. This option is recommended, because of the run time it saves, if there are no time-temperature dependent variables in the release calculations other than inventory and solubility.

The average release rate for the repository, in parts/year, is output to the FOR042.DAT data file with C-14 of the file shown in Table B.3. The time in this file is the time since repository closure. The average release rate is calculated by summing the release rates for the individual simulated waste packages and dividing this result by the number of waste packages. Note that the release from the crud occurs much sooner than the release from any of the other release models. This happens because crud exists on the outer surface of the zircaloy cladding and can be released as a gas. The other sources of release do not occur until resaturation of the waste form.

Table B.4 contains the total amount released from the repository (FOR043.DAT). The amount released is calculated by summing the release rates for all the different release models, integrating this total average release from the repository, in parts/year, and multiplying this result by the total inventory of the nuclide in the repository. The results are displayed in several different units.

TABLE B.1. Waste Package Containment

<u>Waste Package</u>	<u>Cladding Failure</u>			<u>Container Failure</u>	
	<u>Time</u>	<u>Mechanism</u>	<u>Temperature</u>	<u>Time</u>	<u>Temperature</u>
1	9.17E+02	Normal d.	114.94	9.17E+02	114.94
2	8.41E+02	Normal d.	132.42	8.41E+02	132.42
3	9.80E+02	Normal d.	115.03	9.80E+02	115.03
4	8.86E+02	Normal d.	121.58	8.86E+02	121.58
5	1.10E+03	Normal d.	111.88	1.10E+03	111.88
6	9.54E+02	Normal d.	117.70	9.54E+02	117.70
7	1.38E+03	Normal d.	102.23	1.38E+03	102.23
8	9.31E+02	Normal d.	120.05	9.31E+02	120.05
9	9.70E+02	Normal d.	112.17	9.70E+02	112.17
10	1.01E+03	Normal d.	115.30	1.01E+03	115.30

**TABLE B.2. Individual Waste Package Release**

C-14

T1/2 (yrs)	Csat (g/m**3)	K1	K2	E1	E2	Inven (g/pack)
5.73E+03	1.40E+00	1.00E+00	1.00E+00	5.00E-01	1.00E-01	5.01E-01
WP #	% Matrix	% Gap	% Grain	% Crud	% Clad	
1	3.50E+01	1.00E+00	0.00E+00	1.00E+00	6.30E+01	
Time (yrs)	Matrix (1/yr)	Matrix (cm**3/s)	Gap (1/yr)	Crud (1/yr)	Clad (1/yr)	Cumulative (1/yr)
0.001	3.66E-10	4.15E-12	2.27E-05	7.03E-03	1.76E-14	7.05E-03
0.020	1.03E-07	1.17E-09	4.57E-03	8.31E-05	3.50E-12	4.65E-03
0.030	1.20E-07	1.36E-09	4.31E-03	4.53E-05	3.07E-12	4.36E-03
0.040	1.31E-07	1.48E-09	4.17E-03	2.94E-05	2.68E-12	4.20E-03
0.050	1.35E-07	1.53E-09	4.00E-03	2.11E-05	2.37E-12	4.03E-03
0.060	1.34E-07	1.52E-09	3.90E-03	1.60E-05	2.14E-12	3.92E-03
0.070	1.30E-07	1.48E-09	3.77E-03	1.27E-05	1.97E-12	3.78E-03
0.080	1.25E-07	1.41E-09	3.64E-03	1.04E-05	1.83E-12	3.65E-03
0.090	1.19E-07	1.35E-09	3.57E-03	8.72E-06	1.71E-12	3.58E-03
0.100	1.13E-07	1.28E-09	3.47E-03	7.45E-06	1.62E-12	3.48E-03
0.200	7.33E-08	8.31E-10	2.98E-03	2.63E-06	1.15E-12	2.98E-03
0.300	5.58E-08	6.33E-10	2.59E-03	1.43E-06	9.56E-13	2.59E-03
0.400	4.64E-08	5.26E-10	2.30E-03	9.31E-07	8.44E-13	2.30E-03
0.500	4.05E-08	4.59E-10	2.07E-03	6.66E-07	7.69E-13	2.08E-03
0.600	3.64E-08	4.13E-10	1.87E-03	5.07E-07	7.14E-13	1.87E-03
0.700	3.34E-08	3.78E-10	1.54E-03	4.02E-07	6.72E-13	1.54E-03
0.800	3.10E-08	3.51E-10	1.42E-03	3.29E-07	6.38E-13	1.42E-03
0.900	2.91E-08	3.30E-10	1.31E-03	2.76E-07	6.09E-13	1.31E-03
1.000	2.75E-08	3.12E-10	1.22E-03	2.35E-07	5.86E-13	1.22E-03
10.000	9.22E-09	1.05E-10	7.75E-05	7.40E-09	2.90E-13	7.75E-05
100.000	3.82E-09	4.33E-11	3.03E-06	2.18E-10	2.00E-13	3.04E-06
1000.000	2.17E-09	2.46E-11	8.83E-08	1.60E-12	1.68E-13	9.05E-08
10000.000	6.17E-10	6.99E-12	9.43E-10	9.17E-13	4.73E-14	1.56E-09
100000.000	7.82E-15	8.87E-17	5.57E-16	8.43E-17	8.54E-19	8.47E-15

TABLE B.3. Average Release Rate from the Repository

C-14

T1/2 (yrs)	Csat (g/m**3)	K1	K2	E1	E2	Inven (g/pack)
5.73E+03	1.40E+00	1.00E+00	1.00E+00	5.00E-01	1.00E-01	5.01E-01

WP #	% Matrix	% Gap	% Grain	% Crud	% Clad
1000	3.50E+01	1.00E+00	0.00E+00	1.00E+00	6.30E+01

<u>Time (yrs)</u>	<u>Matrix (1/yr)</u>	<u>Gap (1/yr)</u>	<u>Grain (1/yr)</u>	<u>Crud (1/yr)</u>	<u>Clad (1/yr)</u>
500.000	0.00E+00	0.00E+00	0.00E+00	6.46E-09	0.00E+00
600.000	0.00E+00	0.00E+00	0.00E+00	8.89E-08	0.00E+00
700.000	0.00E+00	0.00E+00	0.00E+00	2.43E-07	0.00E+00
800.000	0.00E+00	0.00E+00	0.00E+00	8.24E-07	0.00E+00
900.000	0.00E+00	0.00E+00	0.00E+00	1.94E-06	0.00E+00
1000.000	0.00E+00	0.00E+00	0.00E+00	7.65E-06	0.00E+00
1100.000	5.78E-10	1.14E-05	0.00E+00	1.68E-06	1.12E-14
1200.000	8.63E-10	1.56E-05	0.00E+00	2.38E-06	2.12E-14
1300.000	2.23E-09	5.46E-05	0.00E+00	5.32E-06	5.51E-14
1400.000	3.02E-09	6.66E-05	0.00E+00	2.19E-06	8.99E-14
1500.000	3.72E-09	7.15E-05	0.00E+00	1.02E-06	1.29E-13
1600.000	5.55E-09	9.33E-05	0.00E+00	1.51E-08	1.82E-13
1700.000	5.09E-09	5.06E-05	0.00E+00	1.07E-08	1.93E-13
1800.000	4.25E-09	1.89E-05	0.00E+00	6.54E-09	1.93E-13
1900.000	3.64E-09	1.03E-05	0.00E+00	4.61E-09	1.88E-13
2000.000	3.01E-09	7.47E-07	0.00E+00	3.49E-09	1.83E-13
3000.000	2.36E-09	5.78E-08	0.00E+00	8.76E-10	1.55E-13
4000.000	2.06E-09	2.13E-08	0.00E+00	3.56E-10	1.33E-13
5000.000	1.79E-09	1.08E-08	0.00E+00	1.89E-10	1.14E-13
6000.000	1.55E-09	6.43E-09	0.00E+00	1.14E-10	9.87E-14
7000.000	1.35E-09	4.17E-09	0.00E+00	7.49E-11	8.58E-14
8000.000	1.17E-09	2.85E-09	0.00E+00	5.19E-11	7.50E-14
9000.000	1.02E-09	2.03E-09	0.00E+00	3.78E-11	6.57E-14
10000.000	8.91E-10	1.49E-09	0.00E+00	2.81E-11	5.77E-14
100000.000	5.68E-15	1.68E-16	0.00E+00	3.13E-17	5.49E-19

**TABLE B.4. Total (Cumulative) Release from the Repository**

**C-14**

T1/2 (yrs)	Csat (g/m**3)	K1	K2	E1	E2	Inven (g/pack)
5.73E+03	1.40E+00	1.00E+00	1.00E+00	5.00E-01	1.00E-01	5.01E-01
WP #	% Matrix	% Gap	% Grain	% Crud	% Clad	
1000	3.50E+01	1.00E+00	0.00E+00	1.00E+00	6.30E+01	
Time (yrs)	Amount (g)	Amount (Ci/1000MTU)	Amount (mol/cm**2)	Amount (mol)		
500.000	6.05E-04	8.34E-04	4.66E-10	4.32E-05		
600.000	7.78E-03	1.07E-02	5.99E-09	5.56E-04		
700.000	4.79E-02	6.61E-02	3.69E-08	3.42E-03		
800.000	8.03E-02	1.11E-01	6.19E-08	5.74E-03		
900.000	2.57E-01	3.54E-01	1.98E-07	1.84E-02		
1000.000	3.61E-01	4.97E-01	2.78E-07	2.58E-02		
1100.000	9.30E-01	1.28E+00	7.17E-07	6.65E-02		
1200.000	1.74E+00	2.40E+00	1.34E-06	1.25E-01		
1300.000	3.46E+00	4.77E+00	2.67E-06	2.47E-01		
1400.000	6.52E+00	8.98E+00	5.02E-06	4.66E-01		
1500.000	9.94E+00	1.37E+01	7.66E-06	7.10E-01		
1600.000	1.38E+01	1.90E+01	1.06E-05	9.84E-01		
1700.000	1.81E+01	2.50E+01	1.40E-05	1.29E+00		
1800.000	2.05E+01	2.83E+01	1.58E-05	1.47E+00		
1900.000	2.11E+01	2.91E+01	1.63E-05	1.51E+00		
2000.000	2.14E+01	2.94E+01	1.65E-05	1.53E+00		
3000.000	2.15E+01	2.97E+01	1.66E-05	1.54E+00		
4000.000	2.16E+01	2.97E+01	1.66E-05	1.54E+00		
5000.000	2.16E+01	2.97E+01	1.66E-05	1.54E+00		
6000.000	2.16E+01	2.97E+01	1.66E-05	1.54E+00		
7000.000	2.16E+01	2.97E+01	1.66E-05	1.54E+00		
8000.000	2.16E+01	2.97E+01	1.66E-05	1.54E+00		
9000.000	2.16E+01	2.97E+01	1.66E-05	1.54E+00		
10000.000	2.16E+01	2.97E+01	1.66E-05	1.54E+00		
100000.000	2.16E+01	2.98E+01	1.66E-05	1.54E+00		

APPENDIX C

ARESTIN MODULE

## APPENDIX C

### ARESTIN MODULE

#### PURPOSE

ARESTIN is the main input module of AREST. ARESTIN reads data from the primary and secondary data files, and provides a user-interface for editing the data. The module also allows the user to save modified data sets.

#### DESCRIPTION

The ARESTIN module was designed to be flexible in that it could be easily adapted to a wide range of input requirements, without major changes to input routines and without unduly constraining the format of input files. The basic idea used in the implementation of ARESTIN is to read data into strings, process the strings, and read values from the strings. This method provides the needed flexibility, whereas directly reading the values does not.

ARESTIN is highly structured; it consists of numerous subroutines, the most important being those that are application-independent, in the sense that no assignments are made to variables that get used outside of ARESTIN. These assignments are instead made in higher-level routines, typically after calls to application-independent input/editing routines. In effect, the application-independent routines are used as a "toolbox" from which application-dependent routines can be constructed with minimal effort. This also provides flexibility.

Because ARESTIN is designed to be flexible, the implementation is probably best described in terms of how the various mechanisms are typically used, rather than in terms of the specific calling sequences currently used in AREST.

Consider the overall process. First, the data must be read in. Since ARESTIN allows the user to have comments (i.e. non-data) in the data files, these comments must be separated from the data and saved. The data is then edited and otherwise processed. Finally, the non-data and revised data are recombined and saved.

Next, consider the process of reading the data. ARESTIN distinguishes between two types of data lines, numeric and character, on the basis of the first data item on each line. Also, as mentioned above, non-data lines are allowed. ARESTIN searches for data using the following rules:

- 1) All data lines must begin in the first column.
- 2) Numeric data is assumed to start with a digit, sign, or decimal point.
- 3) When searching for character data, ARESTIN first looks for a line ending in a colon. The next line with a character in the first column is assumed to be the character data line.
- 4) Multiple lines of similar data (e.g., a series of temperature values), where the exact number of lines are not known beforehand, must be consecutive and must be delimited (terminated) by a blank line. Such groups of lines are referred to in this appendix as "blocks".
- 5) Any lines encountered during the search that do not match the descriptions set forth for the desired type of data line are assumed to be comments and are ignored. However, all comment lines are reproduced in the file used for the revised data set.

The data structures used for input are fairly simple. Data lines are written to the DATALIN 80-character string array. A parallel array, ENTRIES, is used to keep track of the number of data items in each DATALIN entry. Non-data lines are stored in a temporary file. Data and non-data are separated as they are read.

Typically, one of two routines, GETDATA (for single data lines) or DOBLOCK (for blocks), is called for both reading and editing. These two routines handle most common data configurations. In this respect, they are the most important toolbox routines and merit closer examination.

SUBROUTINE GETDATA (GETLIN, N, PERLINE, FORM)

This routine is used to read/edit non-block data. GETLIN is a variable which contains the name of the subroutine used to get individual data lines: GETLIN equals GETSTR for character data lines, and GETNSTR for numeric data

lines. N is the number of data lines to process, and cannot exceed the ARESTIN constant, MAXEDRM (currently 20). PERLINE is the number of data items per line, and is used to fill the ENTRIES array, so the number should be the same for all data lines in the current read. If the number of items per line varies, ENTRIES is initialized, and GETDATA is called with PERLINE = 0. FORM is the number of the format statement used by the MENU subroutine to display the data.

**SUBROUTINE DOBLOCK (GETLIN1, PERLINE, FORM, COUNT)**

This routine is used to read/edit block data. GETLIN1 is the name of the routine (either GETSTR or GETNSTR) used to locate the first line of the block. PERLINE, again, is the number of data items per line, but, in this case, must be constant throughout the block. FORM is the format number to use. COUNT returns with the number of lines in the block (after editing). There is no limit on this number.

Now, consider the editing process. As mentioned above, GETDATA and DOBLOCK are called for both reading and editing. These routines store each data line in the DATALIN array, which is common to all of ARESTIN. While the calling sequence varies between the two routines, both eventually cause the DATAED subroutine to be invoked. DATAED is the driver for all editing; it calls other routines to display the data in a menu and it executes editing commands, and perform those commands. Each command is implemented by a single subroutine which, in most cases, calls still other subroutines or functions. Some of these routines perform data transfers which aid in reconstructing the revised data set. Most others involve string manipulations of the data; e.g. individual data items are edited by substring replacement.

After editing, the data is ready for use by the calling routine. GETDATA leaves the data in the DATALIN array. DOBLOCK deposits its edited data in the INRESLT file.

Finally, we should consider the numerous data transfers which occur in ARESTIN. While the overall paths of these transfers can be complicated, and may vary greatly, these paths are broken into a series of steps, each of which consists of relatively simple i/o. Most of these transfers are controlled by the toolbox routines. An important exception concerns the subroutine KILLBLK.

DOBLOCK creates the INRESLT file from which the calling routine reads and processes block data. After the data is processed, INRESLT is expendable, and KILLBLK should be called to delete it.

This concludes a simplified description of ARESTIN. Some additional detail is listed below in the Modifications section. The interested reader can also consult the source code, which is well documented.

## MODIFICATIONS

One of the main goals of this appendix is to provide information which will aid in adapting the ARESTIN routines for use with an updated AREST data configuration. To do this the programmer must know how to use the main toolbox routines, the global control variables, and the menu system.

The main toolbox routines, GETDATA and DOBLOCK (in conjunction with KILLBLK), have already been described. The sequence of calls to these routines depends, of course, on the contents of the input files.

To control data flow, ARESTIN uses several COMMON variables: INFILE, OUTFILE, HEADFIL, EDITING, EDSIZE, MENUID, and NXTMENU. INFILE and OUTFILE designate the current input and output files. HEADFIL indicates where header information in the input file is to be copied. EDITING is .TRUE. iff editing is enabled. EDSIZE differentiates between block and non-block data. MENUID differentiates between two similar menu subroutines. Finally, NXTMENU is the number of the next menu to display. The majority of these are set automatically by the toolbox routines. The most important exceptions are INFILE and OUTFILE which the programmer must set in the calling routines, enabling switching between primary and secondary data files if the user chooses. These files must also be opened and closed in calling routines.

Menus must be designed or redesigned to accommodate new data configurations, so knowledge of the menu system is necessary. First, menus are identified by their format numbers (in the MENU subroutine). On each call, MENU is passed the number of lines in DATALIN and the menu identification number. The DATALIN strings are then displayed, one string per menu option, with options being labelled with consecutive letters (A, B, C, etc.).

Menu identification numbers increase in order of appearance. This convention is used in implementing the GOTO editing command. Recall that GOTO allows the user to specify a menu for ARESTIN to display. This is accomplished by setting NXTMENU equal to the targeted number. The menus are still processed in order, but those with numbers less than NXTMENU are not displayed.

To add a menu, one must define a MENU subroutine format statement using A, B, C, etc., and assign it a number based on when it appears. To add a GOTO stop, one must change the PARAMETER and DATA initialization in the SETJUMP procedure.

It should be possible to make most necessary modifications using the information in this appendix. However, it must again be noted that this is a fairly cursory explanation of ARESTIN; for more detail, the user is urged to examine the commented source code.

#### ARGUMENT LIST

Flags passed as arguments are all of the form 0=false/1=true, and are described in terms of the true value.

- IFSCF is a flag which is true when corrosion is to be simulated by a distribution, the alternative being simulation by a model.
- PFAIL0 is the probability of prefailure for a single package.
- NIFAIL is a limit on the number of prefailures.
- MIC is an array of flags indicating if the particular corrosion model is used.
- MSIMCF indicates the distribution type used to simulate corrosion (when applicable):
  - 1 = point
  - 2 = uniform
  - 3 = normal
  - 4 = exponential
- SIMCF1,  
SIMCF2 are coefficients for the MSIMCF distribution.

IFCO is a flag indicating when only corrosion (no release) is to be simulated.

MIR is an array of flags indicating which release models are being used.

IFSOLF is a flag for using the UO2 solubility function.

IFCONG is a flag for assuming congruent release of UO2.

ISIM is an array indicating the distribution type used to simulate releases:  
 0 = not used  
 1 = uniform  
 2 = normal

PSIM is an array of coefficients used with ISIM distributions.

NPERC is the number of percentiles to use in the output

IFMATR is a flag indicating use of a array to store releases (in which case the releases are assumed to be time-independent).

NRAD is the number of radionuclides being considered.

NUMWP is the number of waste packages being simulated.

IFOREL is a flag for printing release figures on the output.

IFOUO2 is a flag for printing UO2 solubility figures.

NTCUM is the number of cells in the release cumulation array.

IPPT indicates which precipitation parameter to solve for:  
 1 = XPPT, 2 = CPPT, 3 = TPPT

NPPT is the number of (XPPT, CPPT, TPPT) sets read.

XPPT is the packing distance percentage at which precipitation will occur.

CPPT is the concentration (in % of surface concentration) at which precipitation will occur.

TPPT is the time at which precipitation will occur.

ROUTINES CALLED .

INENVIR, ASSIGNS, INCOR, GETREL, GETDATA, RADION, RTIME, INPPT, SKIPALL, RENAME, UPCASE, and TRUELEN are called directly by ARESTIN. GETDATA, DOBLOCK, GETSTR, GETNSTR, KILLBLK, DATAED, MENU, and SETJUMP are only called indirectly,

but are included here for completeness since they are referred to elsewhere in the appendix.

INENVIR reads the environmental/temperature data. Modifications to this routine are discussed in Appendix T (USER-SUPPLIED ROUTINES).

ASSIGNS sets some constants.

INCOR reads corrosion data.

GETREL reads release data.

GETDATA reads and edits single lines of data.

RADION reads radionuclide data.

RTIME initializes the time arrays.

INPPT reads precipitation data.

SKIPALL reads and duplicates any comment lines remaining in the input file.

RENAME renames or deletes files.

UPCASE returns the uppercase version of a character.

TRUELEN returns the length of a string, not counting any blanks padding the string end.

GETDATA is used to read and edit non-block data.

DOBLOCK is used to read and edit block data.

GETSTR gets a character data line.

KILLBLK deletes the INRESLT temporary file.

DATAED is the driver for all editing, calling other routines to display and edit the data.

MENU displays the current editing menu.

SETJUMP displays the GOTO menu, gets a selection, and uses that selection to set the next displayed editing menu.

### TEST PLAN

The ARESTIN module has been tested as a whole, using typical AREST input data sets. The testing was done by matching the input to the output. The basic testing procedure for a given data set is as follows:

1. Use ARESTIN to input the data set.
2. Issue a series of ARESTIN commands, including each type of legal command (edit, insert, delete, etc.) as well as illegal commands (to check error flagging).

3. Check echo files (produced by ARESTIN) to see that the resultant data is as expected.

This test procedure has been used for a number of test runs. Similar tests have been performed for the case where the data is used "as is", i.e., without editing.

APPENDIX D

ENVIR MODULE

## APPENDIX D

### ENVIR MODULE

#### PURPOSE

The ENVIR module is composed of the subroutines ENVIR, SENVIR, PHASE, and TRESAT, and the function TEMPF. These subroutines, in the order listed above, perform the following tasks: 1) calculate the time-dependent environmental parameters (such as temperature), 2) calculate the temperature vector for a simulated heat loading, 3) return the environmental phase, steam or water, and 4) calculate the time of resaturation (or wetting for an unsaturated repository). The function TEMPF calculates the temperature for the current time step.

#### DESCRIPTION

Subroutine ENVIR adjusts the temperature for the current time by calling the function TEMPF.

Subroutine SENVIR simulates the time-temperature vector for a given heat load. The subroutine calls function RANDOM to return the uniform random deviate [0,1). SENVIR then uses the heat loading information and the initial temperature, to compute the time-temperature distribution(a).

Subroutine PHASE determines whether the current environment is steam or liquid. If the current time is greater than the time of resaturation or wetting, then the current environment is liquid. Otherwise, the current environment is steam.

Function TEMPF calculates the temperature at the current time step. If the current time is not in the TIMES array, the subroutine uses linear interpolation to compute the temperature for the current time.

- 
- (a) Liebetrau, A. M., M. J. Apped, D. W. Engel, M. K. Altenhofen, D. M. Strachan, C. R. Reid, C. F. Weindisch, R. L. Erikson, and K. I. Johnson. 1987. The Analytical Repository Source-Term (AREST) Model: Description and Document. PNL-6346, Pacific Northwest Laboratory, Richland, Washington.

Subroutine TRESAT calculates the time of resaturation or wetting. This subroutine assumes a minimum 300-year containment. The only limitation to this code is that if the temperature of resaturation is very large, i.e. 500, then the time of resaturation is zero.

#### ARGUMENT LIST

##### Subroutine ENVIR:

TIME is the current time.  
ITIME is the array index for the current time.  
IPHASE is the current environment.  
If IPHASE = 0 then steam environment.  
If IPHASE = 1 then liquid environment.

##### Subroutine PHASE:

TIME is the current time.  
IPHASE is the current environment.  
If IPHASE = 0 then steam environment.  
If IPHASE = 1 then liquid environment.

##### Function TEMPF:

TIME is the current time.  
ITIME is the array index for the current time.  
TFRAC is the fraction of temperature.  
TEMP is the temperature array.  
TIMES is the time array.  
NST is the number of time steps.

#### LOGIC

##### ENVIR

1. Adjusts temperature for this time step.  
(TEMPF)

##### PHASE

1. Initialize variable IPHASE=0 (steam environment).

2. If current time TIME is greater than the time of resaturation (TSAT) then IPHASE = 1 (liquid environment).
3. Set the current environment IPHASE.

#### SENVIR

1. Calculate uniform random deviate [0,1) (Function RANDOM).
2. Find heat loading.
3. Get initial temperature.
4. Compute time-temperature distribution.

#### TEMPF

1. Get the time and temperature at the current time step.
2. If the current time is not equal to the current time step, then do the following:
  - a. Search the time array (TIMES) for the first time at or past the current time.
  - b. Use linear interpolation to compute the temperature at the current time.

#### TRESAT

1. Initialize temporary variables.
2. Read time array (TIMES) and temperature array (TEMPS) until the current temperature from TEMPS is less than or equal to the temperature at resaturation or wetting, and the current time from TIMES is greater than or equal to 300, assuming a minimum 300-year containment.
  - a. If previous temperature from TEMPS is not equal to the current temperature from TEMPS then calculate FRAC.
  - b. If FRAC is negative, set to 0.
  - c. Calculate the time of resaturation using linear interpolation or wetting.
3. If time of resaturation or wetting is negative, set to 0.

#### TEST PLAN

To test ENVIR, which calls TEMPF to calculate the current temperature, the following cases were run:

1. If the current time (TIME) was equal to the current time step (TIMES(ITIME)), then the temperature at the current time is equal to the value in the temperature array at the current time step (TEMP(ITIME)).
2. If the current time (TIME) is not equal to the current time step (TIMES(ITIME)), then the temperature at the current time is calculated using linear interpolation. This was verified with a hand calculation.

Subroutine PHASE determines and returns the current environment. This subroutine was tested as follows:

1. Set the current time (TIME) less than or equal to the time of resaturation. The current phase (IPHASE) is equal to 0, for the steam environment.
2. Set the current time (TIME) greater than the time of resaturation. The current phase (IPHASE) is equal to 1, for the liquid environment.

SENVIR simulates a temperature vector for a given head load. SENVIR first calls function RANDOM to return the uniform random deviate. The testing of function RANDOM is documented in Appendix H. SENVIR was run with a typical data set, and the following was calculated and tested by hand calculations:

1. heat loading information,
2. initial temperature,
3. time-temperature distribution.

Subroutine TRESAT calculates the time of resaturation or wetting, TSAT. This code assumes a 300-year containment. The code was tested as follows:

1. The calculation for the time of resaturation (TSAT) was verified by a hand calculation.
2. The following additional tests cases were run:
  - a. TSAT is negative. TSAT is set to 0.0.
  - b. TSAT is positive. TSAT is not changed.

APPENDIX E

GRDH20 MODULE

## APPENDIX E

### GRDH20 MODULE

#### PURPOSE

The GRDH20 module contains two subroutines, INGRDH20 and GRDH20. The subroutine INGRDH20 loads groundwater composition data from an input data file into a table. The GRDH20 subroutine uses linear interpolation to calculate the groundwater composition for a simulated temperature.

#### DESCRIPTION

Subroutine INGRDH20 reads the groundwater composition variables(a) from the data file ENV SOL.DAT, and stores them in array SWATER that is stored in the common block WATER. The variables in the data file are read using a set format. Table E.1 shows the variables and units required; these are listed in the order in which they are stored in the array SWATER. In other words, the temperature is stored in row 1, pH is stored in row 2, Eh is stored in row 3, and so on. Variable names are read from columns one to six, while the concentrations are read from columns 16 to 25. A different groundwater composition is read in for each different temperature in the input file. The variables for temperature, pH, and Eh must exist for each groundwater composition or the program will stop. Blank lines between records are arbitrary, since the routine reads a given data set (one set of data for each groundwater composition) until it finds the next occurrence of the variable TEMPERATURE.

Subroutine GRDH20 calculates the groundwater composition for the simulated temperature, TEMP. The groundwater composition is calculated using linear interpolation of the tabled values. If the simulated temperature is

---

(a) Liebetrau, A. M., M. J. Apted, D. W. Engel, M. K. Altenhofen, D. M. Strachan, C. R. Reid, C. F. Weindisch, R. L. Erikson, and K. I. Johnson. 1987. The Analytical Repository Source-Term (AREST) Model: Description and Document. PNL-6346, Pacific Northwest Laboratory, Richland, Washington.

TABLE E.1. Sample Input Data File (ENVSOL.DAT)

TEMPERATURE	25.00	degrees celsius
PH	7.20000	
EH	0.79300	volts
CL-	0.2012D-03	molal conc
P04---	0.1500D-09	molal conc
S04--	0.1805D-03	molal conc
C03--	0.2003D-05	molal conc
NO3-	0.0000D+00	molal conc
O2	0.0000D+00	molal conc
HC03-	0.2210D-02	molal conc
BR-	0.0000D+00	molal conc
F+	0.0000D+00	molal conc
SI	0.0000D+00	molal conc
ZN++	0.0000D+00	molal conc
SR+	0.0000D+00	molal conc
MG++	0.0000D+00	molal conc
AL+++	0.0000D+00	molal conc
CA++	0.0000D+00	molal conc
K+	0.0000D+00	molal conc
NA+	0.0000D+00	molal conc

less than all temperatures in the table (array SWATER), then data for the lowest temperature in the table is used. Similarly, if the simulated temperature is greater than all temperatures in the table, then the data for the highest temperature in the table is used.

#### ROUTINES CALLED

Subroutine INGRDH20 uses Function CONVCTR to convert character variables in the input file (ENVSOL.DAT) to real numbers.

#### ARGUMENT LIST

TEMP is the temperature being simulated.

#### LOGIC

GRDH20

1. Find the two temperatures from the table (array SWATER) that surround the simulated temperature (TEMP).
  - a. Retrieve the simulated temperature.

- b. See if stored temperature is greater than simulated temperature.
2. If simulated temperature TEMP is greater than the last value in the table, then use the maximum values in the table.
3. Use linear interpolation to calculate the groundwater composition.

#### INGRDH2O

1. Open data file ENV SOL.DAT. If file is not found, stop run.
2. Read file until 'TEMPERATURE' is found. If no temperature is found, stop run.
3. Store initial temperature.
4. Initialize temporary variables.
5. Read rest of file and store concentrations for each temperature.
  - a. If Eh and pH were not entered for each temperature, stop run.
  - b. Store groundwater compositions for each temperature into SWATER.

#### TEST PLAN

The testing of GRDH2O included checking the following cases by hand calculations:

- a. If TEMP is less than the lowest temperature in SWATER, then the groundwater composition for TEMP is equal to the groundwater composition for the lowest temperature in SWATER.
- b. If TEMP is equal to a temperature in SWATER, then the groundwater composition for TEMP is equal to the groundwater composition for that temperature in SWATER.
- c. If TEMP is between two temperatures in SWATER, then the groundwater composition is calculated using linear interpolation as follows:
  - LOWTEMP = the lower of the two temperatures surrounding TEMP
  - HIGHTEMP = the higher of the two temperatures surrounding TEMP
  - DIFFTEMP = HIGHTEMP - LOWTEMP (the difference between the two temperatures surrounding TEMP)
  - SLOPE = (TEMP - LOWTEMP)/DIFFTEMP.

Then, for each composition:

- LOWCOMP = the composition at LOWTEMP
- HIGHCOMP = the composition at HIGHTEMP
- COMPOSITION = LOWCOMP + SLOPE \* (HIGHCOMP - LOWCOMP).

- d. If TEMP is greater than the highest temperature in SWATER, then the groundwater composition for TEMP is equal to the groundwater composition for the highest temperature in SWATER.

The testing of subroutine INGRDH20 consisted of verifying by inspection that the data in SWATER corresponds with data in ENVSQL.DAT. The data in array SWATER are as follows:

Row 1	=	Temperature
Row 2	=	pH
Row 3	=	Eh
Row 4	=	Cl-
Row 5	=	PO4---
Row 6	=	SO4--
Row 7	=	CO3--
Row 8	=	NO3-
Row 9	=	O2
Row 10	=	HCO3-
Row 11	=	Br-
Row 12	=	F+
Row 13	=	Si
Row 14	=	Zn++
Row 15	=	Sr+
Row 16	=	Mg++
Row 17	=	Al+++
Row 18	=	Ca++
Row 19	=	K+
Row 20	=	Na+

APPENDIX F

CORRODE MODULE

## APPENDIX F

### CORRODE MODULE

#### PURPOSE

CORRODE is the driver for the waste package containment phase of the AREST code. CORRODE is called by the main AREST program, and uses the modules PREFAIL, DISTRIB, and MODEL.

#### DESCRIPTION

CORRODE is called once for each waste package. It simulates corrosion of the package, and returns a time and cause of failure. The package may be prefailed, meaning that it was defective or damaged at the time of emplacement. If the package is not prefailed, a corrosion time is determined, either via modeling or via a statistical distribution. CORRODE is essentially a driver; it calls other routines to do this work.

#### ROUTINES CALLED

CORRODE calls three subroutines: PREFAIL, DISTRIB, and MODEL. PREFAIL determines if the package has prefailed, DISTRIB simulates failure times from a distribution, and MODEL simulates failure times via corrosion models. Each of these routines is documented in a separate appendix.

#### ARGUMENT LIST

STEPS	is the number of time steps used when simulating corrosion via modeling.
IFSCF	indicates the type of corrosion simulation: 0 => Model 1 => Distribution
KIND	indicates the type of distribution (if any): 1 => Point 2 => Uniform 3 => Normal 4 => Exponential

A, B are coefficients for the distribution (if any):

KIND	A	B
1	Fail time	--
2	Minimum fail time	Maximum fail time
3	Mean fail time	Standard deviation
4	Minimum fail time	Lambda

NIFAIL is a minimum number of containment prefailures.

PFAILO is the probability of containment prefailure for each package.

CORROSN is an array indicating which types of corrosion are being considered. It is indexed by corrosion type code:

1 => Uniform corrosion

2 => Stress corrosion

3 => Pitting corrosion

4 => Fracture corrosion

5 => Cladding corrosion

Array entries are numerical flags:

0 => The corrosion type is not used.

1 => The corrosion type is used.

TIMES is the time value array.

LIMIT is an array of maximum corrosion values (depth in most cases), and is indexed by corrosion type code (see CORROSN above).

MECHF is the code for the cause of containment failure:

-4 => Exponential distribution

-3 => Normal distribution

-2 => Uniform distribution

-1 => Point distribution

0 => No failure

1 => Uniform corrosion

2 => Stress corrosion

3 => Pitting corrosion

4 => Fracture corrosion  
 5 => Cladding corrosion  
 6 => Prefailure

**TFAIL** is the time of containment failure (infinity if no failure has occurred).

**MECHFC** is the code for the cause of container failure. See MECHF above.

**TFAILC** is the time of container failure (infinity if no failure has occurred).

**STATUS** is the code for the current status of the container and containment. Constants have been defined in the STATUS.PAR include file, and are used in referencing status levels.

<u>Value</u>	<u>Constant</u>	<u>Meaning</u>
0	OK	The package is OK; neither container nor containment has failed.
1	NOCONT	The container wall has failed, but the cladding is intact.
2	NOCLAD	The cladding has failed, but the container wall is intact.
3	FAIL	Both container and cladding have failed; release will occur.

MECHF, TFAIL, MECHFC, TFAILC, and STATUS are initialized before the call to CORRODE and may be changed by CORRODE. All other arguments are inputs only.

#### LOGIC

1. Simulate prefailures  
(PREFAIL)
2. If containment did not prefail, then
  - a. Simulate via distribution or  
(DISTRIB)

- b. Simulate via model.  
(MODEL)

#### TEST PLAN

The only function performed by CORRODE is the driving of other subroutines. For this reason, the test plan for CORRODE consists only of testing its subroutines. The test plan for each of the subroutines, PREFAIL, DISTRIB, and MODEL are listed in the Appendices G, H, and I, respectively.

**APPENDIX G**

**PREFAIL MODULE**

## APPENDIX G

### PREFAIL MODULE

#### PURPOSE

PREFAIL determines if the current waste package should be considered to be prefailed, i.e., defective or damaged at the time of emplacement in the repository.

#### DESCRIPTION

PREFAIL uses a saved local variable to keep track of the number of prefailed packages. The first NIFAIL packages are considered prefailed, by default. Thereafter, each package has probability PFAILO of being prefailed. A uniformly random number on  $[0,1)$  is generated to simulate this probability.

#### ROUTINES CALLED

PREFAIL uses function RANDOM to generate the random number.

#### ARGUMENT LIST

NIFAIL	is a minimum number of containment failures.
PFAILO	is the probability of containment prefailure for each package.
MECHF	is the code for the cause of containment failure, and will be 6 if prefailure occurs.
MECHFC	is the code for the cause of container failure, and will be 6 if prefailure occurs.
TFAIL	is the time of containment failure (infinity if no failure has occurred).
TFAILC	is the time of container failure (infinity if no failure has occurred).
STATUS	is the code for the current status of the container and containment. Constants have been defined in the STATUS.PAR include file, and are used in referencing status levels.

<u>Value</u>	<u>Constant</u>	<u>Meaning</u>
0	OK	The package is OK; neither container nor containment has failed.
1	NOCONT	The container wall has failed, but the cladding is intact.
2	NOCLAD	The cladding has failed, but the container wall is intact.
3	FAIL	Both container and cladding have failed; release will occur.

All arguments are used as inputs to PREFAIL. MECHF, MECHFC, TFAIL, TFAILC, and STATUS may also have their values changed.

#### LOGIC

1. If the prefailure quota is not met or if a prefailure is randomly generated, then store container and containment failure data.  
(RANDOM)

#### TEST PLAN

PREFAIL is a straightforward routine to generate package prefailures, based on two factors: a minimum number of prefailures (NIFAIL), and the probability of prefailure (PFAILO). PREFAIL was tested as follows:

1. A test program was written to drive a large number of trials of PREFAIL, and output resultant failure times and mechanisms.
2. A second program was used to reproduce the sequence of pseudo-random numbers generated in step 1.
3. Results were hand-calculated from the random numbers and compared to the step 1 results.

APPENDIX H

DISTRIB MODULE

## APPENDIX H

### DISTRIB MODULE

#### PURPOSE

DISTRIB simulates containment failure times from a specified statistical distribution.

#### DESCRIPTION

DISTRIB performs a case selection on the KIND of distribution. Each type of distribution is handled separately with a few lines of code which mathematically define the distribution.

#### ROUTINES CALLED

DISTRIB calls functions RANDOM and NORMAL. RANDOM generates a uniformly random number on the interval  $[0,1)$ , and is based on the function RAN1(a). NORMAL generates a random normal deviate based on a random number returned from RANDOM, given a mean and standard deviation, and is based on the function GASDEV(a). The failure time and cause are then stored.

#### ARGUMENT LIST

KIND indicates the type of distribution:  
1 => Point  
2 => Uniform  
3 => Normal  
4 => Exponential  
A, B are coefficients for the distribution:

---

(a) Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterline. 1986. Numerical Recipes: The Art of Scientific Computing. Cambridge Press, New York, New York.

<u>KIND</u>	<u>A</u>	<u>B</u>
1	Fail time	--
2	Minimum fail time	Maximum fail time
3	Mean fail time	Standard deviation
4	Minimum fail time	Lambda

MECHF is the code for the cause of containment failure.

For distributions, the codes are:

-4 => Exponential distribution

-3 => Normal distribution

-2 => Uniform distribution

-1 => Point distribution

TFAIL is the time of containment failure (infinity if no failure has occurred).

MECHFC is the code for the cause of container failure.  
See MECHF above.

TFAILC is the time of container failure (infinity if no failure has occurred).

STATUS is the code for the current status of the container and containment. Constants have been defined in the STATUS.PAR include file, and are used in referencing status levels.

<u>Value</u>	<u>Constant</u>	<u>Meaning</u>
0	OK	The package is OK; neither container nor containment has failed.
1	NOCONT	The container wall has failed, but the cladding is intact.
2	NOCLAD	The cladding has failed, but the container wall is intact.
3	FAIL	Both container and cladding have failed; release will occur.

MECHF, TFAIL, MECHFC, TFAILC, and STATUS are initialized before the call to DISTRIB and may be changed by DISTRIB. All other arguments are inputs only.

## LOGIC

1. Simulate via one of the following:
  - a. Point distribution.
  - b. Uniform distribution.
  - c. Normal distribution.
  - d. Exponential distribution.
2. Store container and containment failure data.

## TEST PLAN

DISTRIB consists of straightforward computation built around "book" routines for random deviates. The following testing was performed:

1. A test program was written to drive several trials of DISTRIB using a normal distribution. The computations involved are typical of those for other distributions.
2. The sequence of pseudo-random numbers generated in step 1 was reproduced.
3. The normal deviates were hand-calculated from the random numbers and compared against the step 1 results.

**APPENDIX I**

**MODEL MODULE**

## APPENDIX I

### MODEL MODULE

#### PURPOSE

MODEL simulates the corrosion, or degradation, of a waste package. Arguments (some of which are passed by COMMON) are used to specify which types of corrosion to simulate and which models to use to simulate them.

#### DESCRIPTION

In this section we describe MODEL in a systematic way. First, we identify the classes and types of corrosion which MODEL considers. After that we describe, in general terms, the corrosion processes leading to waste package failure. Then we consider how MODEL incorporates these failure processes into its logic, with close attention paid to how each corrosion process is simulated. Finally, MODEL's data structures are discussed.

MODEL distinguishes between three classes of corrosion: uniform corrosion of the waste package container, nonuniform corrosion of the container, and corrosion of the waste form cladding. Uniform corrosion affects all portions of the container evenly. In contrast, nonuniform corrosion consists of various localized corrosive forces. The current version of the AREST code considers three nonuniform types: stress, pitting, and fracture. The degradation of the cladding is limited to only one process, in each environment, in this version of the AREST code. This limitation can be easily eliminated if the need arises.

In all, MODEL supports five types of corrosion or degradation: uniform, stress, pitting, fracture, and cladding. For each type, the AREST code input must specify which types are to be considered for the particular run; and for each type being considered, a particular model must be designated. From a programming standpoint, the selected model is unimportant in that MODEL uses the corrosion rate returned by the modeling subroutine, but does not care how that rate is determined.

Understanding how waste package failure can occur is one of the keys to understanding MODEL. Containment fails when both the container and cladding have been breached, allowing release of the waste form to begin. There are two ways that containment failure can occur. The first uniform corrosion. When the container wall fails due to uniform corrosion, it is assumed that the collapse of the container onto the waste form causes the cladding to be breached too. The result is containment failure. The second way that containment failure can occur is the combination of nonuniform and cladding corrosion. When nonuniform corrosion causes a breach in the container wall, the cladding becomes exposed to corrosive forces. Subsequent cladding failure will also constitute containment failure.

A few remarks may be helpful. The corrosion processes act concurrently, although cladding corrosion cannot start until the container fails. Note that container failure is not synonymous with containment failure; after nonuniform corrosion breaches the container wall, cladding prevents release of the waste form. However, exposure of the cladding to the environment allows cladding corrosion to start, leading to eventual release. It is unnecessary to simulate nonuniform corrosion after the wall is breached.

MODEL simulates all corrosion types via the same general procedure. First, MODEL computes the environmental conditions, including the temperature and state (gas or liquid) of the surroundings. This information is passed in a common block to the modeling subroutine. Model-dependent parameters may also be passed in COMMON blocks, to the routine, from input routines external to MODEL. MODEL then receives a corrosion rate from the modeling routine. This rate is numerically integrated (using the trapezoidal rule) over the most recent time step, and added to the previous corrosion depth to produce the current corrosion depth. If this depth exceeds a specified limit, then failure occurs. Interpolation is used to determine a more precise time of failure (the time when depth equaled the limit).

Depth limits are input values for all corrosion types except pitting. For pitting, a limit is computed for each package (see Appendix PITDPTH).

The logic flow of MODEL is fairly simple. First, initialization is performed. The routine then simulates corrosion, time step by time step. At each time step, MODEL updates the environmental conditions. Then MODEL

simulates either nonuniform or cladding corrosion. Nonuniform corrosion is simulated prior to breach of the container wall, while cladding corrosion is simulated after a breach. Finally, uniform corrosion is simulated time stepping continues until containment fails or until there are no more time steps. MODEL returns the time and mechanism of failure to the calling routine.

When the container wall breaches during a time step, cladding corrosion must be simulated for the post-breach portion of the step. This is accomplished by allowing the nonuniform corrosion routine to call the cladding corrosion routine.

There are few important data structures in MODEL. One such structure is necessary for the numerical integration used in corrosion simulation. The integration requires information (the time, rate, and depth for each corrosion type) for the previous time array entry. An array (PREV) is used to store this information. The only other important variable is STATUS, which MODEL uses primarily as a flag to indicate if the container wall has been breached. (For more information on STATUS or other arguments, see the Argument List section.)

#### ROUTINES CALLED

MODEL calls PHASE, ENVIR, INITCUM, CLADDNG, NONUNIF, and UNIFORM directly; and PUTPREV, GETPREV, BREAKDN, DRIVECO, INTEGRA, YINTERP, and user-selected modeling routines indirectly. PHASE and YINTERP are functions; the rest are subroutines.

PHASE determines if the package is currently in a liquid or gas environment.

ENVIR sets time-dependent environmental values.

INITCUM loads the PREV array for the initial time step.

CLADDNG drives the simulation of cladding corrosion.

NONUNIF drives the simulation of nonuniform corrosion types, determining which, if any, cause failure first. If failure by non-uniform corrosion occurs during a time step, then cladding corrosion is simulated for the remainder of the time step.

UNIFORM drives the simulation of uniform corrosion of the container.

PUTPREV saves the current values of time, corrosion rate, and corrosion depth, for use by the integration routine in the next pass of the time loop.

GETPREV retrieves the previously saved values of time, corrosion rate, and corrosion depth.

BREAKDN is used to simulate any corrosion type for a single time step. Simulation is as outlined in the description section.

DRIVECO is a driver used to select the desired corrosion-modeling routine.

INTEGRA performs trapezoidal rule integration.

YINTERP performs linear interpolation.

### ARGUMENT LIST

- STEPS is the number of time steps used when simulating corrosion via modeling.
- CORROSN is an array indicating which types of corrosion are being considered. It is indexed by corrosion type code:
- 1 => Uniform corrosion
  - 2 => Stress corrosion
  - 3 => Pitting corrosion
  - 4 => Fracture corrosion
  - 5 => Cladding corrosion
- Array entries are numerical flags:
- 0 => The corrosion type is not used.
  - 1 => The corrosion type is used.
- TIMES is the time value array.
- LIMIT is an array of maximum corrosion values (depth in most cases), and is indexed by corrosion type code (see CORROSN above).
- MECHF is the code for the cause of containment failure:
- 4 => Exponential distribution
  - 3 => Normal distribution
  - 2 => Uniform distribution
  - 1 => Point distribution
  - 0 => No failure

- 1 => Uniform corrosion
- 2 => Stress corrosion
- 3 => Pitting corrosion
- 4 => Fracture corrosion
- 5 => Cladding corrosion
- 6 => Prefailure

**TFAIL** is the time of containment failure (infinity if no failure has occurred).

**MECHFC** is the code for the cause of container failure. See MECHF above.

**TFAILC** is the time of container failure (infinity if no failure has occurred).

**STATUS** is the code for the current status of the container and containment. Constants have been defined in the STATUS.PAR include file, and are used in referencing status levels.

<u>Value</u>	<u>Constant</u>	<u>Meaning</u>
0	OK	The package is OK; neither container nor containment has failed.
1	NOCONT	The container wall has failed, but the cladding is intact.
2	NOCLAD	The cladding has failed, but the container wall is intact.
3	FAIL	Both container and cladding have failed; release will occur.

Only non-COMMON arguments are listed. Of these, MECHF, TFAIL, MECHFC, TFAILC, and STATUS are initialized before the call to MODEL and may be changed by MODEL. The other listed arguments are strictly inputs.

LOGIC

1. Get initial environment.  
(PHASE, ENVIR)
2. Set initial corrosion values.

(INITCUM)

3. For each time step,
  - a. Update environment.  
(PHASE, ENVIR)
  - b. If container failure has not occurred, then simulate nonuniform corrosion.  
(NONUNIF)
  - c. If container failure has occurred, then simulate cladding corrosion.  
(CLADDNG)
  - d. Simulate uniform corrosion.  
(UNIFORM)
  - e. If containment failed, then done modeling package; stop stepping through time.

Set Initial Corrosion Values

1. For each uniform or nonuniform corrosion type,
  - a. If used, then
    1. compute initial corrosion rate  
(DRIVECO)
    2. compute initial corrosion depth.

Simulate Cladding Corrosion

1. Compute cladding corrosion.  
(BREAKDN)
2. If failure, then store containment failure data.

Simulate Non-Uniform Corrosion

1. For each nonuniform corrosion type,
  - a. Compute corrosion.  
(BREAKDN)
  - b. If earliest nonuniform failure, then store container failure data.
2. If nonuniform failure occurred, then
  - a. Store containment failure data.

- b. Set initial cladding values.
- c. Simulate cladding corrosion.  
(CLADDNG)

### Simulate Uniform Corrosion

1. Compute uniform corrosion.  
(BREAKDN)
2. If earliest failure of container, then store container failure data.
3. If earliest failure of containment, then store containment failure data.

### Compute Corrosion

1. Compute corrosion rate.  
(DRIVECO)
2. Integrate rate curve over time step, and add to previous depth.  
(INTEGRA)
3. If depth exceeds limit, then interpolate failure time.  
(YINTERP)

### TEST PLAN

To test MODEL, several routines were written. Test drivers were used to call MODEL or its subroutines. Stub routines were written, replacing existing routines where expedient. For example, since MODEL does not care how these rates are computed (i.e. corrosion rate models are largely independent of the AREST corrosion framework), stub routines were used to generate corrosion rates.

The following outline indicates the test process:

1. Test the BREAKDN subroutine. Recall that BREAKDN drives the simulation of a corrosion type for one time step. Two cases were run:
  - a. Failure occurs precisely on an array time value.
  - b. Failure occurs between array time values, necessitating interpolation of the failure time.
2. Simulate typical MODEL runs. We are primarily concerned with three main paths MODEL can take. By choosing the stub routine

corrosion rates appropriately, MODEL was tested along each path:

- a. The container fails due to a nonuniform corrosion type. (For the tests, pitting corrosion was used.) Then containment fails due to cladding corrosion.
- b. The container fails due to nonuniform corrosion, then containment fails due to uniform corrosion.
- c. The container and containment both fail due to uniform corrosion.

**APPENDIX J**

**PITDPH MODULE**

## APPENDIX J

### PITDPTH MODULE

#### PURPOSE

PITDPTH computes the pitting depth which is sufficient to constitute container failure.

#### DESCRIPTION

With the exception of some conventions that have been adopted to preserve computational consistency with other corrosion modules, PITDPTH implements a pitting corrosion module that is essentially the same as that described in the AREST report(a). The major modification involves the computations that are used to determine the time of containment failure by pitting corrosion. In PITDPTH, it is also assumed that all potential pit sites are located on the exterior surface of the waste package container. This assumption is conservative because potential pitting sites may actually be situated within the container wall and would not be subject to pitting until exposed to the environment (by uniform corrosion, for example).

Containment failure by pitting is described in terms of  $k_f$ , which is the number of wall-penetrating pits determined to constitute failure. A waste package is determined to have failed by pitting corrosion at the time that the probability,  $p_f$ , of having at least  $k_f$  penetrations exceeds a conservatively small value (currently set at 0.1); this calculation involves the binomial distribution with parameters  $n$  (the number of potential pit sites) and  $p$  (the probability that a "representative" pit has penetrated to container). The probability  $p$  is estimated by

---

(a) Liebetrau, A. M., M. J. Apte, D. W. Engel, M. K. Altenhofen, D. M. Strachan, C. R. Reid, C. F. Windisch, R. L. Erikson, and K. I. Johnson. 1987. The Analytical Repository Source-Term (AREST) Model: Description and Document. PNL-6346, Pacific Northwest Laboratory, Richland, Washington.

$$p = \frac{\text{pit depth}}{\text{container thickness}}$$

In estimating  $p$ , it is implicitly assumed that all pits have the same probability of penetrating the container. Instead of doing the binomial calculation described above, a "failure depth"  $d_f$  is calculated by means of the following standard normal approximation:

$$z = \frac{y - \text{mean}}{\text{standard deviation}} = \frac{y - np}{\text{sqrt}(np(1 - p))}$$

where  $y$  is the number of pit failures and  $n$  is the number of pit sites. The approximation is concerned with the  $z$ -value obtained when  $y = k_f$ :  $p_f$ , the probability of failure by pitting corrosion is estimated as the probability that a standard normal value exceeds some conservatively small value of  $z$  (currently set at  $z = 1.2816$ , to correspond to the probability 0.1 that the number of penetrations exceeds  $k_f$ ). The values of  $z$ ,  $y$ , and  $n$  are known at the time of failure, so it is possible to solve for  $p$ :

$$p \leq k_f/n$$

The resulting value of  $p$  is used to compute the failure depth

$$d_f = \frac{k_f}{n} * \text{container thickness}$$

At each time step, the simulated depth of the representative pit is compared to  $d_f$  to determine whether or not containment failure by pitting has occurred.

#### ROUTINES CALLED

POISSON returns a Poisson-distributed random deviate (used for the number of pitting sites).

### ARGUMENT LIST

BADPITS is the number of pits sufficient for failure.

SITES is the number of potential pitting sites.

CTHICK is the thickness (cm) of the container.

PDEPTH. returns the pitting failure depth.

### LOGIC

The number of pitting sites is determined, and a simple computation is performed as described above.

### TEST PLAN

No testing was deemed necessary since the routine is trivial with the exception of the function call to POISSON which has undergone a number of test runs.

**APPENDIX K**

**CTINV MODULE**

## APPENDIX K

### CTINV MODULE

#### PURPOSE

This module calculates and stores the time dependent inventory for a given radionuclide.

#### DESCRIPTION

For each nuclide, subroutine CTINV calls the subroutine RPARAM to retrieve the parameters for the nuclide. RPARAM is fully documented in Appendix L. The subroutine CTINV then calculates the radioactive decay (Function DECAY). This version of the AREST code does not calculate decay or growin of the waste form, this feature will be added later. The growin of radium, however, is modeled in the CTINV routine.

The DECAY function calculates the radioactive decay scalar (the factor to multiply the 1000 year inventory by to get the inventory at any time). The inventory that is input to the AREST code is the inventory at 1000 years after emplacement. Therefore, if the time of interest (TIME) is less than 1000 years, DECAY is greater than one. Similarly, if the time of interest (TIME) is greater than 1000 years, then DECAY is less than one.

#### ROUTINES CALLED

RPARAM is called to retrieve the half life and the 1000 year inventory for a given nuclide. DECAY calculates the radioactive decay scalar.

#### ARGUMENT LIST

ITS is the number of time steps in the release calculations.  
NRN is the number for each nuclide being calculated.  
NRAD is the total number of nuclides.  
IFLAG if IFLAG = 1 then calculate growin (Ra-226).

TFA is the time of container failure after resaturation or wetting.

### LOGIC

1. For each nuclide:
  - a. Set parameters for individual nuclide.  
(RPARAM)
  - b. For each time step:
    1. Calculate radioactivity decay (DECAY).
    2. For Radium, calculate growin inventory.
    3. Scale inventory by radioactive decay.

### TEST PLAN

The testing of subroutines CTINV and DECAY included checking the following cases by hand calculation:

1. Test with TIME > 1000 years.
2. Test with TIME < 1000 years.

APPENDIX L

RPARAM MODULE

## APPENDIX L

### RPARAM MODULE

#### PURPOSE

This module retrieves, calculates, or simulates the parameters used to calculate the release rate.

#### DESCRIPTION

Subroutine RPARAM assigns values to the variables to be used in the calculations for the release of individual nuclides. The code retrieves the diffusion coefficients, and if necessary calculates shared solubilities. RPARAM then calculates the decay constant. If the gap or crud release models are being used, RPARAM calculates the concentration in the gap or crud. The code then checks for simulation of release parameters, porosities and sorption in this version. If a uniform distribution is being used to simulate variables, function RANDOM is called. If a normal distribution is being used, function NORMAL is called.

#### ROUTINES CALLED

This module calls subroutines NORMAL and RANDOM to simulate a normal or a uniform random variable. These routines are described in other appendices. This module also calls Function DECAY, which calculates the radioactive decay of the inventory at 1000 years. Function DECAY is described in Appendix K.

#### ARGUMENT LIST

- IR is the type of release model.
- If IR = 0, set parameters not specific to a particular release model.
- If IR = 1, Matrix release model.
- If IR = 2, Gap release model.
- If IR = 3, Grain release model.
- If IR = 4, Cladding Crud release model.

If IR = 5, Cladding Zircaloy release model.  
ICSAT is the radionuclide index.  
XINVEN is the 1000 year inventory of the nuclide in the waste form  
(g/package).  
TFAIL is the time of container failure.  
IPHASE is the type of environment.  
If IPHASE = 0, steam environment.  
If IPHASE = 1, water environment.  
IT is the index for the time dependent inventory.

### LOGIC

1. Get time or temperature dependent parameters.
  - a. Diffusion coefficients, use DFS in steam or DFW if in water environment.
  - b. If using shared solubility, then calculate shared solubilities by ratio of time dependent inventories.
2. Retrieve parameters for specific nuclide (half life, inventory, solubility, specific activity, diffusion coefficients, and Kd's).
3. Calculate decay constant ( $\lambda$ ).
4. If IR = gap (2) or cladding crud (4) then
  - a. Calculate concentration in gap or crud (CNOG).
  - b. Correct initial concentration in gap or crud by its radioactive decay at time of container failure (TFAIL).
5. Check for simulation of release parameters.
  - a. If uniform distribution then simulate uniform random deviate [0,1), scale to (min,max) (RANDOM).
  - b. If normal distribution, call NORMAL.
  - c. Set simulated parameters for:
    1. Backfill porosity.
    2. Rock porosity.
    3. Backfill retardation coefficient.
    4. Rock retardation coefficient.

## TEST PLAN

Module RPARAM retrieves, calculates, or simulates parameters used to calculate the release rate. The testing of RPARAM included the following cases by looking at the values assigned and comparing them to hand calculations where needed:

1. Test the case of shared nuclide solubilities (say for Cs-135 and Cs-137). This is done by testing two times after container failure (100 and 1000 years).
2. Test the case of  $IR = 2$ , the gap release model. The concentration in the gap and the radioactive scalar is calculated.
3. Test simulated values when using a uniform distribution.
4. Test simulated values when using a normal distribution.

**APPENDIX M**

**SOLUB MODULE**

## APPENDIX M

### SOLUB MODULE

#### PURPOSE

The SOLUB module calculates the UO<sub>2</sub> solubility. It is the driver subroutine that calls GRDH20, which calculates the groundwater composition for the simulated temperature, and UO2SOL, which calculates the UO<sub>2</sub> solubility for a given groundwater composition.

#### DESCRIPTION

The SOLUB module begins by calling GRDH20, which calculates the groundwater composition for the simulated temperature TEMP. This subroutine is fully described in Appendix E.

The SOLUB module then calculates a UO<sub>2</sub> solubility by calling the subroutine UO2SOL. Subroutine UO2SOL uses the UO<sub>2</sub> solubility function(a) to calculate the UO<sub>2</sub> solubility. This function is good only for reducing conditions, up to the UO<sub>2</sub>/U<sub>4</sub>O<sub>9</sub> boundary. The temperature dependent variables, constants A and B, and the equilibrium constants, have been hardcoded. For a given temperature (TEMP from the common block TIMETEMP) the A and B constants are calculated using linear interpolation of the tabled values, while the equilibrium constants are calculated using linear interpolation of the logarithms of the tabled values. No extrapolation of the tabled values is done.

The UO<sub>2</sub> solubility is calculated in moles per litre and converted to grams per meter cubed. The UO<sub>2</sub> solubility function uses the variables temperature (degrees C), pH, Eh (volts), carbonate (m/kg), phosphate (m/kg), sulphate (m/kg), and chloride (m/kg). The groundwater composition variables are stored in the common block WATER. Each variable has an allowable range for which

---

(a) Garisto, F., N. C. Garisto. 1986. A UO<sub>2</sub> Solubility function for the Assessment of Used Nuclear Fuel Disposal. AECL-8515, Atomic Energy of Canada, Limited.

the UO<sub>2</sub> solubility function can be used. If a range is violated, then the variable is set equal to its limit. The limits on the variables are

$$\begin{aligned}5 < \text{pH} < 10 \\ \text{Eh/pH} < \text{Eh} < \text{UO}_2/\text{U}_4\text{O}_9 \\ \text{CO}_3^{--} < 0.01 \\ \text{PO}_4^{---} < 0.000002 \\ \text{SO}_4^{--} < 0.01 \\ \text{Cl}^- < 1.0,\end{aligned}$$

where

$$\begin{aligned}\text{Eh/pH} &= -A * \text{pH} \text{ and} \\ \text{UO}_2/\text{U}_4\text{O}_9 &= B - A * \text{pH}.\end{aligned}$$

#### ARGUMENT LIST

- IWP is the number of the waste package being simulated.
- IFLAG is the flag for outputting the calculated UO<sub>2</sub> solubility. The solubility is output (to unit 60) if IFLAG equals 1.
- TEMP is the simulated temperature at which the UO<sub>2</sub> solubility calculated.
- UO2S is the calculated UO<sub>2</sub> solubility.

#### LOGIC

##### SOLUB

1. Find groundwater composition for temperature TEMP (Subroutine GRDH20).
2. Output UO<sub>2</sub> solubilities.
3. Calculate UO<sub>2</sub> solubility (Subroutine UO2SOL).
4. Output UO<sub>2</sub> solubility.

##### UO2SOL

1. Find temperature index (INDT) and multiplier (SLOPE) for log<sub>10</sub> linear interpolation.
  - a. If temperature is smaller than any tabled value, then use smallest value.
  - b. Find slope in original and log<sub>10</sub> scale.

2. If temperature is larger than any tabled value, then use largest value.
3. Check if variables exceed their limits.
4. Calculate range for Eh. Use linear interpolation to get a and b constants.
5. Output variables if desired (if SOLUB = -1.0 on input to routine).
6. Calculate Eh/a.
7. Calculate the 13 individual contributions to the UO<sub>2</sub> solubility. Use linear interpolation on the logarithm to get equilibrium constants.
8. Sum all individual contributions to get total UO<sub>2</sub> solubility in mole/l.
9. Convert total UO<sub>2</sub> solubility from mole/l to g/m\*\*3.

#### TEST PLAN

The testing of UO<sub>2</sub>SOL included checking the following steps by hand calculations:

1. Find the temperature index (INDT), slope, and log<sub>10</sub> slope for log<sub>10</sub> linear interpolation.
  - a. If the current temperature (TEMP) is less than the smallest value in the table, array SOLTEMP, use the smallest value. The slope and log<sub>10</sub> slope will be equal to 0.
  - b. If TEMP is equal to one of the tabled values (25C, 60C, 100C, 150C, 200C), then use the tabled data.
  - c. If TEMP is between the tabled values in SOLTEMP, then the slope and log<sub>10</sub> slope are calculated as follows:
    - TEMP = the current temperature.
    - TEMI = the first value in SOLTEMP that is greater than TEMP.
    - TEMIP = the temperature in SOLTEMP that precedes TEMI.
    - DIFF = TEMI-TEMP.
    - SLOPE = DIFF/(TEMI-TEMIP)
    - LOG<sub>10</sub> SLOPE = LOG<sub>10</sub>(TEMI-TEMP)/LOG<sub>10</sub>(TEMI/TEMIP)
  - d. If the current temperature is greater than the tabled values, then use the largest value in the table (SOLTEMP). The slope and log<sub>10</sub> slope will be set equal 0.
2. Check to see if the pH, Eh, CO<sub>3</sub>, PO<sub>4</sub>, SO<sub>4</sub>, and Cl variables have exceeded their limits. The A and B constants will be calculated using linear interpolation, and then will be used to calculate the range for Eh.

3. Calculate the 13 individual contributions to the  $UO_2$  solubility.
4. Sum the individual solutions,  $K(1)$  through  $K(13)$ , to get the total  $UO_2$  solubility in mole/l.
5. Convert the total  $UO_2$  solubility from mole/l to  $g/m^3$ .

**APPENDIX N**

**RMATRIX MODULE**

## APPENDIX N

### RMATRIX MODULE

#### PURPOSE

The RMATRIX module calculates the mass transfer and fractional release rates for the solubility-limited nuclides. Either a congruent or an incongruent boundary condition at the waste form surface may be used.

#### DESCRIPTION

RMATRIX is called by AREST separately for each nuclide of each simulated waste package. It calls several routines for calculating and storing the mass transfer and fractional release rate. If the parameter MRUN is set to one by AREST, RMATRIX will not calculate the mass transfer rate for this nuclide-waste package combination, instead it will use the rates that were calculated for the first waste package to calculate the fractional release rate. This option is used to reduce calculations and may be used if the parameters of the release equations do not change with time or temperature.

RMATRIX first calculates the mass transfer rate using the incongruent boundary condition at the waste form surface. This mass transfer rate is calculated only once, for the first waste package, for each nuclide if MRUN is 0 and is denoted as  $M(i,t,r)$  where  $i$  is the individual nuclide,  $t$  is the time since containment failure, and  $r$  is the distance where the release rate is to be calculated (RDIST). The distance  $r$  must be between the waste form surface ( $R_0$ ), and the waste package host rock interface ( $R_1$ ).

Next, RMATRIX calculates the congruent mass transfer rate from the waste form surface. The equation for calculating this mass transfer rate is:

$$M(i,t,R_0,\text{cong}) = M(\text{UO}_2,t,R_0) \cdot \frac{I(i,t')}{\dot{i}(\text{UO}_2,t')} \quad (1)$$

where  $I(i,t')$  is the time-dependent inventory for nuclide  $i$  at time since repository closure ( $t'$ ). Then the congruent mass transfer rate away from the

waste form surface, at distance  $r > R_0$ , is calculated. This release rate is calculated using the incongruent mass transfer rate of nuclide  $i$  and the mass transfer rate from Equation (1):

$$M(i,t,r,\text{cong}) = M(i,t,R_0,\text{cong}) \cdot \frac{M(i,t,r)}{M(i,t,R_0)} \quad (2)$$

Finally RMATRIX calculates the fractional release rate. The fractional release rate,  $F(i,t,r)$  for incongruent release and  $F(i,t,r,\text{cong})$  for congruent release, is calculated by dividing the mass transfer rate by the 1000 year inventory of the individual nuclide  $i$ .

#### ROUTINES CALLED

SINCON calculates the incongruent mass transfer rate for a given nuclide at any distance.

SUO2R0 calculates the mass transfer rate of the UO<sub>2</sub> matrix from the waste form surface.

SUO2R1 calculates the mass transfer rate of the UO<sub>2</sub> matrix in the packing material or into the host rock.

SCONG1 calculates the ratio of the incongruent release rate in the packing material, or into the host rock, with the incongruent release rate at the waste form surface.

INCON calculates the incongruent fractional release rate.

CONGRO calculates the congruent fractional release rate from the waste form surface.

CONGR1 calculates the congruent fractional release rate in the packing material or into the host rock.

#### ARGUMENT LIST

MRUN	is a flag for calculating mass transfer rates, 0 --> do not calculate, use store array (Vector Approach) 1 --> calculate mass transfer rates.
NRN	is the index for the nuclide being used.
ITS	is the number of time steps to calculate release.

**ICSAT** is the flag for shared solubility nuclides,  
 0 --> not shared solubility.

**ISOL** is the solubility limit flag,  
 0 --> No solubility limit applied  
 -1 --> Solubility limit applied and also calculating  
 matrix release.  
 -2 --> Solubility limit applied and not calculating  
 matrix release, thus only calculate incongruent  
 release.

**IFCONG** is the Flag for the type of boundary condition used,  
 0 --> Incongruent  
 1 --> Congruent.

**IPPT** is the Flag for using precipitation,  
 > 0 --> precipitation.

**TFA** is the time of containment failure.

**XINVEN** is the 1000 year inventory.

**UO2S** is the solubility of UO<sub>2</sub>.

**TPREC** is the time of precipitation.

**XPREC** is the distance (radial) of precipitation (% of packing).

**CPREC** is the concentration for precipitation (% of Csat).

**STOREL1** is an array for storing time dependent fractional release  
 rates (1/yr).

**STOREL2** is an array for storing time dependent release rates  
 (cm\*\*3/s).

### LOGIC

#### 1. Calculate mass transfer rates.

- a. Incongruent from waste form:  $M(i,t,R_0)$   
 (SINCON)
- b. Incongruent into host rock:  $M(i,t,r)$   
 (SINCON)
- c. Incongruent UO<sub>2</sub> from waste form:  $M(UO_2,t,R_0)$   
 (SUO2R0)
- d. Incongruent UO<sub>2</sub> into host rock:  $M(UO_2,t,r)$   
 (SUO2R1)

- e. Ratio of incongruent releases:  $M(i,t,r)/M(i,t,R_0)$
2. Calculate fractional release rates.
- a. Incongruent from waste form  
(INCON)
  - b. Incongruent into host rock  
(INCON)
  - c. Congruent from waste form  
(CONGRO)
  - d. Congruent into host rock  
(CONGR1)

### TEST PLAN

The test plan for the matrix release model (analytical solution) is outlined below. In AREST we assume that the  $UO_2$  matrix consists entirely of U-238,  $M(i,t,r)$  is the incongruent time-dependent mass transfer rate, and the fractional release rate is  $F(i,t,r) = M(i,t,r)/I(i,1000)$ .

1. First verify that the actual release (incongruent: using the individual nuclides parameters) calculations are correct. This was done by matching the results from the AREST code to published results(a). The test run was for both release from the waste form ( $r = R_0$ ) and release into the host rock ( $r = R_1$ ).  $M(i,t,r)/Csat$
2. Calculate incongruent release for U-238 and C-14 into the host rock ( $r = R_1$ ).  $F(i,t,R_1)$
3. Calculate incongruent release for U-238 and C-14 from the waste form ( $r = R_0$ ).  $F(i,t,R_0)$
4. Calculate congruent release from the waste form.  $F(i,t,R_0,cong)$ , where

$$\begin{aligned} M(UO_2,t,R_0,cong) &= M(UO_2,t,R_0), \text{ so} \\ F(UO_2,t,R_0,cong) &= F(UO_2,t,R_0), \text{ and} \\ (4) &= (3) \end{aligned}$$

$$M(i,t,R_0,cong) = M(UO_2,t,R_0) * I(i,t')/I(UO_2,t'), \text{ or}$$

---

(a) Chambre', P. L., T. H. Pigford, W. W. Lee, J. Ahn, S. Kajiwara, C. L. Kim, H. Kimura, H. Lung, W. J. Williams, and S. J. Zavoisky. 1985. Mass Transfer and Transport in a Geologic Environment. LBL-19430, Lawrence Berkeley Laboratory, Berkeley, California.

$$F(i,t,R_0,cong) = M(UO_2,t,R_0) * I(i,t')/I(UO_2,t')/I(i,1000).$$

Using the approximation  $I(UO_2,1000) = I(UO_2,t')$ , we get

$$F(i,t,R_0,cong) = F(UO_2,t,R_0) * I(i,t')/I(i,1000).$$

- a. For C-14 calculate the ratio of the time dependent inventory to the 1000 year inventory.  $I(i,t')/I(i,1000)$
- b. Then calculate release

$$F(i,t,R_0,cong) = F(UO_2,t,R_0) * I(i,t')/I(i,1000)$$

$$(4) \quad = \quad (3) \quad * \quad (4a)$$

5. Finally, calculate congruent release into the host rock and check by hand calculation.

$F(i,t,R_1,cong)$ , where

$$F(UO_2,t,R_1,cong) = F(UO_2,t,R_1), \text{ and}$$

$$(5) \quad = \quad (2)$$

$$M(i,t,R_1,cong) = M(i,t,R_1) * M(i,t,R_0,cong)/M(i,t,R_0), \text{ so}$$

$$F(i,t,R_1,cong) = F(i,t,R_1) * F(i,t,R_0,cong)/F(i,t,R_0), \text{ so}$$

$$(5) \quad = \quad (2) \quad (4) \quad (3)$$

**APPENDIX 0**

**RGAP MODULE**

## APPENDIX 0

### RGAP MODULE

#### PURPOSE

The RGAP module calculates the mass transfer and fractional release rates for the soluble inventory-limited nuclides, from the gap/grain boundary.

#### DESCRIPTION

RGAP calculates and stores the mass transfer and fractional release rate separately for each nuclide of each simulated waste package. The flag MRUN is set to one if the mass transfer rate is to be calculated, and is set to zero if the mass transfer rate for the first waste package is used for calculating the fractional release rate. This option is used to reduce calculations and may be used if the parameters of the release equations do not change with time or temperature.

#### ROUTINES CALLED

The subroutine RELEASE is called to select the release model to be used to calculate the mass transfer rate.

#### ARGUMENT LIST

MRUN	is a flag for calculating mass transfer rates, 0 --> do not calculate, use store array (Vector Approach) 1 --> calculate mass transfer rates.
NRN	is the index for the nuclide being used.
ITS	is the number of time steps to calculate release.
IPHASE	is the flag for the environmental conditions, 0 --> steam 1 --> water.
TFA	is the time of containment failure.
XINVEN	is the 1000 year inventory.
STOREL	is an array for storing time-dependent release rates (1/yr).

## LOGIC

1. For each time step calculate mass transfer rate:  $M(i,t,r)$ 
  - a. Calculate time since closure.
  - b. Calculate release (g/yr).  
(RELEASE)
  - c. Normalize release by concentration ( $m^{**3}/yr$ ).
2. For each time step calculate fractional release rate:  $F(i;t,r)$ 
  - a. Retrieve release rate ( $m^{**3}/yr$ ).
  - b. Multiply release by concentration (g/yr).
  - c. Normalize release by 1000 year inventory (1/yr).

## TEST PLAN

For testing the gap model, the results from the AREST code were compared to results that have been published(a).

---

(a) Kim, C. L., P. L. Chambre', T. H. Pigford. 1986. Mass Transfer-Limited Release of a Soluble Waste Species. LBL-20899, Lawrence Berkeley Laboratory, Berkeley, California.

APPENDIX P

RCRUD MODULE

## APPENDIX P

### RCRUD MODULE

#### PURPOSE

The RCRUD module calculates the mass transfer and fractional release rates from the crud part of the cladding. This release is for the highly soluble inventory-limited nuclides. The release from the crud can occur in either the steam or water environment.

#### DESCRIPTION

RCRUD calculates and stores the mass transfer and fractional release rate for C-14 for each simulated waste package. The release can be calculated in either the steam or the water environment.

The flag MRUN is set to one if the mass transfer rate is to be calculated, and it is set to zero if the mass transfer rate for the first waste package is used for calculating the fractional release rate. This option is used to reduce calculations and may be used if the parameters of the release equations do not change with time or temperature.

#### ROUTINES CALLED

RPARAM is called to return time and environmental (water or steam) parameters.

The subroutine RELEASE is called to select the release model to be used and calculate the mass transfer rate.

#### ARGUMENT LIST

MRUN	is a flag for calculating mass transfer rates, 0 --> do not calculate, use store array (Vector Approach) 1 --> calculate mass transfer rates.
NRN	is the index for the nuclide being used.
ITS	is the number of time steps to calculate release.
IPHASE	is the flag for the environmental conditions,

0 --> steam  
1 --> water.

TFAC is the time of container failure.  
TFACR is the time of container failure after the resaturation of the waste package.  
XINVEN is the 1000 year inventory.  
STOREL is an array for storing time dependent release rates (1/yr).

### LOGIC

1. Calculate mass transfer rate for steam and water environment.
  - a. Set release parameters for environment (steam or water).  
(RPARAM)
  - b. For each time step calculate release:  $M(i,t,r)$ 
    1. Calculate time since closure.
    2. Calculate release (g/yr).  
(RELEASE)
    3. Normalize release by concentration ( $m^{*3}/yr$ ).
2. For each time step calculate fractional release rate:  $F(i,t,r)$ 
  - a. Retrieve release rate ( $m^{*3}/yr$ ).
  - b. Multiply release by concentration (g/yr).
  - c. Normalize release by 1000 year inventory (1/yr).

### TEST PLAN

The testing for this module was done by comparing the results for the gap release with those for the calculation of the crud release, since the same models are used by the AREST code, for the two sources. By setting the parameters and distribution in each model of C-14, the results for the release from the gap and those from the crud are identical.

**APPENDIX Q**

**RCLAD MODULE**

## APPENDIX Q

### RCLAD MODULE

#### PURPOSE

The RCLAD module calculates the mass transfer and fractional release rates from the zircaloy cladding. A congruent matrix (Zr) release model, solubility-limited, is used to calculate the release of C-14 from the cladding.

#### DESCRIPTION

RCLAD calculates and stores the mass transfer and fractional release rate for C-14 from the zircaloy cladding each simulated waste package. The release from the cladding is similar to that of the congruent  $UO_2$  matrix release. The same equations and steps that were used in the RMATRIX module are used in the RCLAD module.

The first step is to calculate the mass transfer rate of Zr at the waste form surface. Then the mass transfer rate of C-14, at the waste form surface, is calculated by multiplying the release of Zr, at the waste form surface, by the ratio of the time dependent inventories, Equation 1 of Appendix N. Next the incongruent release of C-14 at the distance  $r$  (RDIST) in the packing material or into the host rock is calculated. Then the mass transfer rate of C-14 is calculated using Equation (2) of Appendix N, and finally the fractional release rate is calculated.

The flag MRUN is set to one if the mass transfer rate is to be calculated, and it is set to zero if the mass transfer rate for the first waste package is used for calculating the fractional release rate. This option is used to reduce calculations and may be used if the parameters of the release equations do not change with time or temperature.

## ROUTINES CALLED

RPARAM is called to return time-dependent parameters.

The subroutine RELEASE is called to select the release model to be used and to calculate the mass transfer rate.

## ARGUMENT LIST

MRUN is a flag for calculating mass transfer rates,  
0 --> do not calculate, use store array (Vector Approach)  
1 --> calculate mass transfer rates.

NRN is the index for the nuclide being used.

ITS is the number of time steps to calculate release.

TFA is the time of containment failure.  
the waste package.

XINVEN is the 1000 year inventory.

STOREL is an array for storing time dependent release rates  
(1/yr).

## LOGIC

1. Calculate mass transfer rate for Zr from the cladding:  $M(\text{Zr}, t, R_0)$ 
  - a. Set release parameters for zircaloy.  
(RPARAM)
  - b. For each time step calculate release.
    1. Calculate time since closure.
    2. Calculate release for zircaloy from the cladding (g/yr).  
(RELEASE)
    3. Calculate time dependent inventories for C-14 and Zr.
    4. multiply zircaloy release by ratio of inventories.
3. Set release parameters for C-14.  
(RPARAM)
4. Calculate mass transfer rate for Zr from the cladding:  $M(\text{Zr}, t, R_0)$ 
  - a. For each time step calculate release.
    1. Calculate time since closure.
    2. Calculate C-14 release from the cladding (g/yr).  
(RELEASE)

5. Calculate mass transfer rate for C-14 into host rock:  $M(C-14,t,r)$ 
  - a. For each time step calculate release.
    1. Calculate time since closure.
    2. Calculate C-14 release from the cladding (g/yr).  
(RELEASE)
6. Calculate fractional release rate for C-14 into the host rock.
  - a. For each time step calculate mass transfer rate:  $M(C-14,t,r,cong)$ 
    1. Retrieve mass transfer rates.
      - a. C-14 congruent release from the cladding.
      - b. C-14 incongruent release from the cladding.
      - c. C-14 release into host rock.
    2. Multiply congruent release of C-14 by the ratio of the incongruent releases (g/yr).
  - b. Normalize release by 1000 year inventory (1/yr).

#### TEST PLAN

Since the same release models are used for the release from the cladding and congruent release from the  $UO_2$  matrix, the test plan for the module RCLAD is simply to compare the release of C-14 from the  $UO_2$  matrix with a similar release of C-14 from the cladding. Thus, the parameters for Zr were set equal to those for U-238 and the distribution of C-14 in the  $UO_2$  matrix and the cladding were set equal. The results indicated that the release rates for the cladding and the  $UO_2$  matrix were identical.

**APPENDIX R**

**RSUM MODULE**

## APPENDIX R

### RSUM MODULE

#### PURPOSE

AREST simulates containment failure for each waste package. Release rates are then calculated and stored in individual vectors (one for each release model nuclide combination). Finally these individual vectors are summed to and stored in the array RCUMUL. This final step, summing and storing release rates, is done in the RSUM module. RSUM is called from the ESR component to sum release rates for all waste packages.

#### DESCRIPTION

RSUM is called once for each nuclide, release model, waste package combination. The routine stores the total repository release rates in the array RCUMUL and the time since repository closure, in the array TCUMUL.

The individual release rates are calculated from the time of containment failure (Tfail). Originally the RCUMUL array is set to zero. Any call to the RSUM routine results in summing the individual release rates (STREL) to the proper column of the RCUMUL array. Both the time since repository closure vector (TCUMUL) and the time since waste package failure vector (STIME) are initialized in the RTIME subroutine (see Appendix T).

Table R.1 is provided to illustrate what happens in RSUM when distributed failure times occur. All of these columns are vectors used in RSUM. TCUMUL is the time since repository closure, STIME is the time since containment failure, STREL is the release rates calculated for the first waste package, the Vector Approach is being used. The next column, TFAIL = 0.0, lists the release rates for a waste package failing at 0.0 years after closure of repository. The column with the header TFAIL = 2.5 lists the release rates for a waste package failing at 2.5 years after repository closure, and the next column lists the release rates for a waste package that failed 5.0 years after repository closure. The last column shows the results of summing the release rates over all of the waste packages.

TABLE R.1. Example of Different Failure Times

<u>TCUMUL</u>	<u>STIME</u>	<u>STREL</u>	<u>TFAIL=0.0</u>	<u>TFAIL=2.5</u>	<u>TFAIL=5.0</u>	<u>RCUMUL</u>
1.0	1.0	1.0	1.0	0.0	0.0	1.0
2.0	2.0	2.0	2.0	0.0	0.0	2.0
3.0	3.0	3.0	3.0	0.5	0.0	3.5
4.0	4.0	4.0	4.0	1.5	0.0	5.5
5.0	5.0	5.0	5.0	2.5	0.0	8.0
6.0	6.0	6.0	6.0	3.5	1.0	10.5
7.0	7.0	7.0	7.0	4.5	2.0	13.5
8.0	8.0	8.0	8.0	5.5	3.0	15.6
9.0	9.0	9.0	9.0	6.5	4.0	19.5
10.0	10.0	10.0	10.0	7.5	5.0	22.5

ROUTINES CALLED

RSUM calls the function AREA to calculate area under the individual release rate curve, between two time steps. The area is calculated for each time interval of the TCUMUL array.

ARGUMENT LIST

NRN is the index of the nuclide being stored.  
NTCUM is the number of release, time combinations being stored.  
IR is the index of the release model that is being stored.  
ITS is the number of time steps, release rate combinations.  
IWP is the number for waste package that is being simulated.  
IEXH is the index in the STIME array for when exhaustion of nuclide occurs.  
TFAILUR is the time of container failure.  
NPERC NPERC > 0 --> store individual release to calculate percentiles later.  
STREL is the array of individual release rates (1/yr).  
STIME is the array of times (yr) where release is calculated.

LOGIC

1. Find first time step on the stored time array that is greater than the time of failure.
2. Get release rates for first time interval.
  - a. Calculate time since failure.

- b. Calculate area under individual release curve for first time interval.  
(CAREA)
- c. Calculate release at end points of first time interval.
3. Sum release to other waste packages for first time interval.
4. Check for exhaustion of nuclide (time step greater than IEXH).
5. For remaining time steps, sum release to previous waste packages.
  - a. Calculate time since failure.
  - b. Calculate area under individual release curve for new time interval (AREA)
  - c. Calculate release at time step.
  - d. Sum release to other waste packages.
  - e. Check for exhaustion of nuclide.

#### TEST PLAN

The testing of the RSUM submodule was done in the four steps described below. These results were checked by hand calculations.

1. Make sure that the location of the individual release vector is correct. In other words, locate the time of containment failure with respect to the TCUMUL vector. This is done by observing when the release rates start compared to the time of containment failure (1 year and 1000 years were used).
2. Test the routine that calculates the area under the individual release vector (AREA) from T0 to T1.
  - a. Case where the endpoints (T0 and T1) fall directly on the time steps.
  - b. Case where interpolation is needed.
3. Test the estimation of release rates for the RCUMUL array.
4. Test the summing for two waste packages.

APPENDIX S

CUMREL MODULE

## APPENDIX S

### CUMREL MODULE

#### PURPOSE

The CUMREL module calculates the total release rate for each nuclide and waste package by summing the rates for all release models (matrix, gap/grain boundary, clud, and clad), to check for the exhaustion of the nuclide.

#### DESCRIPTION

The subroutine CUMREL adds the release rate from one release model to another (i.e., CUM = matrix + gap + cladding). It also calculates the amount released (parts) to see if the nuclide has been exhausted for the given model. Exhaustion occurs when the area under the fractional release rate curve (1/yr) for the given model equals the fraction in that release model (e.g., 1.0 for 100% in matrix). The release rate is set to 0 for all time steps after the exhaustion of the nuclide.

#### ROUTINES CALLED

Function CUMUL uses the trapezoidal rule to numerically integrate the area under the release rate curve. CUMUL integrates from the calculated X value to the previous X value, and from the calculated Y value to the previous Y value.

#### ARGUMENT LIST

ITS is the number of values in each array to sum.  
IEXH is the index for the time step at which exhaustion occurs,  
CUM(iexh) = 0.  
REL is the array of release rates (1/yr).  
CUM is the array of cumulative release (across the models).  
TIME is the time array (years) used for calculating the area under the release rate curve.

PERR is the fraction of the nuclide in the release model (e.g., 1.0 if 100% in matrix).

#### LOGIC

1. Initialize variables.
2. For each time step cumulate release.
  - a. Check to see if exhaustion has occurred. If it has not occurred, do the following.
    1. Calculate the area under the release curve.  
(CUMUL)
    2. Check for exhaustion of the nuclide. If it has occurred, do the following.
      - a. Use the trapezoidal rule to make the area under the release curve equal 1.0.
      - b. Store the release at the time of exhaustion.
      - c. Set the index in the time array where exhaustion occurred.
    3. Sum the release across the models (1/yr).
  - b. Store the release for the next release model.

#### TEST PLAN

The testing of CUMREL included checking by hand calculations the following steps:

1. Test for exhaustion of Cm-245. Run the code with Time of failure at 1000 years, and use incongruent release.
2. Test that the array containing the accumulated release rates (CUM) is being filled properly. Run the code with U-238 and C-14 with Time of failure at 1000 years, and use congruent release.

APPENDIX T

USER-SUPPLIED ROUTINES

## APPENDIX T

### USER-SUPPLIED ROUTINES

This appendix is intended to describe the corrosion and release models that are present in the AREST code, to discuss the procedure someone would follow to add their own models, and to show how to modify existing routines in the AREST code. Also, the common blocks used by the AREST code are briefly discussed, and non-standard features that have been used in the AREST code are described.

### CURRENT CORROSION MODELS

The AREST code provides a number of low-level subroutines which perform rate calculations for various corrosion models. The code also provides a mechanism for implementing user-defined models. These models are identified numerically within the program by the values listed in Table A.1 (Appendix A) under the "Corrosion Models" heading.

Currently, the AREST code provides the following pre-programmed corrosion-rate subroutines: BROOK, BWIP, BWIPS, DTPUN, DTPPIT, PNLGIT, PNLSTE, PNLSTZ, PNLUNI, PNLUNZ, and WESTER. In addition, the code includes dummy user-definable routines: CUSER1, CUSER2, CUSER3, CUSER4, CUSER5, CUSER6, and CUSER7.

The pre-programmed corrosion routines are all time-dependent, and all return both corrosion rates and depths. BROOK implements a uniform corrosion model for low-carbon steel in salt water. There are two Basalt Waste Isolation Project models for uniform corrosion (Sagar et al.): BWIP is used for an aqueous environment, and BWIPS is used for a steam environment. Two models are included for waste packages using carbon steel canisters with basalt-bentonite packing: DTPPIT computes pitting corrosion, and DTPUN computes uniform corrosion. Several pre-programmed routines are provided by the Pacific Northwest Laboratory (Liebetrau et al. 1987): PNLGIT for pitting of steel in an aqueous environment, PNLSTE for uniform corrosion of steel in steam, PNLSTZ for uniform corrosion of zircaloy in steam, PNLUNI for uniform corrosion of

steel in an aqueous environment, and PNLUNZ for uniform corrosion of zircaloy in an aqueous environment. Finally, WESTER models uniform corrosion of iron-based materials in an aqueous salt repository (Westerman et al. 1984).

User-defined models can be implemented using the CUSER routines. Each current CUSER routine is a placeholder which can be replaced by a user-written routine. The specific CUSER routine to replace depends on the environmental conditions and corrosion mode of interest: CUSER1 is used for uniform corrosion in a steam environment. CUSER2 is used for uniform corrosion in an aqueous environment. CUSER3 is used for stress corrosion in an aqueous environment. CUSER4 is used for pitting in an aqueous environment. CUSER5 is used for mechanical fracturing in an aqueous environment. CUSER6 is used for uniform corrosion of cladding in an aqueous environment. CUSER7 is used for uniform corrosion of cladding in a steam environment.

All corrosion routines require the same arguments: TIME, RATE and DEPTH, in that order. TIME is an input; corrosion RATE and DEPTH are outputs. At present, DEPTH (but not RATE) can be returned with a dummy value and rate is returned in centimeters per year. The routines may require other input values which need to be input by the ARESTIN module and passed by way of common blocks.

### CURRENT RELEASE MODELS

The current version of the AREST code contains release models developed for saturated media. Two models are available for solubility-limited release (release from the  $UO_2$  matrix and release from the cladding) and two models for inventory-limited release (release from the gap/grain boundaries and release from the crud).

One of the solubility-limited release models assumes a spherical waste package geometry, and is limited to one-member decay chains (Chambre' et al. 1985). These models also assume that mass transfer rates are diffusion limited. The solubility-limited models for  $UO_2$  matrix and cladding release are collectively denoted as the Pigford and Chambre' model in Table A.1 of Appendix A. Another release model that has been recently added to the AREST code calculates steady-state releases in a partially saturated media taking

into account rates of matrix dissolution, convection and diffusion (Reimus et al. 1988). It compares the release rates that result from these three processes and uses the appropriate limiting rate. This model is the PNL TUFF model in Table A.1 of Appendix A.

For unsaturated media, the PNL TUFF model, is also used to calculate release rates from the gap and crud, except that matrix dissolution is not considered as a limiting factor. For saturated media, another Pigford and Chambre' model (Kim, Chambre', and Pigford 1986) is used to calculate release from the gap and crud. This model assumes a planar geometry and diffusional release only.

The user also has the option of adding their own release models. There are five dummy routines that the user can modify and add to the AREST code. The routines (RUSER1, RUSER2, RUSER3, RUSER4, and RUSER5) correspond to a release model (matrix, gap, grain boundaries, crud, and cladding), where RUSER1 calculates the release from the matrix, RUSER2 calculates release from the gap, RUSER3 calculates release from the grain boundaries, RUSER4 calculates release from the crud, and RUSER5 calculates release from the cladding. The release from the grain boundaries is not used directly, in this version of the AREST code. Instead, the release from the grain boundaries is combined with the release from the gap.

To add new release models, simply modify the correct RUSER routine, compile and link the routine with the AREST code, and choose the user supplied option when running the code (option 0 for each release model from Table A.1 of Appendix A). The variables available to the release model are found in the common blocks. The release model returns release in grams per year.

#### MODIFIABLE ROUTINES

The AREST code has been designed to be easily modified. It is recommended that the modifications be made by a programmer, using this document the source code as a guide. The AREST code has been broken into several modules with most modules contained in a separate file. The rest of this section will be describing several modules that may need to be modified by the user.

The routine RTIME initializes the time vectors for the calculation of the release. Two time vectors are initialized by this routine. The first time vector, STRT, is used to calculate the release from an individual waste package and refers to the time after containment failure. The second time vector, TCUMUL, is used to calculate the total release from the repository, and refers to the time after the closure of the repository.

The individual radionuclide parameters are read into the AREST code using the RADION routine. This routine is called from the ARESTIN module. This routine reads in the parameters and stores them in arrays. The distribution of the nuclides in the release models (e.g., 50% in the matrix and 50% in the gap) are stored in the RPERC array. While the individual parameters for each nuclide are stored in the array RPARA.

The release parameters for the individual nuclides are set in the routine RPARAM. This routine retrieves the parameters from the arrays RPERC and RPARA, and also simulates different parameters. With this version of the AREST code the only parameters that may be simulated are the porosities and retardation coefficients for the packing and host rock.

INEVIR is the routine used to input the thermal data base. It is called by the ARESTIN module and sets the time/temperature distribution. This routine may need to be modified, depending on how the thermal data base is to be used. The algorithm used by the AREST code is described by Liebetrau et al. (1987).

The groundwater composition is input into AREST by the INGRDH20 routine. This routine will have to be modified if a new format is used for the groundwater. INGRDH20 is discussed in Appendix E (GRDH20 MODULE).

#### COMMON BLOCKS

Most of the variables and parameters used in the AREST code are stored in common blocks. There are several common blocks that are used by the AREST code. These common blocks are incorporated into each routine with the use of the include statement. This statement is not part of the FORTRAN 77 standard, but is used in most versions of FORTRAN 77. Each common block is contained in its own file and each parameter and variable contained in the common block is documented in the file.

Besides the use of the include statement, there are other features of the Fortran 77 standard that have been violated. They include non-standard length of variable and routine names and the use of the "!" as a comment delimiter. These non-standard features are easy to fix and are implemented in most compilers, thus we have not converted them to the equivalent standard.

#### REFERENCES

Chambre', P. L., T. H. Pigford, W. W. Lee, J. Ahn, S. Kajiwara, C. L. Kim, H. Kimura, H. Lung, W. J. Williams, and S. J. Zavosky. 1985. Mass Transfer and Transport in a Geologic Environment. LBL-19430, Lawrence Berkeley Laboratory, Berkeley, California.

Kim, C. L., P. L. Chambre', T. H. Pigford. 1986. Mass Transfer-Limited Release of a Soluble Waste Species. LBL-20899, Lawrence Berkeley Laboratory, Berkeley, California.

Liebetrau, A. M., M. J. Apted, D. W. Engel, M. K. Altenhofen, D. M. Strachan, C. R. Reid, C. F. Windisch, R. L. Erikson, and K. I. Johnson. 1987. The Analytical Repository Source-Term (AREST) Model: Description and Document. PNL-6346, Pacific Northwest Laboratory, Richland, Washington.

Reimus, P. W., A. M. Liebetrau, M. J. Apted, and D. W. Engel. 1988. Performance Assessment for Spent Fuel Waste Packages in Hydrologically Unsaturated Geologic Media. Spectrum '88, American Nuclear Society, Chicago, Illinois.

Sagar, B., P. W. Eslinger, R. G. Baca, R. P. Anantatmula. 1985. Probabilistic Modeling of Radionuclide Release at the Waste Package Subsystem Boundary of a Repository in Basalt. BWI-TA-012, Rockwell Hanford Operations, Richland, Washington.

Westerman, R. E., J. L. Nelson, S. G. Pitman, W. L. Kuhn, S. J. Basham, and D. P. Moak. 1984. Evaluation of Iron-Based Materials for Waste Package Containers in a Salt Repository, Materials Research Society Symposium Proc. Vol. 26.

DISTRIBUTION

No. of  
Copies

No. of  
Copies

OFFSITE

2	DOE/Office of Scientific and Technical Information	K. Chang U.S. Nuclear Regulatory Commission Division of Waste Management Mail Stop 623-SS Silver Springs, MD 20910
3	U.S. Department of Energy Office of Civilian Radioactive Waste Management, RW-24 Washington, DC 20545 D. H. Alexander S. Gomberg R. Stein	W. J. Conover College of Business Administration Texas Tech University Lubbock, TX 79409
	S. Russo, Acting Director U.S. Department of Energy Office of Civilian Radioactive Waste Management, RW-1 Washington, DC 20585	S. Coplan U.S. Nuclear Regulatory Commission Mail Stop 623-SS Silver Springs, MD 20910
2	U.S. Department of Energy Office of Civilian Radioactive Waste Management, RW-20 Forrestal Building Washington, DC 20585 T. H. Isaacs S. H. Kale	G.A. Dinwiddie U.S. Department of the Interior U.S. Geological Survey National Center, Mail Stop 410 12202 Sunrise Valley Drive Reston, VA 22092
3	U.S. Department of Energy Office of Civilian Radioactive Waste Management, RW-222 Forrestal Building Washington, DC 20585 J. F. Daly M. W. Frei J. L. Morris	Document Control Center U.S. Nuclear Regulatory Commission Division of Waste Management Washington, DC 20555
	C. L. Carnahan Lawrence Berkeley Laboratory Mail Stop 50E 1 Cyclotron Road Berkeley, CA 97420	W. E. Coons RE/SPEC Incorporated P.O. Box 725 Rapid City, SD 57709
		N. A. Eisenberg Office of NMSS Mail Stop 4-H-3 U.S. Nuclear Regulatory Commission Washington, DC 20555
		B. G. Gale U.S. Department of Energy Office of Civilian Radioactive Waste Management, RW-223 Forrestal Building Washington, DC 20585

No. of  
Copies

P. Gnirk  
RE/SPEC  
P.O. Box 14984  
Albuquerque, NM 87191

W. Harrison  
Argonne National Laboratory  
6700 South Cass Avenue  
Argonne, IL

G. Jacobs  
Oak Ridge National Laboratory  
P.O. Box X  
Bethel Bailey Road  
Mail Stop 039, Building 1505  
Oak Ridge, TN 37831

C. M. Jantzen  
E.I. du Pont de Nemours & Co.  
Savannah River Laboratory  
Aiken, SC 29808

H. J. Machiels  
Electric Power Research Institute  
P.O. Box 10412  
Palo Alto, CA 94303

T. J. Nicholson  
U.S. Office of Nuclear Regulatory  
Research  
Waste Management Branch  
5650 Nicholson Lane  
Rockville, MD 20582

U-Sun Park  
SAIC  
101 Convention Point Center Drive  
Las Vegas, NV 89109

C. Pescatore  
Brookhaven National Laboratory  
Upton, NY 11973

P. W. Reimus  
3977 B Nickel Street  
Los Alamos, NM 87544

No. of  
Copies

L. R. Rickertson  
Weston  
955 L'Enfant Plaza  
Washington, DC 20024

2 Disposal Safety, Incorporated  
1629 K Street NW, Suite 600  
Washington, DC 20006  
S. Amter  
B. Ross

2 I.T. Corporation  
2340 Alamo SE  
Albuquerque, NM 87106  
W. W. Ballard, Jr.  
J. Myers

3 Lawrence Livermore National  
Laboratory  
P.O. Box 808  
Livermore, CA 94550  
W. J. O'Connell  
M. Revelli  
H. Shaw

3 Office of Waste Technology  
Development  
7000 South Adams Street  
Willowbrook, IL 60521  
H. Avci  
A. Branstetter  
J. Cunnane

5 Sandia National Laboratory  
P.O. Box 5800  
Albuquerque, NM 87185  
F. Bingham  
E. J. Bonano  
R. W. Cranwell  
T. O. Hunter  
R. L. Iman

2 University of California  
Department of Nuclear Engineering  
Berkeley, CA 94720  
W. W.-L. Lee  
T. H. Pigford

No. of  
Copies

- 2 U.S. Department of Energy  
Chicago Operations Office  
6800 South Cass Avenue  
Argonne, IL 60439  
R. Baker  
T. Bendokas
- 4 U.S. Department of Energy  
Nevada Operations Office  
P.O. Box 14100  
Las Vegas, NV 89114-4100  
H. Ahagan  
M. Blanchard  
M. Cloninger  
D. Livingston

FOREIGN

AECL - Whiteshell Nuclear Research  
Establishment  
Pinawa, Manitoba  
ROE 1LO CANADA  
N. C. Garisto  
D. M. LeNeveu

G. Bidoglio  
Joint Research Centre  
Radiochemistry Division  
21020 Ispra (VA)  
ITALY

B. Grambow  
Hahn-Meitner-Institut  
GMBH, Postfach 39 01 28  
Glienicker Str. 100  
D-1000 Berlin 39  
FEDERAL REPUBLIC OF GERMANY

N. Kjellbert  
SKB AB  
P.O. Box 5864  
S-102 48 Stockholm  
SWEDEN

No. of  
Copies

- 2 NAGRA  
Parkstrasse 23  
CH-5401 Baden  
SWITZERLAND  
C. McCombie  
I. McKinley

I. Neretnieks  
Royal Institute of Technology  
Department of Chemical  
Engineering  
S-100 44 Stockholm  
SWITZERLAND

E. Peltonen  
TVO Industrial Power Co. Ltd.  
Fredrikinkatu 51-53  
Helsinki  
FINLAND

S. Sharland  
Atomic Energy Research  
Establishment  
Theoretical Physics Division  
Harwell  
Oxon OX11 0RA  
UNITED KINGDOM

C. Thergerstrom  
OECD/Nuclear Energy Agency  
Division of Radiation Protection  
and Waste Management  
38 Boulevard Suchet  
F-75016 Paris  
FRANCE

L.O. Werme  
Swedish Nuclear Fuel and Waste  
Management Co.  
P.O. Box 4864  
S-102 48 Stockholm  
SWEDEN

No. of  
Copies

No. of  
Copies

ONSITE

3 DOE Richland Operations Office

D. C. Langstaff  
D. L. Sours  
J. J. Sutey

Westinghouse Hanford Company

J. D. Davis

50 Pacific Northwest Laboratory

M. K. Altenhofen  
M. J. Apted  
R. C. Arthur  
D. J. Bradley  
J. A. Buchanan  
R. L. Cheatham  
P. G. Doctor

D. M. Elwood  
D. W. Engel (20)  
P. W. Eslinger  
T. L. Gilbride  
A. M. Liebetrau  
B. P. McGrail  
T. E. Michener  
G. C. Nakamura  
A. R. Olsen  
Y. Onishi  
B. Sagar  
R. J. Serne  
B. D. Slonecker  
G. M. Stokes  
B. M. Thornton  
A. E. Van Luik  
C. F. Voss  
R. E. Westerman  
Publishing Coordination  
Technical Report Files (5)