

INFORMATION ONLY

30048-2999 Rev. 00
TSPA; VA7-WPD
April 30, 1998
QA: L

SOFTWARE ROUTINE REPORT
for
WAPDEG (Version 3.06)

CSCI: 30048 v 3.06
DI: 30048-2999 Rev. 00
MI: 30048-M04-001
SCR: LSBR 160

Prepared by: Kevin Mon Date: 4-30-98
K. G. Mon
Performance Assessment Operations

Reviewed by: J. H. Lee Date: 4-30-98
J. H. Lee
Performance Assessment Operations

Reviewed by: D. Stahl Date: 4-30-98
D. Stahl
Waste Package Operations

Concurred by: R. A. Morgan Date: 5-01-98
R. A. Morgan
Engineering Assurance Manager

Approved by: J. A. McNeish Date: 5-1-98
J. A. McNeish
Performance Assessment Operations

INFORMATION ONLY

INFORMATION ONLY

Civilian Radioactive Waste Management System
Management and Operating Contractor
TABLE OF CONTENTS

<u>Section</u>	<u>Page</u>
1.0 Summary.....	4
2.0 Description and Testing.....	4
2.1 Description of WAPDEG.....	5
2.2 Corrosion Models for Carbon Steel.....	8
2.2.1 Humid-Air Corrosion Model for Carbon Steel.....	8
2.2.2 Aqueous Corrosion Model for Carbon Steel.....	10
2.2.3 Incorporation of a Changing Environment.....	11
2.2.3.1 Description of the 'Corrosion Time' Concept.....	11
2.2.3.2 Implementation of the 'Corrosion Time' Concept.....	12
2.2.4 Localized Corrosion Models for Carbon Steel.....	13
2.2.4.1 Humid-Air Localized Corrosion Model for Carbon Steel.....	13
2.2.4.2 Aqueous Localized Corrosion Models for Carbon Steel.....	14
2.3 Corrosion Resistant Material.....	16
2.3.1 General Corrosion Model for Corrosion Resistant Material.....	16
2.3.2 Localized Corrosion Models for Corrosion Resistant Material.....	17
2.4 Spatial Variability in Corrosion Models.....	21
2.4.1 Patches Approach.....	22
2.4.2 Pit Distribution.....	22
2.4.3 Allocation of Variability.....	23
2.5 Galvanic Protection.....	24
3.0 Description of WAPDEG Code.....	25
3.1 General Description.....	25
3.2 Input File Description.....	25
3.3 Treatment of Time.....	37
4.0 Description of Test Cases.....	40
4.1 Description of Test Results.....	42
4.2 Range of Input Parameter Values Over which Results were Verified.....	45
4.3 Identification of Limitations on Software Routine or Validity.....	45
5.0 Reference List of All Documentation Relevant to the Qualification.....	46
6.0 Listings.....	49
6.1 Directory Listing of Executable and Data Files.....	49

6.2 Computer Listing of Source Code	49
6.2.1 Parm306.inc	49
6.2.2 Wap306.inc	51
6.2.3 Wap306.f	57
6.3 Computer Listing of Test Data Input and Output	246
6.3.1 Discussion of Test Data	277
Appendix A. Humid-Air CAM Corrosion Model	284
A1. Humid-Air Corrosion Data Compilation	284
A2. Exposure Parameter Transformations	291
A2.1 Time Fraction for Relative Humidity Greater than 70 %	291
A2.2 Reduction of Relative Humidity and Temperature	292
A3. Humid-Air Corrosion Data References	292
Appendix B. Aqueous CAM Corrosion Model	295
B1. Aqueous Corrosion Data Compilation	295
B2. Aqueous Corrosion Data References	297
Appendix C. Batchwpdg – WAPDEG Unix Script	298
C1. Purpose	298
C2. Listing	298
C3. Description	298
Appendix D. Post306 – The WAPDEG Post Processor	300
D1. Summary	300
D2. Description and Testing	300
D2.1 Description of Post306	300
D3. Description of the Post306 Code	300
D4. Description of Test Cases	304
D4.1 Description of Test Results	304
D4.2 Range of Input Parameter Values Over which Results were Verified	308
D4.3 Identification of Limitations on Software Routine or Validity	308
D5. Reference List of All Documentation Relevant to the Qualification	308
D6. Listing	309

1.0 Summary

The current design concept for a multi-barrier waste package provides the primary component of the engineered barrier system (EBS) for containment and isolation of spent nuclear fuel (SNF) and vitrified defense high-level waste (DHLW) in the potential repository at Yucca Mountain. Fulfillment of the requirements for substantially complete containment and subsequent controlled release of radionuclides into the geosphere will rely upon a robust waste package design, among other EBS components. Initial failure of waste packages subjected to dripping will most likely occur in the form of 'holes' created by localized corrosion of the waste package materials. At later times, general corrosion of the waste package results in the degradation of much larger regions ('patches' as discussed later). Waste packages not subject to drips fail only by general corrosion processes, generally with a lower corrosion rate than is used to model waste packages subjected to dripping. An additional small fraction of waste packages may fail prematurely due to material and/or manufacturing defects or damage during handling. Even if perforated with holes and cracks, the waste package should still be able to provide a substantial barrier to release of radionuclides (Pigford, 1993).

WAPDEG (Waste Package DEgradation) Version 1.0 was originally developed to stochastically simulate waste package degradation as part of the Total System Performance Assessment-1995 (TSPA-1995) (M&O, 1995) for the potential repository at Yucca Mountain. Current WAPDEG versions incorporate a number of empirical and semi-empirical models in order to provide a more realistic representation of waste package degradation. Based on user provided temperature and relative humidity histories at the waste package surface, WAPDEG determines degradation profiles for waste packages undergoing both aqueous and humid-air corrosion processes. These degradation profiles are then used to determine the amount of advective and diffusive radionuclide release from the waste packages.

2.0 Description and Testing

The following has been obtained from the Software Configuration Secretary (SCS) relative to this software:

Software Name:	WAPDEG
Software Version:	3.06
CSCI Identifier:	30048 v 3.06
Document Identifier:	30048-2999 Rev. 00
Media Identifier:	30048-MO4-001

Software Change Request:

LSBR 160

WAPDEG is installed on a Dell PowerEdge 2200 (CRWMS-M&O tag 112371) computer which operates at 266 MHz, and a variety of Hewlett-Packard HP-UX 20 workstations (CRWMS-M&O tags 102877, 112515, 108319, 107436, 107437, 108335, 111031).

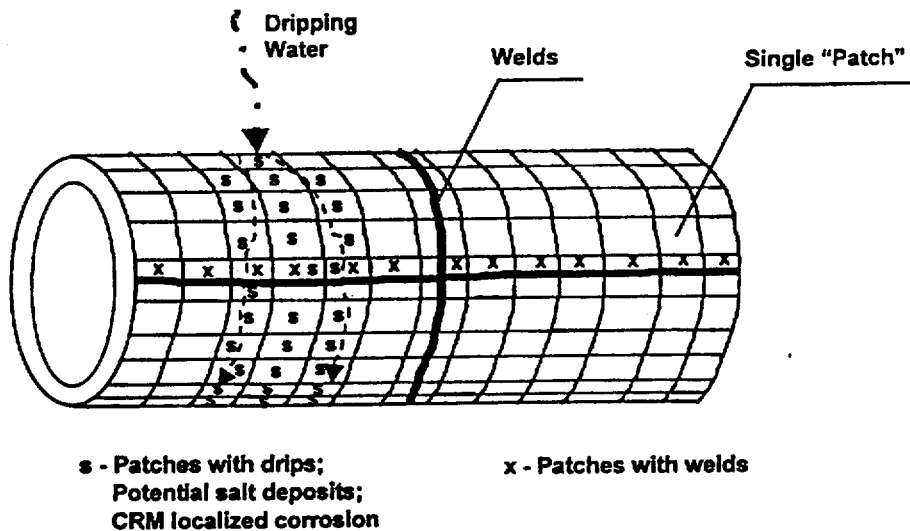
2.1 Description of WAPDEG

WAPDEG has incorporated corrosion models for the candidate waste package materials which result from synthesis and analysis of data from the literature and the projects' corrosion testing and modeling programs. More detailed discussions of the models are given elsewhere (M&O, 1995; Lee, et al. 1996a and 1996b). The current project design for the waste package consists of two concentric cylindrical metal barrier layers: a thick outer layer of carbon steel (Corrosion Allowance Material, CAM) and an inner layer of Corrosion Resistant Material (CRM) (Alloy C-22, in the current design). The thickness of the outer CAM layer is 100 mm, and that of the inner CRM layer is 20 mm (Benton, 1998).

Waste package degradation is modeled by dividing the waste package surface into "patches" and populating the corrosion rates stochastically over the patches, depending on the local exposure conditions. A schematic for this approach is shown below:

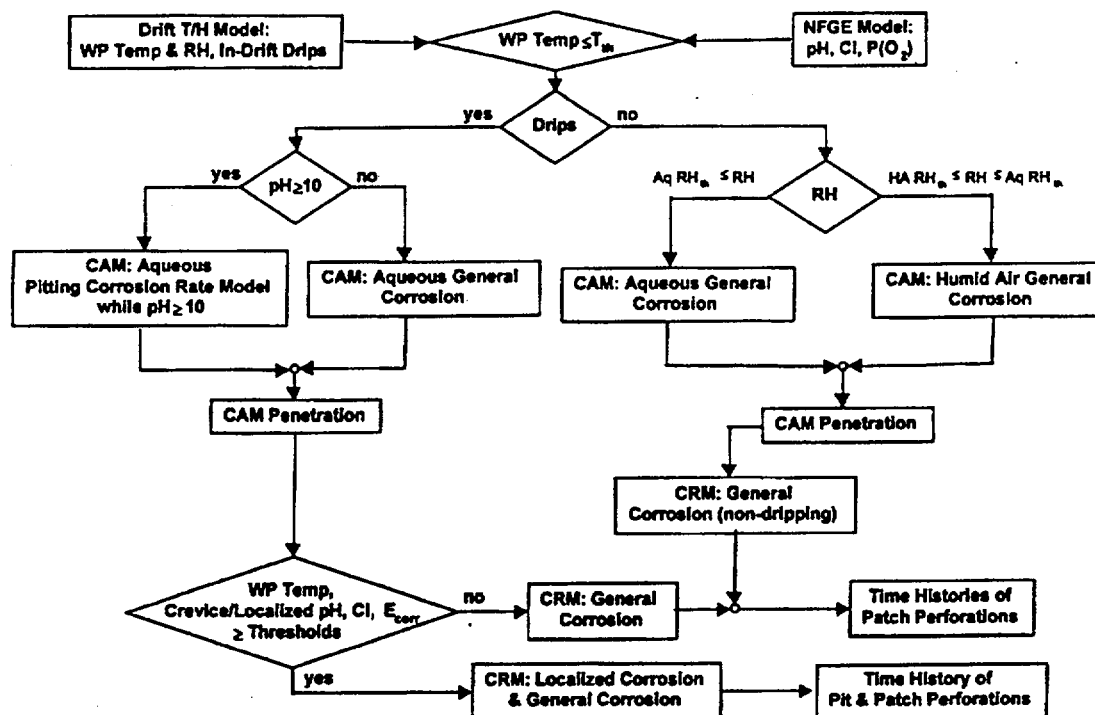
* T, RH, in-drift water dripping
across repository from
drift-scale T-H model abstraction

* pH, [Cl⁻] of dripping water,
P(O₂), across repository
from NFGE model abstraction



The 'patches' approach is an attempt to explicitly represent the variability in corrosion rates within a single waste package and to represent multiple corrosion modes within a single waste package at a given time (Lee, et al, 1997). Variability in waste package degradation across the repository due to varying exposure conditions will be modeled by incorporating explicitly the spatial distribution of the exposure conditions. The exposure conditions intended to be included in the waste package degradation modeling are temperature, relative humidity, in-drift water dripping, and pH and chloride concentration of the contacting water.

The WAPDEG model logic to evaluate the base case is represented schematically below:



In the conceptual model used in WAPDEG, the waste package surface temperature is tested against the temperature threshold (T_{th}) for corrosion initiation of the CAM. When the surface temperature becomes less than the temperature threshold, the waste package can undergo different corrosion degradation modes depending on whether it is dripped on or not.

For waste packages that are not dripped on, the CAM outer barrier undergoes either humid-air or aqueous corrosion dependent on the relative humidity (RH) at the waste package surface. When the CAM is penetrated, aqueous conditions are assumed at the surface of the inner barrier (CRM) patch that is underneath the CAM patch that was penetrated. This assumption is based on observations of the occurrence of capillary condensation of moisture by gel-like porous corrosion products of the outer barrier covering the inner barrier surface (Vernon, 1933) and the hygroscopic nature of corrosion products and other mineral precipitates (Fyfe, 1994; Haynie et al, 1978) which would provide an aqueous corrosion condition at the surface of the inner barrier. The assumption is also based on a potential for crevice formation between the outer and inner barriers, in which moisture could condense and an aqueous condition could exist. The CRM inner barrier patch undergoes general corrosion all the time (after CAM penetration), and fails eventually by a gradual thinning of the inner barrier.

For waste packages that are dripped on, the CAM undergoes aqueous corrosion, and the corrosion mode is dependent on pH of the dripping water. If the pH of the dripping water is greater than or equal

to 10, pitting corrosion with high aspect ratios is assumed to occur; if it is in a neutral pH range (i.e., the pH is greater than 4 and less than 10), general corrosion with low 'localization factor' (commonly called a 'pitting factor') is assumed. When the CAM is penetrated, a crevice is assumed to form on the CRM underneath the CAM patch that was penetrated. The crevice could form between the uncorroded CAM and CRM barriers and/or under CAM corrosion products and other mineral deposits. If the conditions inside the crevice are more aggressive than the crevice corrosion initiation threshold, the CRM undergoes crevice corrosion and general corrosion; if conditions are less aggressive than the initiation threshold, the CRM degrades by general corrosion only. At present, the crevice corrosion initiation threshold is based on the waste package surface temperature at the time of CAM penetration. If this temperature is above a sampled threshold temperature, localized (crevice) corrosion is allowed to occur; if this temperature is below the temperature threshold, only general corrosion processes are modeled. Localized corrosion (pitting and crevice corrosion) is assumed to initiate in the inner barrier only under dripping conditions. This assumption is supported by the Waste Package Degradation Expert Elicitation (M&O, 1998) and various experimental observations on the necessary presence of aggressive ions (such as chloride, etc.) in order to sustain pit growth. The only mechanism for these ions to gain ingress to the crevice at the potential repository is through drips.

The output of the waste package degradation model for waste packages not seeing drips is a time history of patch perforations (no inner barrier localized corrosion). For waste packages that are dripped on, the outputs are time histories for pit perforations and patch perforations. A time-history of structural failure times for the waste packages is also produced based on a CRM thickness failure threshold value input to the model.

2.2 Corrosion Models for Carbon Steel

2.2.1 Humid-Air Corrosion Model for Carbon Steel

A total of 186 atmospheric corrosion data (for up to 16 years of exposure) of cast irons and carbon steels which have similar corrosion behaviors to the candidate corrosion allowance material were collected from literature. The data are from various exposure conditions in tropical, rural, urban, and industrial test locations (Lee, et al 1996a). Important testing parameters considered in the data compilation included test duration, average exposure temperature, average relative humidity, and average sulfur dioxide content of the test atmosphere. The test exposure conditions for the collected data range from 5 to 27°C average temperature, 63 to 84% average relative humidity (RH), and 2 to 406 $\mu\text{g SO}_2/\text{m}^3$ average SO_2 content. Although no significant SO_2 -level is expected in the potential repository, SO_2 -content has a significant effect making it necessary to include the SO_2 -content in the model formulation. A detailed consideration of the data and manipulations used to produce the humid-air corrosion model is presented in Appendix A and in supporting electronic documents (MI: 30048-M04-001).

The following dependencies of general corrosion on exposure conditions in humid-air are based on the current conceptual understanding of corrosion and were incorporated into the model.

$$D_g \propto t^A \quad (1)$$

$$\frac{dD_g}{dt} \propto e^{\frac{B}{RH}} \quad (2)$$

$$\frac{dD_g}{dt} \propto e^{\frac{C}{T}} \quad (3)$$

$$\frac{dD_g}{dt} \propto e^{D[SO_2]} \quad (4)$$

where D_g is general corrosion depth (μm), dD_g/dt is general corrosion rate ($\mu\text{m/yr}$), t is exposure time (years), RH is relative humidity (%), T is temperature (K), and $[SO_2]$ is sulfur dioxide content in the testing atmosphere ($\mu\text{g/m}^3$). A , B , C , and D are constants. By combining Equations (1) through (4), the humid-air general corrosion model can be expressed as follows:

$$\ln D_g = a_0 + \frac{a_1}{T} + \frac{a_2}{RH} + a_3 \ln t + a_4 [SO_2] + \epsilon \quad (5)$$

where a_0 , a_1 , a_2 , a_3 , and a_4 are constants determined from fitting Equation (5) to the collected corrosion data. As discussed in Appendix A, the values of T , RH , and t used in Equation (5) must be adjusted to reflect the exposure conditions during the time periods in which humid-air corrosion actively occurs. ϵ is a term representing uncertainties not explained by the model and is assumed to have a normal distribution with a mean of zero. Linear regression gives the following estimates for the parameters in Equation (5): $a_0 = 17.185$, $a_1 = -623.46$, $a_2 = -974.46$, and $a_3 = 0.62270$. Since there is not expected to be any SO_2 in the repository, a_4 is not used in WAPDEG. The complete covariance matrix was determined to be:

$$V = \begin{bmatrix} 6.9934 & -231.79 & -523.80 & -2.4608 \cdot 10^{-2} \\ -231.79 & 104703 & -10892 & 2.5731 \\ -523.80 & -10892 & 47470 & 1.3009 \\ -2.4608 \cdot 10^{-2} & 2.5731 & 1.3009 & 6.9319 \cdot 10^{-4} \end{bmatrix}$$

and the variance of ϵ was estimated to be 0.12757. This covariance structure was used in WAPDEG to incorporate variability in the general corrosion rate among waste packages and across individual waste packages in humid-air environments.

Initiation of humid-air general corrosion was not modeled explicitly. For simplicity, it was conservatively assumed that humid-air corrosion would occur whenever the relative humidity was above some threshold value (RH_{thresh}). WAPDEG also requires that the temperature be below some threshold (T_{thresh}) for outer (CAM) barrier general corrosion to occur.

2.2.2 Aqueous Corrosion Model for Carbon Steel

Aqueous corrosion data was gathered from a number of sources. Long-term corrosion data (up to 16 years) in polluted river water (Coburn, 1978) and in tropical lake water (Southwell and Alexander, 1970) were used to estimate the time dependence of aqueous corrosion. This data includes the potential effects of microbial activity and various chemical species dissolved in the waters. A set of short-term (100 days) corrosion data of mild steel in distilled water at temperatures from 5 to 90°C (Brasher and Mercer, 1968; Mercer et al, 1968) was used to develop the temperature dependence of aqueous corrosion. This dependence was incorporated into the aqueous corrosion models, but it was scaled to be consistent with the lake and river water since distilled water provides a less corrosive environment.

The following aqueous general corrosion dependencies on exposure conditions were obtained from the literature data and incorporated into the model development:

$$D_g \propto t^E \quad (6)$$

$$\frac{dD_g}{dt} \propto e^{\left(\frac{F_1}{T} + F_2 T^2\right)} \quad (7)$$

where E , F_1 , and F_2 are constants, and other symbols were defined for Equations (1) to (4). By combining Equations (6) and (7), the aqueous general corrosion model is expressed as follows:

$$\ln D_g = b_0 + b_1 \ln t + \frac{b_2}{T} + b_3 T^2 + \epsilon \quad (8)$$

where b_0 , b_1 , b_2 and b_3 are constants to be determined by fitting Equation (8) to the aqueous general corrosion data. ϵ is a term representing uncertainties not accounted for in the model and is assumed to

have a normal distribution with a mean of zero. The estimates of the parameter values are: $b_0 = 111.53$, $b_1 = 0.53199$, $b_2 = -23291$, and $b_3 = -3.1918 \times 10^{-4}$. The covariance matrix for these estimates is:

$$V = \begin{bmatrix} 116.63 & -9.4226 \cdot 10^{-4} & -24761 & -3.7926 \cdot 10^{-4} \\ -9.4226 \cdot 10^{-4} & 7.4149 \cdot 10^{-4} & 1.7704 \cdot 10^{-11} & 2.7491 \cdot 10^{-19} \\ -24761 & 1.1689 \cdot 10^{-11} & 5.2627 \cdot 10^6 & 8.0311 \cdot 10^{-2} \\ -3.7926 \cdot 10^{-4} & 1.8355 \cdot 10^{-19} & 8.0311 \cdot 10^{-2} & 1.2410 \cdot 10^{-9} \end{bmatrix}$$

and the variance of ϵ is 0.0362. This covariance structure was used in WAPDEG to incorporate variability among waste packages and across individual waste packages in the general corrosion rate in aqueous environments. One should note that this covariance matrix is not fully symmetric (the parameters in row 2 column 3 and row 3 column 2 are not equal; same for the parameters in row 2 column 4 and row 4 column 2). The small magnitude of these parameters indicates that time and temperature terms are not strongly correlated, and the asymmetry is due to computational round-off effects. Details of the fitting procedure and data collection for the development of the aqueous corrosion model are presented in Appendix B and in supporting electronic documents (MI: 30048-M04-001).

2.2.3 Incorporation of a Changing Environment

The two general corrosion models described in the previous section assume that the environment is constant, which will not be true of the potential repository. However, time stepping can be refined to an extent so that within each time step the relative humidity and the temperature are relatively constant. In this section, we will discuss the concept of 'corrosion time,' which can incorporate changes in the relative humidity and the temperature between time steps.

2.2.3.1 Description of the 'Corrosion Time' Concept

As general corrosion of the carbon steel outer barrier progresses, the corrosion products accumulate on the surface of the waste package. This corrosion-products layer restricts the movement of the reacting species and the corrosion reaction products, which causes the decrease in corrosion rate with time. In both of the carbon steel general corrosion models, the decreasing corrosion rate with time (due mostly to the accumulation of corrosion products) was reflected with a time term raised to a power which has a value between 0.0 and 1.0. This may be expressed as follows:

$$D_s = f_{NFE} t^d \quad (9)$$

where D_g is general corrosion depth, f_{NFE} includes the constant term and the near-field exposure parameters (i.e. temperature and, for humid-air corrosion, relative humidity) in the model, t is the (real) time elapsed after the initiation of corrosion, and d is constant having a value between 0.0 and 1.0. The rate of general corrosion is given as follows:

$$\frac{\partial D_g}{\partial t} = f_{NFE} d t^{d-1} \quad (10)$$

Equations (9) and (10) are both applicable to a constant exposure condition. In constant exposure environments, the dependence of the general corrosion rate on time is merely a proxy for the actual dependence of the rate on the corrosion product depth. In dynamically changing exposure environments as is expected in the potential repository, Equations (9) and (10) can be generalized to give the corrosion in terms of the corrosion product depth. Substituting $(D_g/f_{NFE})^{1/d}$ (from Equation (9)) for t in Equation (10) gives

$$\frac{\partial D_g}{\partial t} = f_{NFE} d \left[\left(\frac{D_g}{f_{NFE}} \right)^{1/d} \right]^{d-1} = f_{NFE}^{1/d} d D_g^{1-1/d} \quad (11)$$

Equation (11) expresses the general corrosion rate in terms of the general corrosion depth. After the first time step, the general corrosion rate in Equation (11) is bounded. Since the general corrosion depth is known at the beginning of each time step, there will be a unique solution to Equation (11) for any time step after the first time step (Nagle and Saff, 1993, p. 617). Thus, Equation (9) can be used for the first time step and it will suffice to find a solution to Equation (11) with the correct corrosion depth at the beginning of the time step for each time step after the first time step.

2.2.3.2 Implementation of the 'Corrosion Time' Concept

Since the near-field environment inside the emplacement drift of the potential repository changes dynamically with time, f_{NFE} will also change with time. Therefore, a direct use of Equation (9) does not properly capture the effects of the changing exposure conditions. One possible way to represent the effects of changing exposure conditions is to utilize the concept of 'corrosion time' (t^c) and use the corrosion time to account for different exposure conditions in each time step. This entails adjusting the value of t (the real time term in Equation (9)) for changing exposure conditions.

The corrosion time concept and its implementation are described in detail as follows. At the beginning of each time step in which the exposure conditions are relatively constant, the corrosion time for the

current time step is determined based on the corrosion depth at the end of the previous time step and the exposure conditions in the current time step. This may be expressed as follows:

$$t_i^c = \left(\frac{D_{g,i-1}}{f_{NFE,i}} \right)^{1/d} \quad (12)$$

where t_i^c is the corrosion time at the beginning of the current time step i , $D_{g,i-1}$ is the general corrosion depth at the end of the previous time step $i-1$, $f_{NFE,i}$ embodies the exposure conditions for the current time step i , and d is a constant. In simple terms, Equation (12) determines the time (t_i^c) that would be required for corrosion to reach the corrosion depth ($D_{g,i-1}$) if the exposure conditions ($f_{NFE,i}$) of the current time step were applicable for all times following the onset of corrosion. The real time (t) in Equation (9) is replaced with the estimated corrosion time (t_i^c) to calculate the corrosion during the current time step. Using t_i^c in Equation (9) will give the correct corrosion depth at the beginning of the new time interval. Since Equation (9) satisfies Equation (11), using Equation (9) and starting the time interval at t_i^c will give the correct corrosion depth.

2.2.4 Localized Corrosion Models for Carbon Steel

Pitting corrosion is one form of localized corrosion. While it does not affect the entire surface of the waste package, pitting corrosion generally has a much more severe affect on those areas which it does effect. All of the initial waste release could be through pits which have penetrated the waste package while the failed waste package continues to act as a partial barrier to radionuclide release. WAPDEG uses different pitting corrosion models for the outer (CAM) and the inner (CRM) barriers. The site-relevant corrosion testing and model development programs currently underway in this Program should enable incorporation of detailed electrochemical and other physicochemical processes associated with waste package corrosion for future analysis (Farmer 1997; McCright, 1997).

2.2.4.1 Humid-Air Localized Corrosion Model for Carbon Steel

A stochastic pitting corrosion model for the corrosion allowance barrier in humid-air was developed by utilizing a pitting factor that is defined as the ratio of pit depth to the general corrosion depth at the same location on the waste package. This pitting factor is different for each pit, but for a given pit it is assumed constant through time. The results from the extensive corrosion testing programs in inland tropical environments in Panama indicate that the pitting factor for carbon steels and cast irons exposed to 'normal' atmospheric conditions (i.e. in the absence of aggressive conditions such as in acidic or concentrated salt conditions) ranged from 2 to 6 (Southwell and Bultman, 1982; Southwell et al, 1976). The distribution may be similar to a normal distribution but with a long tail to the right

(Marsh and Taylor, 1988; Marsh et al, 1988; Strutt et al, 1985). Accordingly, the pitting factor might be assumed to be normally distributed with a mean of 4 and a standard deviation of 1 (the pitting factor mean and standard deviation are user-supplied quantities in WAPDEG) during early time periods. The recently completed Waste Package Degradation Expert Elicitation (M&O, 1998) indicated that these pits are generally hemispherical in shape and coalesce with time into a more uniform corrosion front with small local roughness perturbations. Thus, the "pitting factor" (henceforth referred to as the "localization factor") appropriate for long time periods is closer to 1.5. In addition, the localization factor is constrained to be greater than or equal to 1, i.e. the local corrosion depth is never less than the average general corrosion depth. The localization factor is sampled randomly and used as a multiplier to the general corrosion depth. Thus, the local corrosion model of corrosion allowance material in humid-air is expressed as follows:

$$D_T = f_p D_g = \begin{cases} \text{normal}(1.5, 0.25) D_g & \text{if normal}(1.5, 0.25) \geq 1 \\ \text{resampled} & \text{if normal}(1.5, 0.25) < 1 \end{cases} \quad (14)$$

where D_T is the total penetration depth and f_p is the localization factor. In the stochastic corrosion modeling, localized corrosion initiation was not explicitly considered. Instead, localized CAM corrosion is conservatively assumed to initiate on the waste package at the same time as when general corrosion initiates.

2.2.4.2 Aqueous Localized Corrosion Models for Carbon Steel

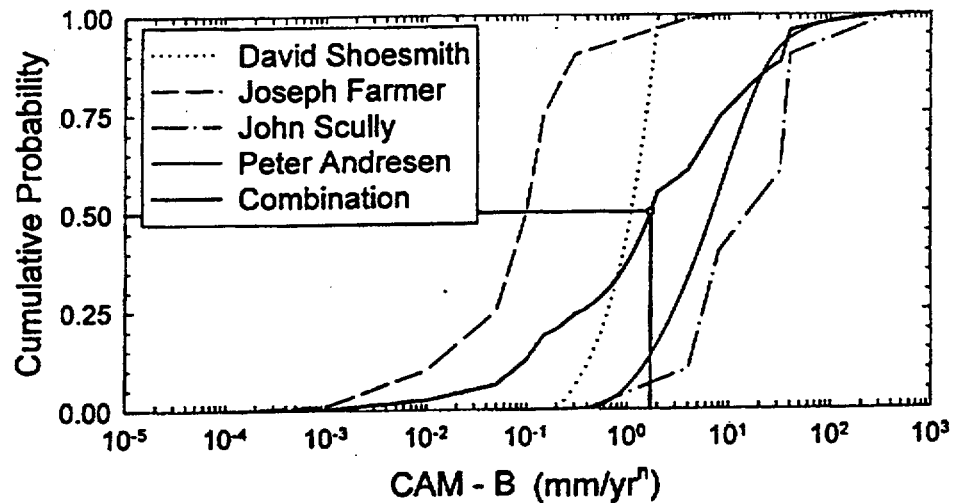
If environmental changes during pit growth justify changing the corrosion conditions from humid-air to aqueous (i.e. the relative humidity becomes greater than the RH_{thresh} for aqueous corrosion), the localization factor remains unchanged. As in the humid-air corrosion model, the localization factor is given by Equation (14).

If a local area of the waste package (a patch as discussed in Section 2.1) is dripped upon, the degradation mode is determined by the chemistry of these drips. For neutral drips, the degradation mode is considered to be identical to that for aqueous corrosion discussed previously. For high pH drips (the elevated pH caused by leaching of OH^- from the concrete drift wall liner) a pit growth law of the following form is used:

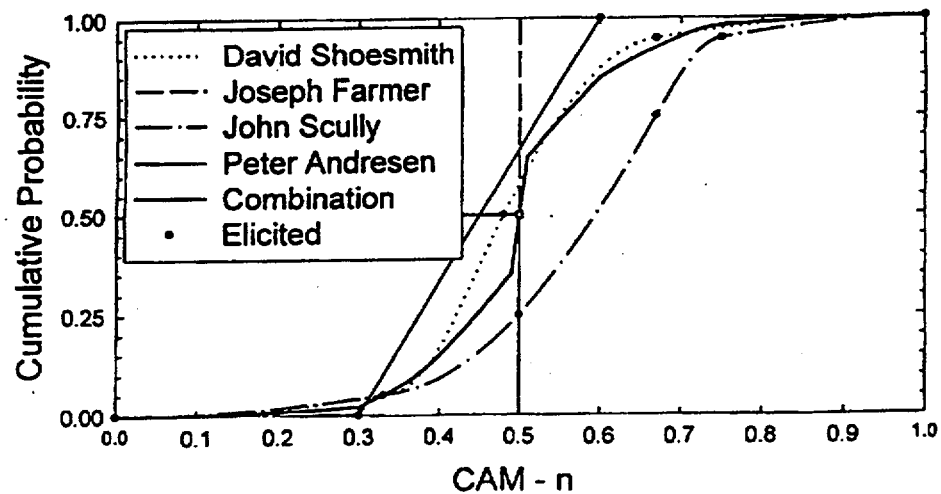
$$D_T = D_g + B t^n$$

where $B t^n$ is the pit depth. B and n are user-specified parameters. Look-up tables for the parameters were provided in the Waste Package Degradation Expert Elicitation (M&O, 1998):

CDF for CAM General Corrosion Rate Term, B



CDF for CAM Localized Corrosion Model n Term



To get a value for the pit depth, values of B and n are sampled independently from one another for each pit and applied throughout the length of the simulation.

2.3 Corrosion Resistant Material

When the outer barrier of the waste package is breached and the inner barrier is exposed, aqueous conditions are assumed at the interface of the barriers. This assumption is based on observations of the occurrence of capillary condensation of moisture by gel-like porous corrosion products of the outer barrier covering the inner barrier surface (Vernon, 1933) and the hygroscopic nature of many corrosion products (Fyfe, 1994; Haynie et al, 1978) which would provide an aqueous corrosion condition at the surface of the inner barrier.

In the current WAPDEG conceptual model, pits are assumed to initiate in the inner barrier only under dripping conditions. This assumption is supported by the Waste Package Degradation Expert Elicitation (M&O, 1998) and experimental observations on the necessary presence of aggressive ions (such as chloride, etc.) in order to sustain pit growth. The only mechanism for these ions to gain ingress to the inside of the emplacement drift is through drips. Conservatively, it is assumed that as soon as a pit has penetrated the outer barrier, corrosion of the inner barrier initiates. While some delay time is expected before localized corrosion initiates, this initiation time is not included in the WAPDEG conceptual model. However, there is an option in WAPDEG to simulate galvanic protection, in which case corrosion of the inner barrier will not occur until the outer barrier is no longer offering this protection of the inner barrier.

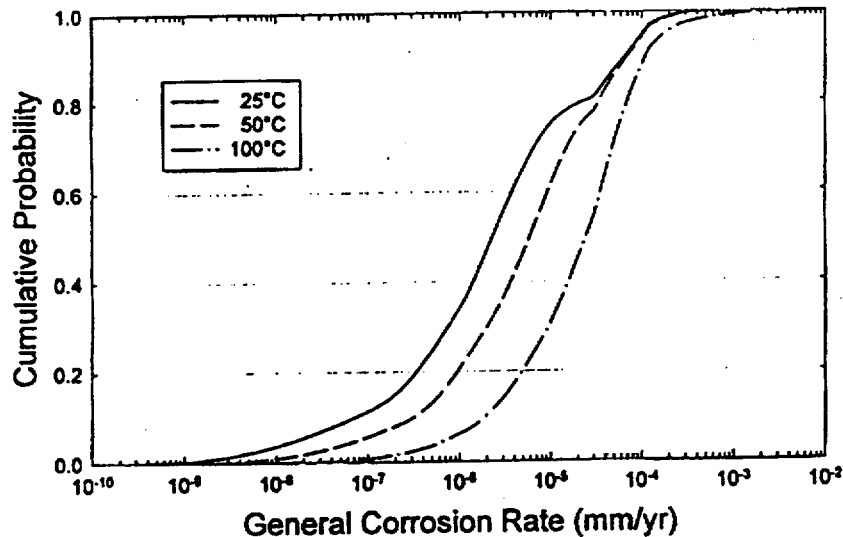
2.3.1 General Corrosion Model for Corrosion Resistant Material

In the current WAPDEG version, the user may choose between two general corrosion models for the CRM. The first of these, referred to as CRMGeneralOnly within the WAPDEG code, uses a CRM general corrosion model of the following form:

$$D_g = A \exp\left(\frac{K}{T}\right)t \quad (15)$$

where D_g is the general corrosion depth (mm), A is a preexponential factor (mm/yr), K is the temperature coefficient (K), and T and t are temperature (K) and time (yr), respectively. A and K are user-supplied parameters. The values of model parameters, A and K, may be based on the Waste Package Degradation Expert Elicitation (M&O, 1998) results. The experts elicited the general corrosion rates of Alloy C-22 at three different temperatures (25, 50 and 100°C) and for a variety of water chemistries. For example,

CRM General Corrosion CDFs at 25, 50, and 100°C
For Dripping Water Chemistry



These elicitations were abstracted and used in a regression analysis to obtain distributions for the model parameters.

The second CRM general corrosion model, referred to as CRMGeneralRateOnly within the WAPDEG code, makes direct use of the Cumulative Distribution Functions (CDFs) for the CRM general corrosion rate at three user-specified temperatures. To determine corrosion rates at temperatures differing from those supplied WAPDEG performs semilog interpolation (on the natural logarithms of the corrosion rates) between the supplied CDFs.

For both of the above CRM general corrosion models, the user may supply different parameters to represent the CRM general corrosion rate in the presence or absence of drips as well as for neutral or chemical drips. In the presence of drips, the user may couple these CRM general corrosion models with the localized corrosion models for the CRM discussed below.

2.3.2 Localized Corrosion Models for Corrosion Resistant Material

At the present time, there are four different localized corrosion modes that may be used to model degradation of the CRM inner barrier. The Table below summarizes the names by which they are referred to internal to the WAPDEG code and their functional forms:

CRM Model	Equation
CRMGeneral+LogPitGrowth CRMGenRate+LogPitGrowth	$D_T = D_g + k \exp \left[Q \left(\frac{1}{T_0} - \frac{1}{T} \right) \right] \log(8766t) + x_0$
CRMGeneral+ArrheniusPit CRMGenRate+ArrheniusPit	$D_T = D_g + \exp \left(b_0 - \frac{b_1}{T} \right) t^n$
CRMGeneral+PitGrowthLaw CRMGenRate+PitGrowthLaw	$D_T = D_g + B t^n$
CRMExponentialPitRate	$D_T = \exp \left(c_0 - \frac{c_1}{T} \right) t^n$

The first three CRM localized corrosion models in the Table make use of the CRMGeneralOnly or CRMGenRate models coupled to a localized corrosion model to obtain the total penetration depth, D_T .

The CRMGeneral+LogPitGrowth and CRMGenRate+LogPitGrowth models couple one of the CRM general corrosion models discussed above with the pit growth law elicited from David Shoesmith (M&O, 1998). Dr. Shoesmith elicited the following values/distributions for the relevant model parameters:

Parameter	Distribution Type	Mean	Standard Deviation
k	Normal	0.75 mm/log(hr)	0.148
x_0	Normal	1.20 mm	0.408
Q	Normal	6615 K	1042.857

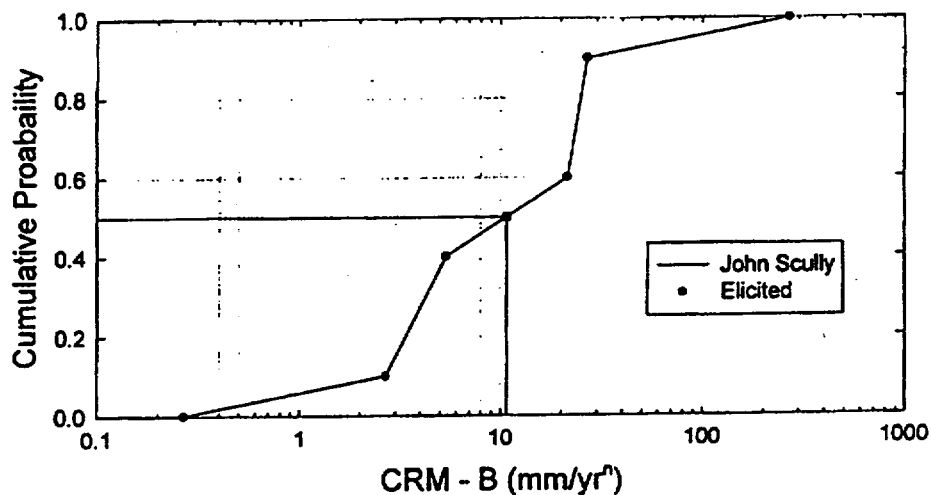
and the value of T_0 is 373.15 K. The 8766 appearing in the log expression is used to convert the time, t , in years into hours, i.e. $8766 = 365.25 \times 24$.

The second CRM localized corrosion models, CRMGeneral+ArrheniusPit and CRMGenRate+ArrheniusPit, couple the CRM general corrosion models discussed above to an exponential pit growth law. This pit growth law originated from analysis of short-term corrosion testing data (Stahl, 1997; McCoy, 1997; Pasupathi, 1997). The pit growth law model was abstracted in a manner similar to what might be used for the CRMGeneralOnly model, i.e., 100,000 pit growth rates were sampled and linear regression techniques were used to determine the best fit pit growth law parameters. These were found to be:

Parameter	Distribution Type	Mean	Standard Deviation
b_o	Normal	13.159	1.7915
b_i	Fixed	5544.5	None
n	Fixed	0.5	None

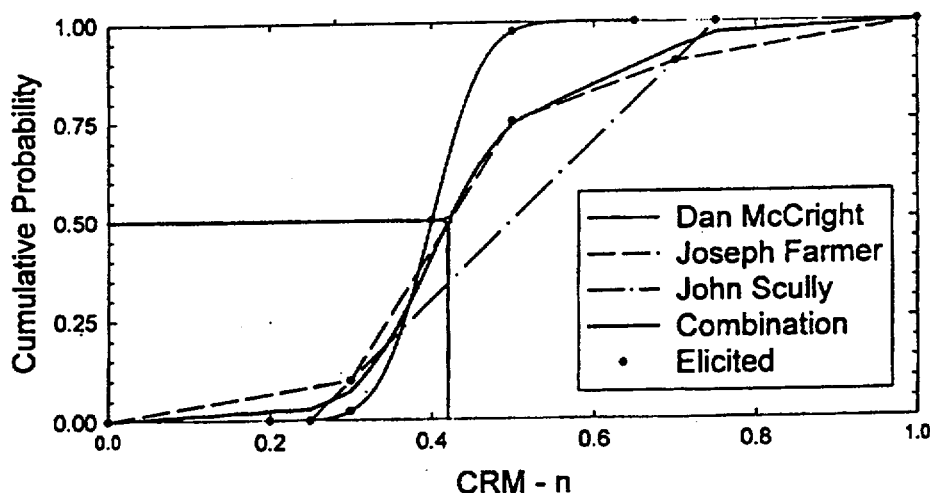
The third CRM degradation models, CRMGeneral+PitGrowthLaw and CRMGenRate+PitGrowthLaw, couple the CRM general corrosion models discussed above to the pit growth law elicited from John Scully (M&O, 1998). Dr. Scully elicited the following distribution for B:

CDF for CRM Localized Corrosion Model B Term



Several experts elicited distributions for the time exponent, n :

CDF for CRM Localized Corrosion Model n Term



No temperature dependence on the pitting rate was provided for this model.

The fourth CRM degradation model, CRMExponentialPitRate, is the inner barrier pitting model used in TSPA-1995 (M&O, 1995) and was developed for Alloy 825 which was then the candidate material for the inner barrier. While the functional form of this model is similar to that of the CRMGeneralOnly model, the conceptual model that underlies it differs. In this model, it is assumed that once a pit penetrates the CAM, a corrosive environment is formed at the base of the penetration that causes initiation of a pit in the CRM at that location. Thus, each CAM pit is matched to a CRM pit whose growth rates increase exponentially with temperature. In the CRMExponentialPitRate model, CRM general corrosion is assumed to be negligible, thus it does not include a CRM general corrosion model, but only a localized aqueous pitting corrosion model. This model was developed from an expert elicitation discussed in a previous TSPA (Andrews et al, 1994). This elicitation is shown below.

Growth Condition	Temp = 70°C		Temp = 100°C	
	Mean Growth mm/yr	95th percentile mm/yr	Mean Growth mm/yr	95th percentile mm/yr
Low growth rate	0.0001	0.001	0.01	0.1
Median growth rate	0.001	0.01	0.1	1.0
High growth rate	0.01	0.1	1.0	10.0

The three rows represent different pit growth-rate scenarios. The expert felt there was about a 50% chance that the growth rate would be higher than the middle row and a 50% chance that it would be lower than the middle row. The text accompanying this table states that the expert felt that there was only a 5% chance that the growth rate would be less than that given in the first row and a 5% chance that it would exceed that given in the third row. The columns represent the distribution of pit growth rates for a given scenario. Thus, if the mean growth rate were 0.001 mm/yr at 70°C, then the experts expected 5% of the pits to have growth rates faster than 0.01 mm/yr.

The CRMExponentialPitRate model is based on the median growth rate scenario. As was discussed in TSPA-1995, only a very contrived distribution would have a 95th percentile value which is a factor of 10 larger than the mean. Thus, the numbers in the first and third column were treated as *median* growth rates and they were used to define an Arrhenius equation for the median growth rate as a function of temperature. Based on the second and fourth columns, it was assumed that at any temperature the pit growth rates would be lognormally distributed such that there would be a 5 percent chance of being a factor of 10 larger than the median. The relation between median pit growth rate and temperature is given by:

$$\ln R_p = c_0 - \frac{c_1}{T}$$

where R_p is the constant pit growth rate (mm/yr), and T is temperature (K). Based on the rates of 0.001 mm/yr at 70°C and 0.1 mm/yr at 100°C, c_0 was estimated to be 50.373 with a standard deviation of 1.3997 and c_1 was estimated to be -19655.85. Using these values, the median pit growth rate will decrease at a rate close to exponential with decreasing temperature, and the rate at room temperature is about 6 orders of magnitude less than the rate at 100°C (373 K). In WAPDEG, pit growth rates for the Alloy 825 inner barrier are sampled randomly, based on the 95th percentile pit growth rate.

2.4 Spatial Variability in Corrosion Models

Since the waste packages will be spread over the repository area, the different waste packages may be exposed to different environments. The different waste packages, and even different locations on the same waste package, may potentially have, on the microscopic scale, different material characteristics and exposure conditions. These may cause variability in corrosion rates, both between different packages and on the same package. Since the environment and even the material characteristics could vary more between packages than within a package, more variability may be expected between locations on different packages than between locations on the same package.

Corrosion results from complicated electrochemical interactions of many factors (including those listed above and others not mentioned) making it difficult to model mechanistically. Thus, in order to

incorporate the variability in the empirical corrosion models from the previous section, WAPDEG models corrosion stochastically. WAPDEG uses the distributions (from the previous section) for the parameters in the various corrosion models to represent the variability between different locations on different waste packages.

2.4.1 Patches Approach

Both the material characteristics and the environment could vary more between locations on the same waste package which are far apart than between neighboring locations on the same package. Thus, the corrosion depth can also be expected to vary more between locations on the same package which are far apart than between neighboring locations on the same package. This can be modeled by imposing a correlation structure on the general corrosion depth at different locations on a waste package. The strength of this correlation should increase as the distance between the two locations decreases.

This conceptual model, with a correlation structure which varies gradually between 1 and some smaller (though still strictly positive) number as the distance between two locations increases, is difficult to simulate. To make this tractable, the waste package is divided into patches. A uniform general corrosion depth is assumed for each patch, while the general corrosion depths of different patches have some correlation to represent the fact that they are on the same waste package.

Literature data, which was used to develop the CAM general corrosion models, provided the general corrosion depth of sample coupons as a function of exposure time (Lee et al., 1996a). This general corrosion depth was derived from the weight loss measurements of sample coupons, and the general corrosion depth at a given exposure time is an average over the entire exposed surface of the sample coupons. Most of the sample coupons were prepared and tested following the American Society for Testing and Materials (ASTM) standard procedures (ASTM G50-76, 1992). The testing standards recommends a sample coupon size of 4 by 6-in. (10.16 by 15.24-cm) which gives the surface area of the sample coupon of about 310 cm^2 (counting both sides). To be consistent with the general corrosion behavior observed experimentally, the patch surface area was taken to be the same size as the ASTM standard test coupon. Thus, the waste package surface area (about 30 m^2 , or 300000 cm^2) was divided by the sample coupon surface area of 310 cm^2 . This process results in a total of 964 patches over the waste package surface.

2.4.2 Pit Distribution

It is known that pits are most stable when growing in the direction of gravity because the dense, concentrated solution within a pit is necessary for its continuing activity (Fontana, 1986, pp. 64-69). Elongation of pits growing in the direction of gravity has been observed (Ruijini, et al, 1989).

Additionally, over the containment and isolation periods, rock may fall on the waste package, or backfill may be introduced. In these cases, there may be enhanced corrosion occurring at the contact points between the rocks and waste package surface. Because of the possibility of enhanced corrosion at the side and bottom of waste packages, it is assumed that pits form uniformly over the entire surface of the CAM, however, as discussed previously, CRM pitting can only occur under dripping conditions.

A pit density of 10 pits/cm² was obtained from previous studies of CAM materials (Marsh and Taylor, 1988; Marsh et al, 1988; Strutt et al, 1985). This gives 3100 pits/patch. In the CAM, pit depths are determined by multiplying the uniform general corrosion depth of the respective patch by the unique localization factor corresponding to each pit. Once a pit penetrates a patch of the outer (CAM) barrier, WAPDEG assumes corrosion processes will initiate on the underlying patch of the inner (CRM) barrier.

2.4.3 Allocation of Variability

There is expected to be variability in corrosion rates between locations on different waste packages. Locations on different patches of the same waste package may have less variability; and locations on the same patch of the same waste package are expected to have even less variability. Information on how this variability should be allocated is not currently available.

The variability allocation in WAPDEG is based on user-defined input parameters. The user specifies the fraction of the total variability to assign to waste packages (referred to as package-to-package variability) and the remainder is assigned to patches within the waste packages (referred to as patch-to-patch variability). For both of the CAM general corrosion models (humid-air and aqueous) the variability in the parameters is divided in the following manner: the covariance matrix, V , is multiplied by the package variability fraction to obtain a new covariance matrix, V_H . Then, V_H is used to select average values of the corrosion parameters for each waste package. To select corrosion parameters for each patch on that waste package, the covariance matrix, V , is multiplied by (1 - the package variability fraction) to obtain a new covariance matrix and this matrix is used to select the patch parameters. For example, 20.221 might be selected as the average value of a_0 (see Equation (5)) for one waste package and the package variability fraction is input as 0.5. In this case, the value of a_0 for each patch on this waste package would be chosen from a normal distribution with a mean of 20.221 and a variance of 3.4967 (half of 6.9934, the total variance). Similarly, for each waste package, average values would be chosen for each of the parameters in the humid-air and aqueous general corrosion models for the carbon steel outer barrier. These average values would be subsequently used to choose values for these parameters for each patch.

Since the outer (CAM) barrier pitting corrosion depth is the general corrosion depth multiplied by the localization factor, the pit depths will reflect the correlation structure of the general corrosion. Based on this, WAPDEG samples every localization factor independently using Equation (14). This will make the amount of variability between pit depths on the same patch smaller than the amount of variability between pit depths on different patches of the same package, which in turn is smaller than the amount of variability between pit depths on different packages.

WAPDEG also takes advantage of the patches approach for inner (CRM) barrier pitting corrosion. This allows the inner barrier corrosion rate to also vary more between patches/pits on different packages than between patches/pits on the same package and to vary more between pits on different patches of the same package than between pits on the same patch. As is the case for the general corrosion models for the outer barrier, there is no information currently available as to how the variability in the inner barrier pitting rates should be allocated. To be consistent with the outer barrier general corrosion models, the same package variability fraction scheme is used.

WAPDEG also allows significant latitude in choosing corrosion initiation temperature thresholds, relative humidity thresholds for humid-air and aqueous corrosion of the outer barrier, the use of multiple temperature/relative humidity histories, and other characteristics of the corrosion modeling process. These are discussed in more detail in Section 3.0 in which the WAPDEG input file format is considered.

2.5 Galvanic Protection

It is generally agreed that in the current waste package design, some degree of galvanic protection of the corrosion resistant inner barrier will be provided by the carbon steel outer barrier. The galvanic protection mechanism may become active when the outer barrier is breached and a galvanic couple is formed between the outer barrier and the inner barrier. Proper galvanic protection is ensured if the two metals maintain intimate contact (both electrical and solution contact). The experts in the recent Waste Package Degradation Expert Elicitation (M&O, 1998) felt that under the conditions expected in the potential repository, galvanic protection could be insignificant as the 'pits' formed in the CAM are likely to be hemispherical in shape, widen rapidly upon contact with the CRM inner barrier, and afford only a short time period of the 'intimate contact' necessary for galvanic protection to occur. The experts felt that only under conditions of high pH pitting of the CAM would outer barrier pits be narrow enough for galvanic protection of the CRM to become operative. When the galvanic protection option is selected, WAPDEG will delay initiation of CRM corrosion processes until the CAM general corrosion depth has reached a user input percentage of the total CAM thickness. WAPDEG applies this criterion at the patch level.

3.0 Description of WAPDEG Code

3.1 General Description

WAPDEG 3.06 is a FORTRAN program 9209 lines in extent. It conforms to the FORTRAN 77 standard and is thus highly portable. WAPDEG has been compiled not only with the FORTRAN 77 compilers on various HP UX 10.0 – PA 2.0 UNIX systems but also with MicroSoft Programmers Workbench 5.1, Lahey Fortran 90, and Digital FORTRAN 5.0 in the Windows/PC environments. WAPDEG is designed to run independently of any other software application, although various utility programs including a script file to run WAPDEG in batch mode (see Appendix C) and a post processor to produce RIP tables and sorted files to facilitate creation of graphic output (See Appendix D) have been written.

3.2 Input File Description

Perhaps the most efficient introduction to the capabilities and usage of WAPDEG can be gained by consideration of its input file. Below is shown an input file (*.inp) for the current WAPDEG version (3.06):

Example Input File

```

** This file will not execute **
    CRM general corrosion models must be the same under all drip conditions

START OF PARAMETERS
3.06                                |Version number of code
2                                  |Number of alternate histories
baseline.hst                       |History file
2, 0.0, 0.0                        |# packages/group and Temp & RH std deviations
va3d-trh.hst                       |History file
3, 0.0, 0.0                        |# packages/group and Temp & RH std deviations
10.0, 2.0                          |Outer,inner barrier thickness (cm)
65.0, 0.5                          |CRM Struc. Failure %, Share of Package Variance
5, 1201, 3100, 3100                |# of packs, patches/pack, CAM, CRM pits/patch
1.0, 1.e5, 1200                    |Bin start time & end time (y), and # of bins
1.e4, 2.e4, 5.e4, 1.e5            |Output times (y) for cumul. pit penetrations
431210                             |Seed for random number generator
4.0, 4.0                           |Max temp, RH change over a time step (C, %RH)
90.0, 10.0                         |Angle defining top(deg), % seeing drips
80.0, 20.0                         |Angle defining bottom(deg), % seeing drips
Fixed                              |Distribution for dripping start time
1000.0                             |Distribution parameter(s)
Uniform                            |Distribution for dripping stop time
20000.0, 30000.0                  |Distribution parameter(s)

```

F, T	Neutral(T/F) water initially, new water (T/F)
Normal	Distribution for water chemistry change time
7000.0, 500.0	Distribution parameter(s)
LogUniform	Distr for time range for ceramic protection
3000.0, 10000.0	Distribution parameter(s)
[No Drip Model, CAM]	This segment always required
CAMGeneral+PitMultiples	CAM corrosion model for no drips
B-Normal	Distribution type for m
1.5, 0.25, 1.0, 1.0e6	Distribution parameter(s)
[No Drip Model, CRM]	This segment always required
CRMGeneralOnly	CRM corrosion model
Normal	Distribution type for ln(A)
3.1359e-1, 2.5761e-2	Distribution parameter(s)
Normal	Distribution type for K=-Q/R
3.4531e+3, 8.3619	Distribution parameter(s)
-2.1436e-1, 1.9387	Covariance and error variance
100.0	Max. CRM Gen. Corr. Rate
[No Drip Features]	This segment always required
File	Distr for thermal protection temperature
TThresh.cdf	Distribution parameter(s)
File	Dist type for humid-air initiation
HARH.cdf	Distribution parameter(s)
File	Dist type for humid-air/aqueous transition
AQRH.cdf	Distribution parameter(s)
1.0	Correlation of HAI _{init} to HA-Aq transition
0.0, 0.0	Galvanic protect depth %, % patches protected
0.0	Spalling depth as a % of thickness
1.0, 1.0	Multiples for CAM, CRM corrosion rates
[Neutral Drip Model, CAM]	Required if any neutral drips can be seen
CAMGeneral+PitMultiples	CAM corrosion model for neutral drips
B-Normal	Distribution type for m
1.5, 0.25, 1.0, 1.0e6	Distribution parameter(s)
[Neutral Drip Model, CRM]	Required if any neutral drips can be seen
CRMGeneral+LogPitGrowth	CRM corrosion model for drips
Normal	Distribution type for ln(A)
1.0272e+1, 1.5535e-2	Distribution parameter(s)
Normal	Distribution type for K=-Q/R
5.9264e+3, 5.0424	Distribution parameter(s)
-7.7948e-2, 7.0498e-1	Covariance and error variance
10.0	Max. CRM Gen. Corr. Rate
Normal	Distribution type for k
0.75, 0.14795918	Distribution parameter(s)
Normal	Distribution for x ₀
1.2, 0.40816327	Value of x ₀
Normal	Distribution for Q
6615.0, 1042.857	Value of Q
[Neutral Drip Features]	Required if any neutral drips can be seen
Fixed	Distr for thermal protection temperature
100.0	Distribution parameter(s)

Uniform	Distr for temperature to init CRM pitting
55.0, 65.0	Distribution parameter(s)
Fixed	Dist type for humid-air initiation
25.0	Distribution parameter(s)
Fixed	Dist type for humid-air/aqueous transition
35.0	Distribution parameter(s)
0.0, 0.0	Galvanic protect depth %, % patches protected
0.0	Spalling depth as a % of thickness
1.0, 1.0	Multiple for CAM, CRM corrosion rates
[Chemical Drip Model, CAM]	Required if any non-neutral drips can be seen
CAMGeneral+PitGrowthLaw	CAM corrosion model for drips
Fixed	Distribution type for B
300.0	Distribution parameter(s)
Uniform	Distribution type for n
0.2, 0.8	Distribution parameter(s)
[Chemical Drip Model, CRM]	Required if any non-neutral drips can be seen
CRMGenrate+ArrheniusPit	CRM corrosion model for drips
3, 1.e+6	Number of dists (temps), max CRM Gen. Rate
25.	Temp appropriate for dist #1
File	Distribution type for #1
gcrm98T1.cdf	Distribution parameter (s)
50.	Temp appropriate for dist #1
File	Distribution type for #1
gcrm98T2.cdf	Distribution parameter (s)
100.	Temp appropriate for dist #2
File	Distribution type for #2
gcrm98T3.cdf	Distribution parameter (s)
Normal	Distribution type for A (c0)
11.276, 2.3848	Distribution parameter(s)
Fixed	Distribution type for K (c1)
5.5514e+003	Distribution parameter(s)
Fixed	Distribution type for n
0.5	Distribution parameter(s)
[Chemical Drip Features]	Required if any non-neutral drips can be seen
Fixed	Distr for thermal protection temperature
100.0	Distribution parameter(s)
Uniform	Distr for temperature to init CRM pitting
55.0, 65.0	Distribution parameter(s)
Fixed	Dist type for humid-air initiation
25.0	Distribution parameter(s)
Fixed	Dist type for humid-air/aqueous transition
35.0	Distribution parameter(s)
0.0, 0.0	Galvanic protect depth %, % patches protected
5.0	Spalling depth as a % of thickness
1.0, 1.0	Multiple for CAM, CRM corrosion rates

This input file will not execute. WAPDEG requires the CRM general corrosion models to be identical in the [No Drip Model, CRM], [Neutral Drip Model, CRM], and [Chemical Drip Model, CRM] input segments. Three different segments are included for illustration purposes only.

All of the text that appears above the 'START OF PARAMETERS' statement is copied into the output file (SamePrefixAsInput.out) and can be used for any additional comments the user feels is necessary to distinguish the output. The version number is read and checked against the version of WAPDEG being executed; a mismatch in version numbers will halt program execution. The next line of the input file:

```

2                               |Number of alternate histories
baseline.hst                   |History file
2, 0.0, 0.0                   |# packages/group and Temp & RH std deviations
va3d-trh.hst                   |History file
3, 0.0, 0.0                   |# packages/group and Temp & RH std deviations

```

specifies the number of package groups which will be used to represent local repository conditions (spatial variability in the drift-scale thermohydrology of the repository). The next lines in the input file are composed of pairs of input lines in which the first line specifies the name of the mean history file (*.hst, composed of three columns of data; time, temperature, and relative humidity) to be used for the package group followed by a line composed of three values; the number of packages that the history file is to be applied to (the number of packages in the group), and the standard deviations for allowed variations in T and RH from the package group's mean history file. For each package, a normal (mean zero, standard deviation as given) value is sampled and applied as a constant (with time) offset from the mean history. In this case WAPDEG will use 'baseline.hst' for the first two packages modeled and 'va3d-trh.hst' for the next three and there will be no variations allowed from the T and RH data given in the history file. The next input lines:

```

10.0, 2.0                     |Outer, inner barrier thickness (cm)
65.0, 0.5                     |CRM Struc. Failure %, Share of Package Variance
5, 1201, 3100, 3100          |# of packs, patches/package, CAM, CRM pits/patch
1.0, 1.e5, 1200              |Bin start time & end time (y), and # of bins
1.e4, 2.e4, 5.e4, 1.e5      |Output times (y) for cumul. pit penetrations
431210                        |Seed for random number generator
4.0, 4.0                      |Max temp, RH change over a time step (C, %RH)

```

allow for input of the outer and inner barrier thicknesses in cm, although internally WAPDEG measures the outer barrier in units of μm and the inner barrier in units of mm. The first value on the next line is related to determination of a 'structural failure time' for the waste package. For each waste package, WAPDEG calculates the average (over all patches) CRM general corrosion depth at each time step. When this average depth exceeds a given percentage (65% in this case) of the total CRM thickness, the package can be considered to have structurally collapsed (presumably due to static loads from its own weight, the weight of rock, etc.). WAPDEG still continues to model corrosion after this

time and the structural failure time is output (to the .out file) for possible use in other models. The second value on this line allows the user to specify the fraction of the total variability in the various model parameters should be assigned to package-to-package variability and $1 -$ (this value) is assigned to patch-to-patch variability. The line following allows the user to specify the number of packages, patches/package, and pits/patch in both the CAM and CRM to be simulated. The next line contains three values related to the storing of temporal results (pit and patch penetrations with time). The start time corresponds to the beginning of the simulation; if the history file does not contain data for this starting time, the simulation is assumed to start with the earliest time step given in the history file. The same is not true for the end time as a time step is added to the end of the history with the same temperature and relative humidity as the last time step in the history file if the history file does not contain data for the end time. The third value specifies how many (logarithmically distributed) time 'bins' are used to store the cumulative pit penetration profiles. In this way, the resolution of the pit penetration profiles reported by WAPDEG is under user control.

The next line allows the user to specify up to 4 time values of particular interest; what is reported by WAPDEG are the cumulative number of pit/patch penetrations in the time bin that contains the specified time values, not the actual number of pit penetrations at the exact time specified. The following input line allows specification of the random number seed used to initialize the random number generator. The next line allows the user to enter the maximum allowed change in T and RH between time steps. WAPDEG will add time steps to the history file and interpolate T and RH data until these criteria are satisfied. The corrosion models used in WAPDEG require the application of constant environments for the duration of each time step. Through wise application of these values the user may increase the accuracy of the WAPDEG output. Entry of negative or zero values results in no modification of the history file applied to a given package.

The next lines:

90.0, 10.0	Angle defining top(deg), % seeing drips
80.0, 20.0	Angle defining bottom(deg), % seeing drips
Fixed	Distribution for dripping start time
1000.0	Distribution parameter(s)
Uniform	Distribution for dripping stop time
20000.0, 30000.0	Distribution parameter(s)

deal with application of dripping models; the user can enter the angular range that defines the 'top' of the package and the percentage of top patches that are subject to dripping as well as those same parameters for the 'bottom' of the package. The remaining patches are defined to be 'side' patches. Side patches differ from top or bottom patches in that they can never be dripped upon. WAPDEG allows dripping on a given (top or bottom) patch to start sometime after the beginning of the simulation, to change chemistry (corrosion model parameters (specified later in the input file)) sometime after that, and then for dripping to cease. The user must enter a dripping start time and end time, which can be given by one of WAPDEG's built-in distributions. The user may choose 'Fixed',

in which case the next input line must contain the fixed value; 'Uniform', in which case the next input line must contain the upper and lower bounds used to define a uniform distribution (as in the 'dripping stop time' shown above); 'Normal', in which case the next input line must contain the mean and standard deviation relevant to the desired normal distribution (as shown above for the 'chemistry change time'); 'B-Normal' for bounded normal, in which case the user must specify the mean and standard deviation, as well as lower and upper bounds (cut-offs); 'LogUniform', in which case the user must specify the lower and upper bounds; 'LogNormal', in which case the user must subsequently specify the mean and standard deviation relevant to the underlying normal distribution; or 'File' in which case the user must supply a the name of a file (look-up table). In the 'File' option, the first line of the file used must consist of an integer indicating how many data pairs follow and these data pairs must consist of a value for the property that the file represents and a corresponding CDF value (between 0 and 1). The user may also specify 'Default' in which case WAPDEG will use CDF values hardwired in the code which correspond to those obtained from the recent Waste Package Degradation Expert Elicitation (M&O, 1998) where possible. These distribution types are summarized in the Table below:

Distribution	Parameters
Default	Hardwired in code
Fixed	Single value
Uniform	Lower, upper bound
Normal	Mean, standard deviation
B-Normal	Mean, standard deviation, min. value, max. value
Loguniform	Lower, upper bound
Lognormal	Mean, standard deviation of underlying normal
File	Read from an input file

In addition to dripping start and stop times, the user can cause dripping to change chemistry (corrosion parameters) at some time during the WAPDEG simulation. The input lines:

F, T	Neutral (T/F) water initially, new water (T/F)
Normal	Distribution for water chemistry change time
7000.0, 500.0	Distribution parameter(s)

allow the user to specify whether the chemistry (corrosion parameters) of the dripping water is neutral (T) or chemical (F) in nature (first input on the line) and whether the dripping water changes chemistry (T) or not (F). If the dripping water changes chemistry, the user must specify a distribution for the water chemistry change time.

The next input lines:

LogUniform
3000.0, 10000.0

|Distr for time range for ceramic protection
|Distribution parameter(s)

are concerned with modeling the use of a ceramic coating to offer additional corrosion protection to the waste package. Ceramic coatings are not part of the 'base case' WAPDEG model, but are included for 'sensitivity' studies. In the current WAPDEG version, the ceramic coating is considered to offer complete corrosion protection to the waste package until it fails in a patch-wise manner, i.e., the coating does not completely fail at a sampled time; rather it fails locally based on a failure time sampled from the distribution input.

The next input segment:

[No Drip Model, CAM]	This segment always required
CAMGeneral+PitMultiples	CAM corrosion model for no drips
B-Normal	Distribution type for m
1.5, 0.25, 1.0, 1.0e6	Distribution parameter(s)
[No Drip Model, CRM]	This segment always required
CRMGeneralOnly	CRM corrosion model
Normal	Distribution type for ln(A)
3.1359e-1, 2.5761e-2	Distribution parameter(s)
Normal	Distribution type for K=-Q/R
3.4531e+3, 8.3619	Distribution parameter(s)
-2.1436e-1, 1.9387	Covariance and error variance
100.0	Max. CRM Gen. Corr. Rate

specifies the outer barrier (CAM) and inner barrier (CRM) corrosion models to be used when drips are not present. WAPDEG offers a variety of CAM and CRM models to choose from. The CAM and CRM models available are listed in the Tables below:

CAM Models	Equation
CAMGeneral+PitMultiples	$D_T = mD_s$
CAMGeneral+PitGrowthLaw	$D_T = D_s + Bt^n$

CRM Model	Equation
CRMGeneralOnly	$D_s = A \exp\left(\frac{K}{T}\right)t$
CRMGeneralRateOnly	$D_s = R(T)t$, $R(T)$ sampled from CDFs

CRM Model	Equation
CRMGeneral+LogPitGrowth CRMGenRate+LogPitGrowth	$D_T = D_s + k \exp \left[Q \left(\frac{1}{T_o} - \frac{1}{T} \right) \right] \log(8766 t) + x_o$
CRMGeneral+ArrheniusPit CRMGenRate+ ArrheniusPit	$D_T = D_s + \exp \left(b_o - \frac{b_1}{T} \right) t^n$
CRMGeneral+PitGrowthLaw CRMGenRate+PitGrowthLaw	$D_T = D_s + B t^n$
CRMExponentialPitRate	$D_T = \exp \left(c_o - \frac{c_1}{T} \right) t^n$

The CAMGeneral+PitMultiples model for the CAM requires a pit multiple, m , distribution. The CAMGeneral+PitGrowthLaw for the CAM requires distributions for B and n in the pit growth equation. Both of these CAM models use the humid-air and aqueous CAM general corrosion models discussed previously.

The CRMGeneralOnly model for the CRM requires distributions for the general corrosion rate parameters. These include a distribution for $\ln(A)$ and K , as well as their covariance and the error variance of the model. The CRMGeneralRateOnly model (as illustrated in the [Chemical Drips, CRM] general corrosion input segment),

```

3, 1.e+6      |Number of dists (temps), max CRM Gen. Rate
25.           |Temp appropriate for dist #1
File          |Distribution type for #1
gcrm98T1.cdf  |Distribution parameter (s)
50.           |Temp appropriate for dist #1
File          |Distribution type for #1
gcrm98T2.cdf  |Distribution parameter (s)
100.          |Temp appropriate for dist #2
File          |Distribution type for #2
gcrm98T3.cdf  |Distribution parameter (s)

```

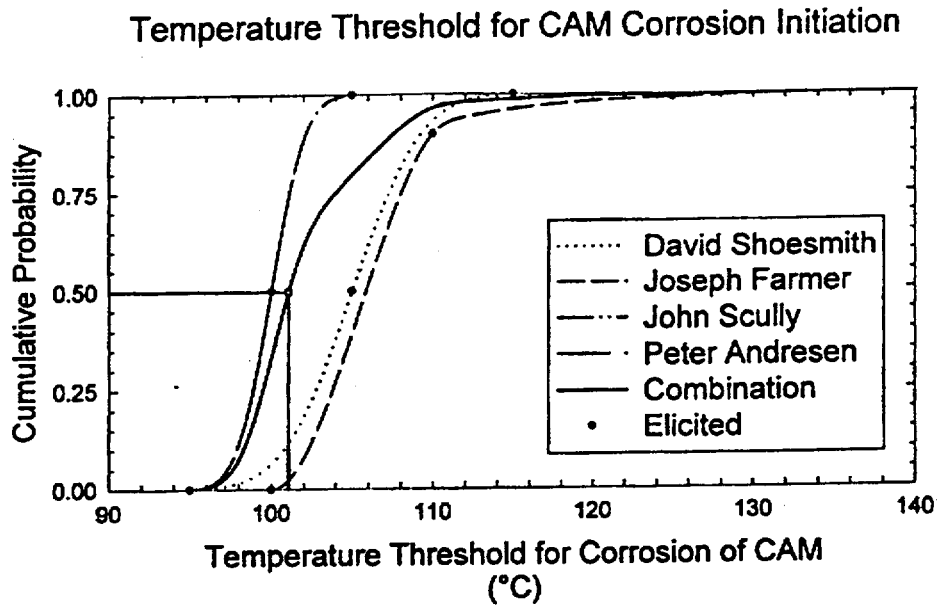
requires specification of up to three temperatures and general corrosion rate distributions (these can be any of the ones built in to WAPDEG). Note that the maximum CRM general corrosion rate is specified just after the number of distributions for the CRMGenRate model while it appears after the specification of the covariance and error variance terms for the CRMGeneral model.

The CRMGeneral+LogPitGrowth and CRMGenRate+LogPitGrowth CRM models require distributions for the general corrosion parameters as well as for the localized corrosion parameters, k , Q , and x_0 . The CRMGeneral+ArrheniusPit model for the CRM requires the distributions for the general corrosion parameters as well as for the localized corrosion parameters, b_0 , b_1 , and n . The CRMGeneral+PitGrowthLaw and CRMGenRate+PitGrowthLaw models for the CRM require distributions for the general corrosion parameters and distributions for B and n . This model has no temperature dependence on the localized corrosion parameters. The CRM degradation model used in TSPA-1995 (M&O, 1995) can be selected via the ExponentialPitRate model. It requires a normal distribution for c_0 , a fixed value for c_1 , and a distribution for n , the time power term. For this model, the number of pits/patch in the CRM and CAM must be identical as the conceptual model used requires that CRM pits initiate only at sites where pits have penetrated the CAM.

The next set of input lines:

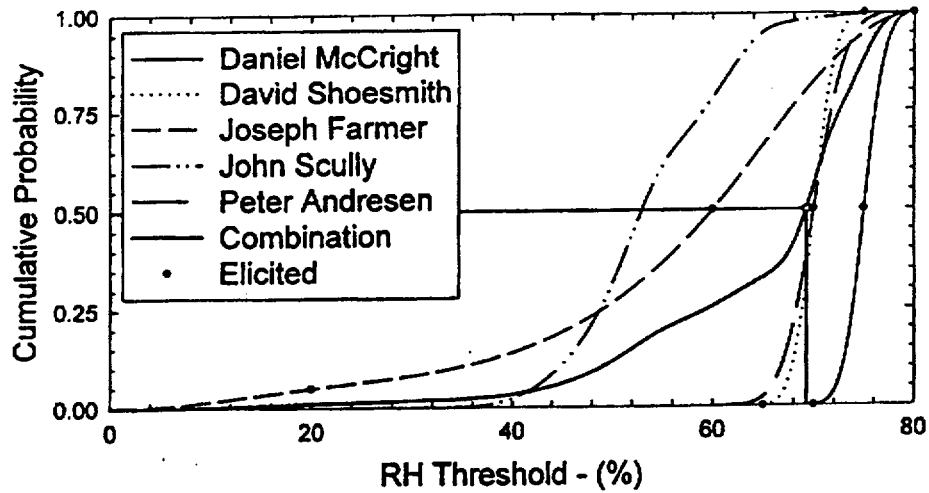
[No Drip Features]	This segment always required
File	Distr for thermal protection temperature
TThresh.cdf	Distribution parameter(s)
File	Dist type for humid-air initiation
HARH.cdf	Distribution parameter(s)
File	Dist type for humid-air/aqueous transition
AQRH.cdf	Distribution parameter(s)
1.0	Correlation of HAINit to HA-Aq transition
0.0, 0.0	Galvanic protect depth $\dagger$, $\dagger$ patches protected
0.0	Spalling depth as a $\dagger$ of thickness
1.0, 1.0	Multiples for CAM, CRM corrosion rates

specify the 'No Drip Features.' These features include a distribution for the 'thermal protection temperature,' i.e., the temperature above which no corrosion is allowed to occur. Such a temperature threshold is used primarily because water will not exist in the liquid phase above this temperature, and thus no corrosion should occur. This threshold could be above the typical boiling point of water due to the presence of impurities in the water film on the waste package and variations in water chemistry. In this example, the temperature threshold is read from a file, TThresh.cdf, resulting from values elicited from the Waste Package Degradation Expert Elicitation (M&O, 1998), and is shown in the following plot:

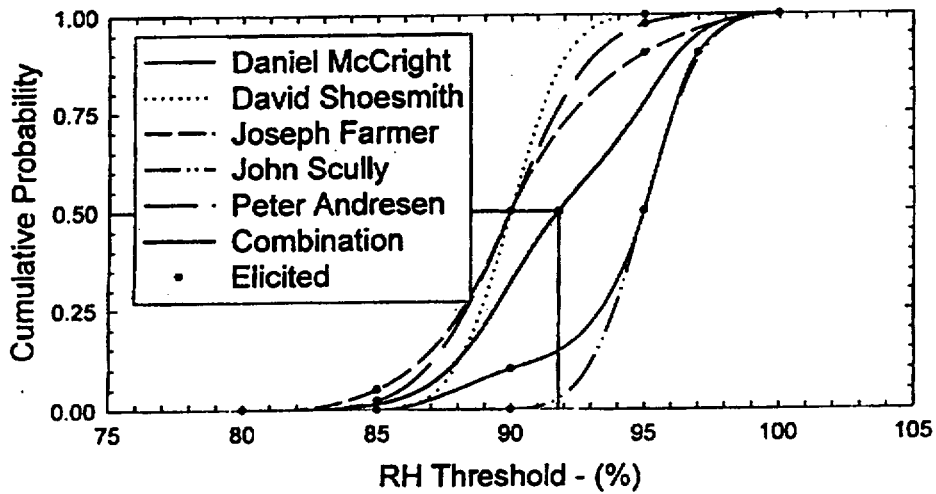


If a negative or zero value is entered, a temperature threshold of 100°C is assumed. Similarly, WAPDEG uses RH thresholds (also from the Waste Package Degradation Expert Elicitation (M&O, 1998)) for the onset of both humid-air and aqueous corrosion modes of the CAM.

RH threshold for Humid Air CAM Corrosion Initiation



RH Threshold for Aqueous CAM Corrosion Initiation



The next input line allows the user to control the degree to which these RH thresholds are correlated. Entering a value of 1 forces the humid-air and aqueous RH thresholds into perfect correlation, i.e. if the aqueous RH threshold has a Cumulative Probability of 0.56, the humid-air RH threshold with a Cumulative Probability of 0.56 will also be selected. Perfect negative correlation (entering -1) would result in the humid-air RH threshold with the Cumulative Probability of $1 - 0.56 = 0.44$ being used.

The 'galvanic protection depth %' specifies the percent of the total CAM thickness that must remain in order for the CAM to afford galvanic protection to the '% of patches protected' of the CRM. In this way the user can control the characteristics of the galvanic protection model used in WAPDEG. The subsequent input line allows specification of how thick a corrosion product film must be before it fails to adhere to the substrate and 'spalls' off the waste package surface. Spalling is not part of the TSPA-VA base case and is not implemented in the current WAPDEG version, but is expected to be in later versions of the code. The 'Multiples for the CAM, CRM corrosion rates' are used in sensitivity studies to allow for modeling of corrosive environments that are either exceptionally aggressive (multiples > 1) or exceptionally benign (multiples < 1). The corrosion rate multiples are typically used when no mechanistic model is available for the phenomena under consideration, however, the investigator is convinced that the conditions are more aggressive or benign than those typically used in WAPDEG simulations.

The next three segments of the input file, and the three succeeding these, are quite similar in form to the preceding three. The user enters [Neutral Drip Model, CAM], [Neutral Drip Model, CRM], and [Neutral Drip Features] as well as [Chemical Drip Model, CAM], [Chemical Drip Model, CRM], and [Chemical Drip Features] segments in much the same manner as was done for the [No Drip . . .] input segments. These may differ in salient parameters such as those used to specify the general corrosion rates, RH thresholds, etc., but the form of the input is identical to that described above for the [No Drip . . .] sections with the exception of two lines of input related to specification of a CRM temperature threshold for initiation of localized corrosion. These two lines appear immediately after specification of the temperature threshold for corrosion initiation only in [. . . Drip Features] segments. The CRM temperature threshold for initiation of localized corrosion is implemented such that if the temperature at the waste package surface is below the CRM temperature threshold for initiation of localized corrosion when the CAM is penetrated, no localized CRM corrosion occurs. However, once localized corrosion (pitting) initiates, it continues regardless of the temperature, although, with the exception of the PitGrowthLaw model, at a temperature dependent rate.

One should note that it is only necessary to enter the input segments that are used by WAPDEG, i.e., if the percent of dripped on patches is zero, it is unnecessary to provide the neutral or chemical drip input segments. Similarly, it is also unnecessary (will result in an error message) to supply input lines related to dripping start/end/chemistry change times if no dripping can occur.

3.3 Treatment of Time

The following discusses how WAPDEG treats the time grids used to define the temperature and RH history. In general terms WAPDEG will:

1. Get temperature and RH curves for each group of waste packages.
2. Perturb these curves for each particular waste package in a group.
3. For each waste package, develop two sets of curves; one for patches that never see drips and one for patches that will see drips.
4. For each patch, start with the appropriate curves but potentially modify them if ceramic protection is being considered.
5. Track general corrosion and pitting (or roughness) on the CAM. Reset time zero on the grid to the time when the CAM is first penetrated.
6. Use the surviving grid to model corrosion on the CRM.

1. Get temperature and RH curves for each group of packages.

The setup for this occurs while reading the input file: The user supplies the number of waste package groups and the file name for the history file intended for each group. Within WAPDEG's loop over waste packages, each time the loop counter enters a new waste package group the next history file is read. There are two ways the history file can be modified at this point:

- a. The last time listed in the history file is less than the last time for which binned results are requested (simulation time). In this case a time step is added at simulation time. The temperature and RH at this new time are the same as those for the last time on the file, effectively extending a constant environment to the end of simulation.
- b. The user wants to restrict the size of temperature and/or RH changes from one end of a time step to the other. In this case several new time steps may be inserted between existing time steps with interpolated values. In all cases WAPDEG assumes that the best interpolation is found by assuming that temperature is linear with the log of time. The same is true with RH versus time.

Both modifications potentially add new time steps, so in both cases there is a check to ensure that arrays that hold time step information do not have their dimensions exceeded. The resulting time grid is saved so that it can be used as the initial grid for each new package in the group.

2. Perturb these curves for a particular waste package in a group.

In addition to the file name for the history file, the user provides a standard deviation for temperature (σ_T) and RH (σ_{RH}) for each package group. For each package in a group WAPDEG samples two values from the standard normal distribution, say z_1 and z_2 . Then the product $z_1 \sigma_T$ is added to the temperature at each point on the time grid. Similarly the product $z_2 \sigma_{RH}$ is added to the RH at each

point on the time grid. Thus, each package will have temperature and RH curves that are different from, but parallel to, the curves read in (and modified as described in 1) from the history file. Of course the curves will be the same for all packages within a group if the user provides zeroes for the standard deviations.

Inherent in this treatment is the assumption that variability of temperature and RH can be described with a normal distribution. If this feature becomes more widely used in the future, we may need to make the treatment more flexible in terms of the distributions assumed.

3. For each waste package develop two sets of curves, one for patches that never see drips and one for patches that will see drips.

At this point in the code the curves for temperature and RH are available for the package and the temperature and RH thresholds have been sampled and are available. Recall that there are 9 thresholds (for the CAM): One temperature and two RH thresholds for each of the three conditions - no drips, neutral water drips, and non-neutral water drips. WAPDEG develops two time grids for the patches on the current package, one for patches that never see drips and one for patches that will see drips.

For the patches that do not see drips, time starts at zero and ends at the end of simulation time. WAPDEG searches through the history and finds each time that any of the three no-drip thresholds are crossed. When detected, WAPDEG inserts a time step at the point in time when the crossing occurs. The temperature (RH) at the crossover time is set to the temperature (RH) threshold. The RH (temperature) at the crossover time is then linearly interpolated versus the temperature (RH) change. The crossover time is interpolated on its logarithms. Note that by interpolating temperature (RH) versus log time and interpolating RH (temperature) versus temperature (RH) linearly, RH (temperature) is also interpolated versus log time.

A potential issue arises at the crossover times. By setting the temperature (RH) to its exact threshold, can WAPDEG know whether corrosion is to proceed or not? This issue is avoided since WAPDEG averages temperature (RH) over each time step and uses the average throughout the time step. So for example, if the temperature at the beginning of the time step exceeds the temperature at the end of the time step (threshold temperature in this example), the average of the two will exceed the temperature threshold. Thus, no corrosion will occur during such a time step.

Developing a time grid for the dripped on patches is more complex. There can be up to four time segments on these grids: An initial period of no dripping, a period of dripping with type 1 water, a period of dripping of type 2 water, and a period of no dripping again. Any subset of these segments can also be appropriate as long as the order of occurrence of the segments is maintained.

The time at which dripping starts, stops, and changes character for each waste package are sampled according to their distributions. If the user is not careful (or in the presence of infinite ranges, e.g.

normal), times could be sampled that indicate dripping stops before it starts or changes character before starting or after stopping. The rules in WAPDEG are

- a. If dripping start time is after simulation time, no drips occur.
- b. If dripping stop time is before dripping start time, no drips occur.
- c. If dripping stop time is after simulation time, drips never stop.
- d. If drip change time is before drip start time, only water type 2 will occur.
- e. If drip change time is after drip stop time or simulation time, only water type 1 will occur.

All rules are straightforward, with the possible exception of rule (b). This rule assumes that the drip stop distribution takes precedence over the drip start distribution. It is possible to make the opposite rule.

WAPDEG looks at each segment in turn and inserts a time step for the time the segment ends. Also it inserts time steps within the segment for crossovers, as necessary, using the thresholds appropriate for the segment. It is possible to have gaps in time when no corrosion occurs. If temperature and RH are both monotonic, such gaps can only occur at the beginning of a segment (or possibly the entire segment). WAPDEG keeps track of whether corrosion can occur in each segment. Note that the same thresholds are used for both the initial and final no-drip segments, if they both appear.

When modeling corrosion, WAPDEG has an outer loop over segments and an inner loop over time steps in the segment. As will be discussed below, it is possible to eliminate entire segments (e.g., ceramics may postpone corrosion to a time after dripping stops). But, the elimination of segments occurs at the patch level.

Finally, time steps are inserted both at the segment boundaries and at each threshold crossover. WAPDEG checks that inserted times do not cause dimensions to be exceeded.

4. For each patch, start with the appropriate curves but potentially modify them if ceramic protection is being considered.

For each patch on the package, WAPDEG first determines whether it sees drips. Depending on the determination, WAPDEG initializes the time grid for the patch to the appropriate package time grid. A corrosion delay time (presumably due to a ceramic layer) is then sampled for the patch. If the time is positive and less than simulation time, the grid is shifted so that the end of ceramic protection becomes time zero on the grid. This shifting can cause the elimination of one or more segments. WAPDEG re-establishes the number of segments and the corrosion start time.

5. Track general corrosion and pitting (or roughness) on the CAM. Reset time zero on the grid to the time when the CAM is first penetrated.

The grid for the patch is used to track corrosion of the CAM. On the first penetration of the CAM of any kind, corrosion starts on the CRM. At this point the grid is slid so that CAM penetration time becomes time zero. Again, segments may be totally or partially lost, so WAPDEG counts the lost segments and adjusts.

Aside. There is a temperature threshold for the CRM below which pitting corrosion does not initiate. If the temperature at the time of CAM penetration exceeds the threshold (and the patch is a dripped-on patch), WAPDEG proceeds with CRM corrosion as though there were no threshold. If the initial temperature is too low for the first surviving segment, only general corrosion is modeled in that segment. However, since the next segment could have a different threshold, the temperature comparison is repeated for the next segment, if present. Thus, the CRM temperature threshold does not affect the time grid in WAPDEG.

6. Use the surviving grid to model corrosion on the CRM.

The time grid of (5) is used to predict penetrations of the CRM. When a time of penetration is found, the sum of the time of first CAM penetration plus the time of ceramic failure is added, thereby giving a time relative to the initial time grid of (1) above. Minimums and binned results are based on such times.

4.0 Description of Test Cases

WAPDEG undergoes a significant number of tests before a new version is accepted for baselining. Most of the tests focus around analysis of output generated through the use of constant history files (with constant values of T and RH for all time) and comparison of these results to those generated through 'hand' calculations (generated in a MathCad™ Professional 7.0 spreadsheet). The two history files, const1.hst and const2.hst, are shown below:

const1.hst

time (yrs)	Temp. (°C)	RH
1.0	95.0	0.95
1000.0	95.0	0.95
10000.0	95.0	0.95
100000.0	95.0	0.95
1000000.0	95.0	0.95

const2.hst

time (yrs)	Temp. (°C)	RH
1.0	95.0	0.85
1000.0	95.0	0.85
10000.0	95.0	0.85
100000.0	95.0	0.85
1000000.0	95.0	0.85

These histories differ only in their RH values and were created to be used with input files whose RH threshold for humid-air CAM corrosion are below 0.85 and whose RH threshold for aqueous CAM corrosion are below 0.95 but above 0.85. In this way, waste packages modeled using const1.hst will experience aqueous CAM corrosion and those modeled using const2.hst will experience humid-air CAM corrosion.

The table below shows the input file options exercised.

Test	CAM Model	CRM Model	Drips
const1.inp	CAMGeneral+PitMultiples	CRMGeneralOnly	none
const1.cerm.inp	CAMGeneral+PitMultiples	CRMGeneralOnly	none
const2.inp	CAMGeneral+PitMultiples	CRMGeneral+PitGrowthLaw, B=0	neutral
const3.inp	CAMGeneral+PitMultiples	CRMGeneral+PitGrowthLaw, B=0	chemical
const4.inp	CAMGeneral+PitMultiples	CRMGeneral+LogPitGrowth	neutral
const5.inp	CAMGeneral+PitMultiples	CRMGeneral+LogPitGrowth	chemical
const6.inp	CAMGeneral+PitMultiples	CRMGeneral+PitGrowthLaw	neutral
const7.inp	CAMGeneral+PitMultiples	CRMGeneral+ PitGrowthLaw	chemical
const8.inp	CAMGeneral+PitMultiples	CRMExponentialPitRate	none
const9.inp	CAMGeneral+PitGrowthLaw	CRMGeneral+LogPitGrowth	chemical
const10.inp	CAMGeneral+PitMultiples	CRMGeneral+ArrheniusPit	chemical
const11.inp	CAMGeneral+PitMultiples	CRMGenRate+ArrheniusPit	neutral

The input file const1.cerm.inp tests the ceramic protection sensitivity option through the use of a constant ceramic delay time of 50 years. In addition to const1.hst and const2.hst, const11.inp uses two constant "CDFs" (50.cdf and 100.cdf) for the CRM general corrosion rate at 50 and 100°C, as shown below:

100.cdf

2	
5.e-4	0.
5.e-4	1.

50.cdf

2	
5.e-5	0.
5.e-5	1.

where the 2 denotes the two rows of the CDF table that follows and the 0th percentile and 100th percentiles are equal for each CDF at 5.0E-4 mm/yr and 5.0E-5 mm/yr for 100°C and 50°C respectively.

4.1 Description of Test Results

The 'raw' output from WAPDEG consists of five files: a *.out file, a *.pat file, a *.bin file, a *.crm file, and a *.aux file (where * = the input file (*.inp) name prefix). Several examples of *.inp and *.out files are included in Section 6.3. Examples of all types of raw output files are included in electronic format (MI: 30048-M04-001). The *.out file contains the input file as a header and then columns of data relevant to the degradation of each package simulated. The table below elucidates the contents of each column in the *.out file:

Column	Contents
1	Package number simulated
2	Advection start time, i.e., the maximum of the earliest patch failure time amongst the top patches which are dripped upon and amongst bottom patches that are dripped upon or not. The advection time corresponds to the earliest time at which water can enter the waste package (from the top), interact with the waste form, and then leave the waste package (from the bottom) to enter the near field environment.
3	Diffusion start time, i.e., the earliest penetration (of any kind) time in the bottom of the waste package.
4	Structural failure time. Interpolated time when the CRM average general corrosion depth (over all patches in a package) surpasses the user supplied structural failure percentage of the total CRM thickness.
5	First penetration (of any type) time for the CAM outer barrier.
6	First penetration (of any type) time for the CRM inner barrier.
7	First pit penetration time (years) of the top of the CAM.
8	First pit penetration time (years) of the side of the CAM.
9	First pit penetration time (years) of the bottom of the CAM.
10	First not-dripped-on patch penetration time (years) of the top of the CRM.
11	First not-dripped-on patch penetration time (years) of the side of the CRM.
12	First not-dripped-on patch penetration time (years) of the bottom of the CRM.
13	First dripped-on pit penetration time (years) of the top of the CRM.
14	First dripped-on pit penetration time (years) of the bottom of the CRM.
15	First dripped-on patch penetration time (years) of the top of the CRM.
16	First dripped-on patch penetration time (years) of the bottom of the CRM.
17	Cumulative pit penetrations of the CRM (at any location) at user supplied time 1.
18	Cumulative pit penetrations of the CRM (at any location) at user supplied time 2.

Column	Contents
19	Cumulative pit penetrations of the CRM (at any location) at user supplied time 3.
20	Cumulative pit penetrations of the CRM (at any location) at user supplied time 4.
21	Cumulative patch penetrations of the CRM (at any location) at user supplied time 1.
22	Cumulative patch penetrations of the CRM (at any location) at user supplied time 2.
23	Cumulative patch penetrations of the CRM (at any location) at user supplied time 3.
24	Cumulative patch penetrations of the CRM (at any location) at user supplied time 4.

Before discussing the patch and bin files, it is instructional to consider how WAPDEG 'bins' temporal data. WAPDEG does not report precise penetration times (beyond those already discussed in the out file), rather a user specified (in the input file) number of logarithmically distributed time bins are used for this purpose. The user specifies the start and end times for binning (call them bin0 and binf) and the total number of bins (call it nbin). To determine the bin in which to place a pit or patch that penetrated the package at time t, one should use the following equation:

$$\text{bin number} = \text{trunc} \left[\text{nbin} * \left(\frac{\ln(t) - \ln(\text{bin0})}{\ln(\text{binf}) - \ln(\text{bin0})} \right) + 1 \right]$$

where trunc represents the truncation operator and assures 'bin number' is an integer that ranges from 1 to (nbin + 1). Using this methodology, a time bin array is created for each waste package whose array elements contain the cumulative number of pit (in the bin file) or patch (in the patch file) penetrations. Penetrations added to a given bin have penetration times greater than or equal to the time associated with that bin. After a waste package has been simulated the pit penetration array is appended to the bin (*.bin) file and the patch penetration array is appended to the patch (*.patch) file.

For each waste package, WAPDEG creates two time bin arrays, one for pit penetrations and one for patch penetrations, in which are stored the cumulative number of penetrations with time.

A portion of the patch file (const1.pat) is reproduced below:

```

1      1019      1201 124440.76
0      0      400   401   400
0      0      400   401   400
0      0      400   401   400
0      0      400   401   400
0      0      400   401   400
.      .      .      .      .
.      .      .      .      .
.      .      .      .      .

```

The first line of the patch file consists of four header values; the waste package number that the subsequent lines of output represent; the number of the first time bin that contains a nonzero value, the

total number of time bins, and the first patch breakthrough time. The subsequent lines of output are composed of five columns:

Column	Contents
1	Cumulative patch penetrations for dripped-on top patches.
2	Cumulative patch penetrations for dripped-on bottom patches.
3	Cumulative patch penetrations for not-dripped-on top patches.
4	Cumulative patch penetrations for not-dripped-on side patches.
5	Cumulative patch penetrations for not-dripped-on bottom patches.

These 'subsequent lines of output' are followed immediately by the four header values for the second package and so on until the final package's results are reported.

The bin file (const1.bin) is quite similar to the patch file in header format, however, since the current conceptual model does not allow for pit growth in the absence of drips and waste package sides are never dripped upon, the bin file's 'subsequent lines of output' contain only two columns reporting cumulative pit penetrations of the top and bottom surfaces, i.e.:

```

1      1019      1201  124440.76
0      0
0      0
0      0
0      0
0      0
.      .
.      .
.      .

```

The CRM file (const1.crm) reports the simulation time and CRM general corrosion depth at 5% increments of CRM general corrosion depth. The header line for each package consists of two values; the package number and the number of subsequent output lines. The header line is then followed by subsequent output lines consisting of the time (years) and depth fraction of CRM general corrosion (i.e., the fraction of the CRM thickness has been corroded) and then the header for the next package and so on., i.e.:

```

1      21
96158.5  0.000
97555.1  0.050
98972.1  0.100
100102.6 0.150
100465.4 0.200
100829.6 0.250
.
.

```


The aux file (const1.aux) consists of any warnings generated and defaults applied during the simulation followed by a detailed listing of properties that were sampled for each package such as the temperature threshold, RH thresholds, delay times, inner barrier time exponent, transition time, and corrosion coefficients. The header for the aux file contains all of the user supplied comments that appear before the 'START OF PARAMETERS' in the input file. const1.aux is shown below:

Const1.hst causes aqueous CAM corrosion
 Const2.hst causes humid air CAM corrosion
 CRM General corrosion only

Defaults and Warning Messages

The number of top, bottom, and side patches per package are
 400, 400, 401, respectively

Package Properties

Package Properties														
Pack	No Drips			Humid-Air Coefficients			Aqueous Coefficients							
No	TmpThr	HAStrt	AqStrt	CRMLnA	CRMk	a0	a1	a2	a3	b0	b1	b2	b3	
1	373.15	80.00	90.00	6.300	4990.0	17.185	-623.46	-974.46	0.62270	111.53	0.53199	-23291.	-3.192E-04	
2	373.15	80.00	90.00	6.300	4990.0	17.185	-623.46	-974.46	0.62270	111.53	0.53199	-23291.	-3.192E-04	
3	373.15	80.00	90.00	6.300	4990.0	17.185	-623.46	-974.46	0.62270	111.53	0.53199	-23291.	-3.192E-04	
4	373.15	80.00	90.00	6.300	4990.0	17.185	-623.46	-974.46	0.62270	111.53	0.53199	-23291.	-3.192E-04	
5	373.15	80.00	90.00	6.300	4990.0	17.185	-623.46	-974.46	0.62270	111.53	0.53199	-23291.	-3.192E-04	
6	373.15	80.00	90.00	6.300	4990.0	17.185	-623.46	-974.46	0.62270	111.53	0.53199	-23291.	-3.192E-04	

4.2 Range of Input Parameter Values Over which Results were Verified

WAPDEG has been run with history files with temperatures varying from 240°C down to room temperatures and with relative humidities from 100% down to 18%, and for time periods ranging up to 1,000,000 years. While there exists little data over these time ranges with which to compare, for short time periods (hours to ~16 years) and low temperature ranges (20 - 90°C) much observed experimental data on the corrosion of carbon steels has been directly incorporated into WAPDEG in the definitions of the corrosion parameters, thus, by definition, WAPDEG agrees with observed experimental results over these time periods. The degradation models and parameters used to simulate corrosion of the inner barrier are based either on elicitations from world-renowned experts (M&O, 1998) or analysis of existing short-term data (Pasupathi, 1997). Thus, results from long-term WAPDEG simulations can be expected to be reasonably accurate given the complexity of the processes modeled.

4.3 Identification of Limitations on Software Routine or Validity

4.3.1 As noted above, one can only be strictly assured of the validity of WAPDEG results for times ranging up to 16 years and for temperatures between 20 and 90°C. One should realize that WAPDEG is unique, in that its entire purpose is to extrapolate corrosion behavior over time ranges inaccessible to mortal experimentalists. The WAPDEG developers are unaware of any strenuously verified and validated software with which WAPDEG results may be compared.

- 4.3.2 WAPDEG is limited by design to temperatures greater than 0°C and relative humidities greater than 0 and less than 100%. History (*.hst) files must be less than 151 lines, there may be no more than 3100 pits/patch, and no more than 800 packages/simulation (in the current version). All of these parameters are in the parm306.inc file and can be changed if larger values are required.
- 4.3.3 As WAPDEG makes use of relatively unsophisticated (fast) interpolation schemes to solve for quantities such as pit breakthrough times, the accuracy of the interpolation is dependent on the granularity of the time grid supplied. As a general rule of thumb, one should use about 4 logarithmically distributed time points in every decade represented.
- 4.3.4 WAPDEG requires the CRM general corrosion models to be identical in the [No Drip Model, CRM], [Neutral Drip Model, CRM], and [Chemical Drip Model, CRM] input segments.
- 4.3.5 Localization factor truncation values are not allowed to be more or less than 5 standard deviations from the mean of the localization factor distribution.
- 4.3.6 The CRMExponentialPitRate model, which was used in the TSPA-1995 analysis, is not implemented properly in this version of the WAPDEG code. Previous versions of the WAPDEG code that include this model have been baselined. Future versions of the code will properly implement the CRMExponentialPitRate model.
- 4.3.7 This Software Routine Report (SRR) is meant to document the models and features available within the WAPDEG code for initial use in the Management and Operations (M&O) Quality Assurance (QA) Program. An associated Software Qualification Report (SQR) will be performed when all the data necessary for complete Verification and Validation (V&V) becomes available.

5.0 Reference List of All Documentation Relevant to the Qualification

Andrews, R.W.; Dale, T.F.; and McNeish, J.A. 1994. *Total System Performance Assessment-1993: An Evaluation of the Potential Yucca Mountain Repository*. B00000000-01717-2200-00099-Rev. 01. Las Vegas, Nevada: Civilian Radioactive Waste Management System, Management and Operating Contractor. NNA.19940406.0158.

ASTM 1992. "Standard Practice for Conducting Atmospheric Corrosion Tests on Metals." ASTM G50-76. *Annual Book of ASTM Standards*, 03.02. Philadelphia, Pennsylvania: ASTM.

Brasher, D.M. and Mercer, A.D. 1968. "Comparative Study of Factors Influencing the Action of Corrosion Inhibitors for Mild Steel in Neutral Solution. I. Sodium Benzoate." *British Corrosion Journal*, 3, 121-129.

Benton, H.A. 1998. *QAP 3-12: Design Input Transmittal, Item 1: Waste Package Materials Selection Analysis*, February 3.

Coburn, S.K. 1978. "Corrosion in Fresh Water: Properties and Selection: Irons and Steels." *Metals Handbook*, Ninth Edition, 1, 733-738. Metal Parks, Ohio: American Society for Metals. 223362.

Farmer, J.C.: 1997. *Crevice Corrosion and Pitting of High-Level Waste Containers: The Integration of Deterministic and Probabilistic Models (II)*, LLNL, UCRL-JC-127980-PT-2, November. 231813.

Fontana, M.G. 1986. *Corrosion Engineering*, 3rd Edition. New York: McGraw-Hill.

Fyfe, D. 1994. "The Atmosphere." *Corrosion, Vol.1 - Metal/Environment Reactions*, 3rd Edition, L. L. Shreir, R. A. Jarman and G. T. Burstein (eds.), 2:31-2:42. Butterworth-Heinemann. 211015.

Haynie, F.H.; Spence, J.W.; and Upham, J.B. 1978. "Effects of Air Pollutants on Weathering Steel and Galvanized Steel: A Chamber Study." *Atmospheric Factors Affecting the Corrosion of Engineering Metals*, ASTM STP 646, S.K. Coburn (ed.), 30-47. American Society for Testing and Materials.

Lee, J. H., J. E. Atkins, and B. Dunlap, 1997. "Incorporation of "Corrosion-Time" and Effects of Corrosion-Product Spalling in Waste Package Degradation Simulation in the Potential Repository at Yucca Mountain," *Scientific Basis for Nuclear Waste Management XX*, Materials Research Society Proceedings, Vol. 465, Ed. W. Gray and I. Triay, Materials Research Society, Pittsburgh, PA (in print).

Lee, J.H.; Atkins, J.E.; and Andrews, R.W. 1996a. "Humid-Air and Aqueous Corrosion Models for Corrosion-Allowance Barrier Material." *Scientific Basis for Nuclear Waste Management XIX, Materials Research Society Symposium Proceedings*, 412, 571, W.M. Murphy, and D.A. Knecht (eds.). Pittsburgh, Pennsylvania: Materials Research Society. MOL.19960816.0028.

Lee, J.H.; Atkins, J.E.; and Andrews, R.W. 1996b. "Stochastic Simulation of Pitting Degradation of Multi-Barrier Waste Container in the Potential Repository at Yucca Mountain," *Scientific Basis for Nuclear Waste Management XIX, Materials Research Society Symposium Proceedings*, 412, 603, W.M. Murphy, and D.A. Knecht (eds.). Pittsburgh, Pennsylvania: Materials Research Society. MOL.19960816.0026.

M&O (Civilian Radioactive Waste Management System, Management and Operating Contractor [CRWMS M&O]) 1998. *Final Report on Waste Package Degradation Expert Elicitation Project*. Rev. 1.0. Las Vegas, Nevada: CRWMS M&O.

M&O (Civilian Radioactive Waste Management System, Management and Operating Contractor [CRWMS M&O]) 1995. "Total System Performance Assessment-1995: An Evaluation of the Potential Yucca Mountain Repository." B00000000-01717-2200-00136, Rev. 01. Las Vegas, Nevada: CRWMS M&O. MOL.19960724.0188.

Marsh, G.P. and Taylor, K.J. 1988. "An Assessment of Carbon Steel Containers for Radioactive Waste Disposal." *Corrosion Science*, 28, 289-320. 223393.

Marsh, G.P.; Taylor, K.J.; and Sooi, Z. 1988. *The Kinetics of Pitting Corrosion of Carbon Steel*. SKB Technical Report 88-09. Stockholm, Sweden: Authors. NNA.19891026.0015.

McCoy, J. K., 1997. "Application of CRM Degradation Models," CRWMS M&O (Civilian Radioactive Waste Management System, Management and Operating Contractor) Interoffice Correspondence, LV.WP.JKM.12/97-248, December 2. MOY-971205-13

McCright, R.D.: 1997. *Engineered Materials Characterization Report for the Yucca Mountain Site Characterization Project. Vol. 3 - Corrosion Data and Modeling*, Rev. 1, LLNL, UCRL-ID-119564.

Mercer, A.D.; Jenkins, I.R.; and Rhoades-Brown, J.E. 1968. "Comparative Study of Factors Influencing the Action of Corrosion Inhibitors for Mild Steel in Neutral Solution. III. Sodium Nitrite." *British Corrosion Journal*, 3, 136-144.

Nagle, R.K. and Saff, E.B. 1993. *Fundamentals of Differential Equations and Boundary Value Problems*. Reading, Massachusetts: Addison-Wesley Pub. Co.

Pasupathi, V. 1997. "CRM Degradation Models-Update," CRWMS M&O (Civilian Radioactive Waste Management System, Management and Operating Contractor) Interoffice Correspondence, LV.WP.VP.12/97-268, December 22. MOY-971231-11

Phipps, P.B. and Rice, D.W. 1979. "The Role of Water in Atmospheric Corrosion." *Corrosion Chemistry*, 235-261, G. R. Brubaker, and P. B. Phipps (eds.), ACS Symposium Series 89, American Chemical Society. 4764.

Pigford, T.H. 1993. "The Engineered Barrier System: Performance Issues." *Scientific Basis for Nuclear Waste Management XVI, Materials Research Society Symposium Proceedings*, 294, 657-662. C.G. Interrante and R.T. Pabalan (eds.). Pittsburgh, Pennsylvania: Materials Research Society.

Ruijini, G.; Srivastava, S.C.; and Ives, M.B. 1989. "Pitting Corrosion Behavior of UNS No. 8904 Stainless Steel in a Chloride/Sulfate Solution." *Corrosion*, 45, No. 11, 874-882.

Southwell, C.R. and Alexander, A.L. 1970. "Corrosion of Metals in Tropical Waters. Structural Ferrous Metals." *Materials Protection*, 14-23. 223416.

Southwell, C.R.; Bultman, J.D.; and Alexander, A.L. 1976. "Corrosion of Metals in Tropical Environments - Final Report of 16-year Exposures." *Materials Performance*, 15, No. 7, 9-26. 224022.

Southwell, C.R. and Bultman, J.D. 1982. "Atmospheric Corrosion Testing in the Tropics." *Atmospheric Corrosion*, 943-967. W.H. Ailor (ed.). Wiley-Interscience. NNA.19890919.0304.

Stahl, D. 1997. "CRM Degradation Models." CRWMS M&O (Civilian Radioactive Waste Management System, Management and Operating Contractor) Interoffice Correspondence, LV.WP.DS.11/97-247, November 26. MOY-971205-13

Strutt, J.E.; Nichols, J.R.; and Barbier, B. 1985. "The Prediction of Corrosion by Statistical Analysis of Corrosion Profiles." *Corrosion Science*, 25, 305-315.

Vernon, W.H.J. 1933. "The Role of the Corrosion Product in the Atmospheric Corrosion of Iron." *Transactions of the Electrochemical Society*, 64, 31-41.

6.0 Listings

6.1 Directory Listing of Executable and Data Files

parm306	inc	4,064	02-26-98	2:07p	parm306.inc
wap306	exe	421,376	02-28-98	1:43p	wap306.exe
wap306	f	335,560	02-28-98	1:43p	wap306.f
wap306	inc	16,585	02-26-98	2:06p	wap306.inc

6.2 Computer Listing of Source Code

6.2.1 Parm306.inc

```
c FILE: PARM306.INC
c February, 1998 version 3.06 WAPDEG
c
c User Adustable Parameters
c
```

```

c  MAXHIST      :Maximum number of alternate temp, RH histories
c  MAXDAT       :Array size for input history data
c  MAXDAT1      :Array size for adjusted history data
c  MAXPITCAM    :Array size for pits per patch on the outer barrier
c  MAXPITCRM    :Array size for pits per patch on the inner barrier
c  MAXPIT       :The MAXIMUM of MAXPITCAM and MAXPITCRM
c  NCOEFF       :Array size for model coefficients
c  NMODEL       :Array size for models
c  MAXBIN       :Array size for time bins
c  MAXCRMT      :Array size for CRM thickness calculations
c  NSEED        :Number of random seeds needed
c  MAXPACK      :Maximum number of packages simulated
c  NAMELEN      :Max string length for file names
c  MAXSEG       :Max number of time segments that can occur
c  NCMODELS     :Number of corrosion models available
c  CAMGPM       :Model number for CAM - General + Pit Multipliers
c  CAMGPGL      :Model number for CAM - General + Pit Growth Law
c  CRMGONLY     :Model number for CRM - General Arrhenius corrosion only
c  CRMGRONLY    :Model number for CRM - General Rates for corrosion only
c  CRMGPGL      :Model number for CRM - General + Pit Growth Law
c  CRMGRPGL     :Model number for CRM - GeneralRate + Pit Growth Law
c  CRMGLOG      :Model number for CRM - General + Logarithmic Pit Growth
c  CRMGRLOG     :Model number for CRM - GeneralRate + Logarithmic Pit Growth
c  CRMGLNL      :Model number for CRM - General + LLNL Arrhenius Law
c  CRMGRNLNL    :Model number for CRM - GeneralRate + LLNL Arrhenius Law
c  CRMEXP       :Model number for CRM - Exponential Pit Rate
c  MAXCRMDISTS  :Maximum number of distributions for CRM rates (at temp)
c  MAXDISTS     :Maximum total number of model-specific distributions
c  MAXPAIRS     :Maximum number of entries in all lookup tables used
c  NODISTS      :Number of distribution types available
c  DSTDEF       :Distribution number for Default type
c  DSTFXD       :Distribution number for Fixed type
c  DSTUNI       :Distribution number for Uniform type
c  DSTNOR       :Distribution number for Normal type
c  DSTBNOR      :Distribution number for Bounded Normal type
c  DSTLOGU      :Distribution number for LogUniform type
c  DSTLOGN      :Distribution number for LogNormal type
c  TOPTYPE      :Index for patch indicating it is a top patch
c  SIDETYPE     :Index for patch indicating it is a side patch
c  BOTTYPE      :Index for patch indicating it is a bottom patch

```

```

c
  integer MAXHIST, MAXDAT, MAXDAT1, MAXPITCAM, MAXPITCRM, MAXPIT,
&      NCOEFF, NMODEL, MAXBIN
  integer MAXCRMT, NSEED, MAXPACK, NAMELEN, MAXSEG
  integer NCMODELS, CAMGPM, CAMGPGL, CRMGONLY, CRMGPGL, CRMGLOG,
&      CRMGLNL, CRMGRONLY, CRMGRPGL, CRMGRLOG, CRMGRNLNL, CRMEXP
  integer MAXCRMDISTS, MAXDISTS, MAXPAIRS
  integer NODISTS, DSTDEF, DSTFXD, DSTUNI, DSTNOR, DSTBNOR, DSTLOGU,
&      DSTLOGN, DSTCDF

```

integer TOPTYPE, SIDETYPE, BOTTYPE

c

```

PARAMETER (MAXHIST = 30)
PARAMETER (MAXDAT = 200)
PARAMETER (MAXDAT1 = MAXDAT + 20)
PARAMETER (MAXPITCAM = 3100)
PARAMETER (MAXPITCRM = 3100)
PARAMETER (MAXPIT = 3100) ! Must be MAX of previous two MAXPITS
PARAMETER (NCOEFF = 10)
PARAMETER (NMODEL = 2)
PARAMETER (MAXBIN = 1301)
PARAMETER (MAXCRMT = 1000)
PARAMETER (NSEED = 4)
PARAMETER (MAXPACK = 800)
PARAMETER (NAMELEN = 30)
PARAMETER (MAXSEG = 4)
PARAMETER (NCMODELS = 11)
PARAMETER (CAMGPM = 1, CAMGPGL = 2, CRMGONLY = 3, CRMGRONLY = 4,
& CRMGPGL = 5, CRMGRPGL = 6, CRMGLOG = 7, CRMGRLOG = 8,
& CRMGLNL = 9, CRMGRNLNL = 10, CRMEXP = 11)
PARAMETER (MAXCRMDISTS = 3, MAXDISTS = 30, MAXPAIRS = 100000)
PARAMETER (NODISTS = 8)
PARAMETER (DSTDEF = 1, DSTFXD = 2, DSTUNI = 3, DSTNOR = 4,
& DSTLOGU = 5, DSTLOGN = 6, DSTCDF = 7, DSTBNOR = 8)
PARAMETER (TOPTYPE = 1, SIDETYPE = 2, BOTTYPE = 3)

```

6.2.2 Wap306.inc

```

c FILE: wap306.inc
c February, 1998 version 3.06 WAPDEG
c
c Array dimensioning parameters appear in include file parm306.inc
c
c Universal variables - hardwired in code
c
c Version :Version identifier for current code (defined below)
c inpfid :Device ID number for the input file
c hstfid :Device ID number for the environment history file
c outfid :Device ID number for the primary output file
c binfid :Device ID number for the pit history file
c auxfid :Device ID number for the auxilliary output file
c patfid :Device ID number for the patch failure history file
c crmfid :Device ID number for the crm structural failure history file
c Mean() :Array of mean values of parameters for corrosion models
c v5(,) :Decomposed covariance matrix for packages and patches
c StanDev() :Array of standard deviations for corrosion models
c v9 :Variance share for patches and pits (TSPA95)

```

```
c  c10          :Pitting term
c
c  Universal variables - obtained while reading input file
c
c  hstfl()       :History file names (string length defined in parm306.inc)
c  SigTmp()      :Temperature standard deviations for each history file
c  SigRH()       :RH standard deviations for each history file
c  Penet1        :Thickness of outer barrier (micrometers)
c  Penet2        :Thickness of inner barrier (millimeters)
c  InFail        :Structural failure when mean CRM thickness is this (mm)
c  NumPack       :Number of waste packages simulated
c  NumPatch      :Number of patches per waste package simulated
c  NPitCAM       :Number of pits per patch simulated on the outer barrier
c  NPitCRM       :Number of pits per patch simulated on the inner barrier
c  PakGrp()      :Number of packages per (history file) group
c  bin0          :Start time for first bin location
c  binf          :Time for final bin location
c  nbins         :Number of bins used for tracking breakthrough times
c  BinFact       :Factor for determining bin location
c  BinTerm       :Term to expedite binning
c  ibin1         :Number of bin containing first print time
c  ibin2         :Number of bin containing second print time
c  ibin3         :Number of bin containing third print time
c  ibin4         :Number of bin containing fourth print time
c  iseed         :Seed for the random number generator
c  TmpAllow      :Max temperature change allowed over a time step
c  RHAllow       :Max relative humidity change allowed over a time step
c  NumTop        :Number of patches designated as top patches
c  TopDripFrc    :Fraction of top patches that are dripped on
c  NumBot        :Number of patches designated as bottom patches
c  BotDripFrc    :Fraction of bottom patches that see pooled water
c  NumSide       :Number of patches designated as side patches
c  DripStartType :Distribution type for describing when dripping starts
c  DripStartPar() :Distribution parameters for DripStartType
c  DripEndType   :Distribution type for describing when dripping stops
c  DripEndPar()  :Distribution parameters for DripEndType
c  SomeDrips     :True if nonzero probability of drips, else false
c  NewDrips      :True if drip character could change, else false
c  HaveNeut      :True if some drips are like connate water, else false
c  HaveChem      :True if some drips are modified water, else false
c  DripChnType   :Distribution type for describing when drip water changes
c  DripChnPar    :Distribution parameters for DripChnType
c  NeutD1        :True if initial drips are of connate water
c  NeutD2        :True if new drips are of connate water
c  CermType      :Distribution type for ceramic protection
c  CermPar       :Distribution parameters for CermType
c  MSpec()       :Model numbers for CAM and CRM for each time segment
c  DTypeStack()  :Distribution type for all model-specific distributions
c  DParmStack()  :Distribution parameters for all DTypeStack distributions
```



```

c EndStack      :Points to where distributions stored in D*Stack arrays
c CDFStack()    :Stores all lookup tables used for the simulation
c CRMMax()      :Maximum CRM general rates allowed
c NumCRMD()     :Number of dists for CRM corrosion rate for each condition
c varshare      :Share of variance for packages (CAM and CRM)
c vg()          :Covariance decomposition for CRM general rate
c NDFeat(1)     :Correlation between humidair&aqueous dists when no dripping
c NDFeat(2)     :Galvanic protection depth, when no dripping
c NDFeat(3)     :Fraction of patches galvanically protected, when no dripping
c NDFeat(4)     :Spalling depth, when no dripping
c NDFeat(5)     :Scaling factor for general CAM corrosion rate, when no drip
c NDFeat(6)     :Scaling factor for general CRM corrosion rate, when no drip
c GDFeat(1)     :Correlation between humidair&aqueous dists, connate drips
c GDFeat(2)     :Galvanic protection depth, connate drips
c GDFeat(3)     :Fraction of patches galvanically protected, connate drips
c GDFeat(4)     :Spalling depth, connate drips
c GDFeat(5)     :Scaling factor for general CAM corrosion rate, connate drips
c GDFeat(6)     :Scaling factor for general CRM corrosion rate, connate drips
c BDFeat(1)     :Correlation between humidair&aqueous dists, tainted drips
c BDFeat(2)     :Galvanic protection depth, tainted drips
c BDFeat(3)     :Fraction of patches galvanically protected, tainted drips
c BDFeat(4)     :Spalling depth, tainted drips
c BDFeat(5)     :Scaling factor for general CAM corrosion rate, tainted drips
c BDFeat(6)     :Scaling factor for general CRM corrosion rate, tainted drips
c ErrFound      :Error found while reading input (or later errors)
c
c Universal variables - obtained while reading history files
c
c NumStep       :Number of time steps in simulation
c tim()         :Time step boundaries from the history file
c tem()         :Temperatures from the history file
c r()           :Relative humidities from the history file
c TmpMx         :Maximum temperature on history file
c TmpMn         :Minimum temperature on history file
c RHMax         :Maximum RH on history file
c RHMin         :Minimum RH on history file
c
c Waste package specific variables
c
c Pack          :Index for current waste package
c GroupEnd      :Last package index for the current group
c GroupNdx      :Group index number for the current group
c SeedPacks(,)  :Array of integer seeds, one for each package
c Pit1st()      :First pit penetration times (CAM and CRM)
c Pat1st()      :First patch failure times (CRM)
c BinStart      :When first pit penetrates the inner barrier
c Bin()         :Record of cumulative pit penetrations (top, side, bottom)
c PatStart      :When first patch fails
c PatBin()      :Record of cumulative patch failures (top, side, bottom)

```

```

c  CRMThk()      :Thickness of CRM for a set of time indexes
c  C9            :Mean value of C0 for the package
c  NDPakLnA     :Mean log(A) term for corrosion rate (CRM), no drips
c  NDPakK       :Mean K term for corrosion rate (CRM), no drips
c  NDCRMT       :Temps at which CRM rate distributions given, no drips
c  NDPakR       :Untransformed CRM corrosion rate for pack, no drips
c  NDTpVal      :Temperature threshold CAM, no drips
c  NDHStart     :RH threshold for humid-air initiation, no drips
c  NDAquaX      :RH threshold for aqueous initiation, no drips
c  GDPakLnA     :Mean log(A) term for corrosion rate (CRM), connate drips
c  GDPakK       :Mean K term for corrosion rate (CRM), connate drips
c  GDCRMT       :Temps at which CRM rate distributions given, connate drips
c  GDPakR       :Untransformed CRM corrosion rate for pack, connate drips
c  GDTpVal      :Temperature threshold CAM, connate drips
c  GDPInit      :Temperature to initiate pits CRM, connate drips
c  GDHStart     :RH threshold for humid-air initiation, connate drips
c  GDAquaX      :RH threshold for aqueous initiation, connate drips
c  BDPakLnA     :Mean log(A) term for corrosion rate (CRM), tainted drips
c  BDPakK       :Mean K term for corrosion rate (CRM), tainted drips
c  BDCRMT       :Temps at which CRM rate distributions given, tainted drips
c  BDPakR       :Untransformed CRM corrosion rate for pack, tainted drips
c  BDTpVal      :Temperature threshold CAM, tainted drips
c  BDPInit      :Temperature to initiate pits CRM, tainted drips
c  BDHStart     :RH threshold for humid-air initiation, tainted drips
c  BDAquaX      :RH threshold for aqueous initiation, tainted drips
c  OnsetDrip    :Time that drips begin on the package
c  NdxDrip      :Time step index for OnsetDrip, after it is inserted
c  ChangeDrip   :Time that dripping water changes character
c  NdxChn       :Time step index for ChangeDrip, after it is inserted
c  StopDrip     :Time that dripping stops for the package
c  NdxEnd       :Time step index for StopDrip, after it is inserted
c  PwrIB        :Power of time for inner barrier corrosion
c  Coef1()      :Sampled corrosion model coefficients for waste package
c  NDNowVal     :Number of adjusted time steps, no drip patches
c  NDntim()     :Time step boundaries, no drip patches
c  NDntem()     :Temperature history, no drip patches
c  NDnr()       :RH history, no drip patches
c  YDNowVal     :Number of adjusted time steps, dripped/pooled patches
c  YDntim()     :Time step boundaries, dripped/pooled patches
c  YDntem()     :Temperature history, dripped/pooled patches
c  YDnr()       :RH history, dripped/pooled patches
c  NDTime       :True if time available for nondripped on patches
c  YDTimeND1    :True if time prior drips for dripped on patches
c  YDTimeND2    :True if time after drips for dripped on patches
c  YDTime1      :True if time for initial drips for dripped on patches
c  YDTime2      :True if time for modified drips for dripped on patches
c  YDTime       :True if any of YDTime* are true
c
c  Patch specific variables

```

```
c
c Patch           :Index for current patch
c PatchType       :Type of patch (top, side, bottom)
c GPDepth()       :Galvanic protection depth in each time segment
c Coef2()         :Conditional sampled corrosion coefficients for patch
c Start          :When corrosion initiates in outer barrier
c C0PatM          :Mean value for C0 for the patch
c dt()           :Time step sizes for selected time grid for patch
c NumSeg         :# time segments that must be modeled
c SegStart        :Starting indexes for each segment
c SegEnd          :Ending indexes for each segment
c Ndx1           :Time index when dripping starts for the patch
c Ndx2           :Time index when water chemistry changes for the patch
c Ndx3           :Time index when dripping ends for the patch
c DoPatch        :Simulate patch if true (time for corrosion to occur)
c CAMType()       :Model type for the CAM in each segment
c CRMTType()      :Model type for the CRM in each segment
c PitInit()       :Pit initiation temperature for each segment
c CRMGen()        :General corrosion rate for inner barrier
c GenRateM()      :Multipliers for general corrosion rates for each segment
c MxCRMRate()     :Maximum general CRM corrosion rate for each segment
c CRMGenR()       :CRM rates at given temperatures for each segment
c CRMTemp()       :Given set of temperatures for each segment
c CRMNoD()        :Number of temps given for each segment
c
c Pit specific variables
c
c NDCritDep()     :Corrosion depths for pit penetration, no drips
c GDCritDep()     :Corrosion depths for pit penetration, connate drips
c BDCritDep()     :Corrosion depths for pit penetration, bad drips
c MaxPM()         :Maximum pit multiplier on CAM for each segment
c C0Pak()         :C0 values (Exp[C0 - PakMean]) for current patch
c BDCAMB()        :Pit growth law constant for CAM, tainted drips
c BDCAMn()        :Pit growth law exponent for CAM, tainted drips
c GDCRM1()        :Pit growth law parameter for CRM, connate drips
c GDCRM2()        :Pit growth law parameter for CRM, connate drips
c GDCRM3()        :Pit growth law parameter for CRM, connate drips
c BDCRM1()        :Pit growth law parameter for CRM, tainted drips
c BDCRM2()        :Pit growth law parameter for CRM, tainted drips
c BDCRM3()        :Pit growth law parameter for CRM, tainted drips
c
c Temporary variables
c
c NowVal          :Number of adjusted time steps
c ntim()          :Time step boundaries
c ntem()          :Temperature history
c nr()            :RH history
c CAMPit()        :Pit properties for CAM for all time segments
c CRMPit()        :Pit properties for CRM for all time segments
```

```

c
c*****
c
c Universal variables - hardwired in code
c
  character * 5 Version
  PARAMETER (Version = '3.06 ')
  integer inpfid, hstfid, outfid, binfid, auxfid, patfid, crmfid
  double precision v9, c10
  double precision Mean(NCOEFF), v5(NCOEFF, NCOEFF),
&      StanDev(NMODEL, 2)
c
c Universal variables - obtained while reading input file
c
  character*(NAMELEN) hstfl(MAXHIST)
  logical SomeDrips, NewDrips, HaveNeut, HaveChem, NeutD1, NeutD2
  logical ErrFound
  integer NumPack, NumPatch, NPitCAM, NPitCRM, PakGrp(MAXHIST)
  integer nbin, ibin1, ibin2, ibin3, ibin4, iseed
  integer NumTop, NumBot, NumSide
  integer DripStartType, DripEndType, DripChnType, CermType
  integer MSpec(2 * MAXSEG), EndStack(3 * MAXSEG)
  integer DTypeStack(MAXDISTS), NumCRMD(3)
c
  double precision SigTmp(MAXHIST), SigRH(MAXHIST)
  double precision Penet1, Penet2, InFail
  double precision bin0, binf, BinFact, BinTerm
  double precision TmpAllow, RHallow
  double precision TopDripFrc, BotDripFrc
  double precision DripStartPar(4), DripEndPar(4), DripChnPar(4),
&      CermPar(4)
  double precision DParmStack(MAXDISTS, 4), CDFStack(MAXPAIRS, 2)
  double precision CRMMax(3), varshare, vg(2, 3, 4)
  double precision NDFeat(6), GDFeat(6), BDFeat(6)
c
c Universal variables - obtained while reading history file
c
  integer NumStep
  double precision tim(MAXDAT), tem(MAXDAT), r(MAXDAT)
  double precision TmpMx, TmpMn, RHMax, RHMin
c
c Waste package specific variables
c
  logical NDTime, YDTimeND1, YDTimeND2, YDTime1, YDTime2, YDTime
  integer GroupEnd, GroupNdx
  integer Pack, NdxDrip, NdxChn, NdxEnd, NDNowVal, YDNowVal
  integer SeedPacks(NSEED, MAXPACK)
  integer Bin(MAXBIN, 3), PatBin(MAXBIN, 5)
  double precision Pit1st(5), Pat1st(5)

```

```

double precision BinStart, PatStart, C9, CRMThk(MAXCRMT)
double precision NDPakLnA, NDPakK, NDTpVal, NDHStart, NDAquaX,
& NDCRMT(MAXCRMDISTS), NDPakR(MAXCRMDISTS)
double precision GDPakLnA, GDPakK, GDTpVal, GDPInit, GDHStart,
& GDAquaX, GDCRMT(MAXCRMDISTS), GDPakR(MAXCRMDISTS)
double precision BDPakLnA, BDPakK, BDTpVal, BDPInit, BDHStart,
& BDAquaX, BDCRMT(MAXCRMDISTS), BDPakR(MAXCRMDISTS)
double precision OnsetDrip, ChangeDrip, StopDrip, PwrIB
double precision Coef1(NCOEFF)
double precision NDntim(MAXDAT1), NDntem(MAXDAT1), NDnr(MAXDAT1)
double precision YDntim(MAXDAT1), YDntem(MAXDAT1), YDnr(MAXDAT1)
c
c Patch specific variables
c
    logical DoPatch
    integer Patch, PatchType, NumSeg, Ndx1, Ndx2, Ndx3
    integer SegStart(MAXSEG), SegEnd(MAXSEG)
    integer CAMType(MAXSEG), CRMTType(MAXSEG), CRMNoD(MAXSEG)
    double precision C0PatM, Start, PitInit(MAXSEG),
    & CRMGen(MAXSEG, 2), GenRateM(MAXSEG, 2),
    & MxCRMRate(MAXSEG)
    double precision CRMGenR(MAXSEG, MAXCRMDISTS),
    & CRMTemp(MAXSEG, MAXCRMDISTS)
    double precision GPDepth(MAXSEG)
    double precision Coef2(NCOEFF)
    double precision dt(MAXDAT1)
c
c Pit specific variables
c
    double precision NDCritDep(MAXPITCAM), GDCritDep(MAXPITCAM),
    & BDCritDep(MAXPITCAM), MaxPM(MAXSEG)
    double precision COPak(MAXPITCRM)
    double precision BDCAMB(MAXPITCAM), BDCAMn(MAXPITCAM)
    double precision GDCRM1(MAXPITCRM), GDCRM2(MAXPITCRM),
    & GDCRM3(MAXPITCRM)
    double precision BDCRM1(MAXPITCRM), BDCRM2(MAXPITCRM),
    & BDCRM3(MAXPITCRM)
c
c Temporary variables
c
    integer NowVal
    double precision ntim(MAXDAT), ntem(MAXDAT), nr(MAXDAT)
    double precision CAMPit(MAXPITCAM, MAXSEG, 2),
    & CRMPit(MAXPITCRM, MAXSEG, 3)

```

6.2.3 Wap306.f

```
c      PROGRAM WAPDEG
c
c      Version 3.06  February, 1998
c      For unix, compile by typing: f77 wap306.f -o wap306 -O2
c      For PC-DOS, can use Lahey 90 compiler or Digital Fortran
c
c      Copyright Notice
c      This program has been developed by Dennis Longsine, Kevin Mon,
c      Joon Lee, and Bryan Bullard of Duke Engineering and Services
c      for the Yucca Mountain Site characterization Project of the
c      Civilian Radioactive Waste Management Program.
c
c      Latest changes include
c      1. Allow different number of pits on CAM vs number on CRM.
c      2. Detect when can speed execution by using only worst pit on CAM.
c      3. Fix the calculation that finds surviving pits after patch failure.
c      4. Allow for histories to have 0 (adjusted) packages.
c
c      Include file parm306.inc contains maximum problem size parameters.
c      Include file wap306.inc declares variables for the main program and
c      provides their definitions.
c
c      IMPLICIT NONE
c      INCLUDE 'parm306.inc'
c      INCLUDE 'wap306.inc'
c
c      I/O file device numbers
c
c      inpfid = 14
c      hstfid = 15
c      outfid = 16
c      binfid = 17
c      auxfid = 18
c      patfid = 19
c      crmfid = 20
c
c      Read in general program parameters.  Output files are opened here.
c      The version number and 5 file unit numbers are sent to the routine,
c      all other arguments are returned values.  If an input error detected,
c      close the opened files and exit.
c
c      CALL ReadFileInp(Version, inpfid, outfid, binfid, auxfid, patfid,
c      &                  crmfid, hstfid, SigTmp, SigRH, Penet1, Penet2,
c      &                  InFail, NumPack, NumPatch, NPitCAM, NPitCRM,
c      &                  PakGrp, bin0, binf, nbin, BinFact, BinTerm,
c      &                  ibin1, ibin2, ibin3, ibin4, iseed, TmpAllow,
c      &                  RHallow, NumTop, TopDripFrc, NumBot, BotDripFrc,
c      &                  NumSide, DripStartType, DripStartPar,
c      &                  DripEndType, DripEndPar, SomeDrips, NewDrips,
```

```

&      HaveNeut, HaveChem, DripChnType, DripChnPar,
&      NeutD1, NeutD2, CermType, CermPar, MSpec,
&      DTypeStack, DParmStack, EndStack, CDFStack,
&      CRMMax, NumCRMD, varshare, vg, NDFeat, GDFeat,
&      BDFeat, NDCRMT, GDCRMT, BDCRMT, ErrFound)
  if(ErrFound) then
    CLOSE(inpfid)
    CLOSE(outfid)
    CLOSE(binfid)
    CLOSE(auxfid)
    CLOSE(patfid)
    STOP
  end if

c
c Set means and covariance for general corrosion parameters for the
c outer barrier.
c
  CALL SetModelParameters(Mean, v5, StanDev, varshare)
c
c   v5 = 0.D0
c   StanDev = 0.D0
c
c Fill seed pack array with random integers, a set for each package.
c Return (Seedpacks)
c
  CALL SetSeedPacks(NumPack, iseed, SeedPacks)
c
c .....
c Loop over packages
c
  GroupNdx = 0
  GroupEnd = 0
  do Pack = 1, NumPack
c   write(*, 9000) Pack
c 9000   format(1H+', ' Simulating package # ',i4)
c
c Setup values for the current package
c Return (Pit1st, ... , BDCRMn)
c
    CALL SetupPackage(auxfid, Mean, v5, StanDev, v9, c10, Penet1,
&      NPitCAM, NPitCRM, binf, nbin, DripStartType,
&      DripStartPar, DripEndType, DripEndPar,
&      SomeDrips, NewDrips, HaveNeut, HaveChem,
&      DripChnType, DripChnPar, NeutD1, MSpec,
&      DTypeStack, DParmStack, Endstack, CDFStack,
&      NumCRMD, varshare, vg, Pack, SeedPacks,
&      Pit1st, Pat1st, BinStart, Bin, PatStart,
&      PatBin, CRMThk, c9, NDPakLnA, NDPakK, NDPakR,
&      NDTpVal, NDHASTart, NDAquaX, NDFeat, GDPakLnA,
&      GDPakK, GDPakR, GDTpVal, GDPInit, GDHASTart,

```

```

&          GDAquaX, GDFeat, BDPakLnA, BDPakK, BDPakR,
&          BDTpVal, BDPInit, BDHASStart, BDAquaX, BDFeat,
&          OnsetDrip, ChangeDrip, StopDrip, PwrIB, Coef1,
&          NDCritDep, GDCritDep, BDCritDep, COPak,
&          BDCAMB, BDCAMn, GDCRM1, GDCRM2, GDCRM3,
&          BDCRM1, BDCRM2, BDCRM3)
c
c If this is the first package of a new group, read the history file
c and refine the grid if requested.
c
      if(Pack .gt. GroupEnd) then
        GroupNdx = GroupNdx + 1
        do while (PakGrp(GroupNdx) .eq. 0)
          GroupNdx = GroupNdx + 1
        end do
        GroupEnd = GroupEnd + PakGrp(GroupNdx)
c
c Get RH and temperature history from file named hstfl(GroupNdx).
c Return (NumStep, tim, tem, r, TmpMx, TmpMn, RHMax, RHMin)
c If an input error detected, close the opened files and exit.
c
      CALL ReadFileHst(hstfid, outfild, auxfid, hstfl, binf,
&          ErrFound, NumStep, tim, tem, r, TmpMx, TmpMn,
&          RHMax, RHMin, GroupNdx)
      if(ErrFound) then
        CLOSE(hstfid)
        CLOSE(outfild)
        CLOSE(binfid)
        CLOSE(auxfid)
        CLOSE(patfid)
        STOP
      end if
c
c If maximum allowable change over time step provided, impose it.
c Return (ErrFound, NumStep, tim, tem, r)
c If an input error detected, close the opened files and exit.
c
      CALL ChkTimeGrid(outfild, auxfid, TmpAllow, RHAllow, ErrFound,
&          NumStep, tim, tem, r, TmpMx, TmpMn, RHMax,
&          RHMin)
      if(ErrFound) then
        CLOSE(outfild)
        CLOSE(binfid)
        CLOSE(auxfid)
        CLOSE(patfid)
        STOP
      end if
    end if
c

```



```

c Sample the history for this package based on the mean history for its
c group and the standard deviations. Then develop potentially two
c histories for this package, one for patches never dripped on and one
c for those who see drips.

```

```

c      CALL GetPackGrids(outfid, auxfid, SigTmp, SigRH, binf, NeutD1,
&      NeutD2, ErrFound, NumStep, tim, tem, r, Pack,
&      GroupNdx, NDTpVal, NDHStart, NDAquaX,
&      GDTpVal, GDHStart, GDAquaX, BDTPVal,
&      BDHStart, BDAquaX, OnsetDrip, NdxDrip,
&      ChangeDrip, NdxChn, StopDrip, NdxEnd,
&      NDNowVal, NDntim, NDntem, NDnr, YDNowVal,
&      YDntim, YDntem, YDnr, NDTime, YDTimeND1,
&      YDTimeND2, YDTime1, YDTime2, YDTime)

```

```

c .....
c Loop over patches. For each patch, determine its location and set its
c required properties accordingly.

```

```

c      do Patch = 1, NumPatch
&      if(Patch .le. NumTop) then
&      PatchType = TOPTYPE
&      else if(Patch .le. NumTop + NumSide) then
&      PatchType = SIDETYPE
&      else
&      PatchType = BOTTYPE
&      end if

```

```

c      Setup patch returns (GPDepth, Coef2, CritDep, MultRat, COPak)

```

```

c      CALL SetupPatch(v5, StanDev, v9, Penet1, NPitCAM, NPitCRM,
&      TopDripFrc, BotDripFrc, HaveNeut, NeutD1,
&      NeutD2, CermType, CermPar, MSpec, DTypeStack,
&      DParmStack, EndStack, CDFStack, CRMMMax,
&      NumCRMD, varshare, vg, NDFeat, GDFeat, BDFeat,
&      c9, NDPakLnA, NDPakK, NDCRMT, NDPakR,
&      GDPakLnA, GDPakK, GDCRMT, GDPakR, GDPInit,
&      BDPakLnA, BDPakK, BDCRMT, BDPakR, BDPInit,
&      NdxDrip, NdxChn, NdxEnd, Coef1, NDNowVal,
&      NDntim, NDntem, NDnr, YDNowVal, YDntim,
&      YDntem, YDnr, NDTime, YDTimeND1, YDTimeND2,
&      YDTime1, YDTime2, YDTime, PatchType, GPDepth,
&      Coef2, Start, COPatM, dt, NumSeg, SegStart,
&      SegEnd, Ndx1, Ndx2, Ndx3, DoPatch, CAMType,
&      CRMTType, PitInit, NDCritDep, GDCritDep,
&      BDCritDep, MaxPM, BDCAMB, BDCAMn, GDCRM1,
&      GDCRM2, GDCRM3, BDCRM1, BDCRM2, BDCRM3, COPak,
&      CAMPit, CRMGen, GenRateM, MxCRMRate, CRMGenR,
&      CRMTemp, CRMNoD, CRMPit, NowVal, ntim, ntem,

```

```

      &                                nr)
C
      if(DoPatch) then
C
C   There is time available for corrosion.  If so, set its properties and
C   then model corrosion taking spalling into account or not.
C   Corrosion routines return (Bin, BinStart)
C
      IF (SpallDep .gt. 0.D0) then
C
C         CALL CorrosionYesSP(c10, Penet2, NumPit, BinFact, BinTerm,
C         &                                SpallDep, RMOuter, RMiner, NewPMs,
C         &                                RMOuter2, RMiner2, Outer1st,
C         &                                BinStart, Bin, AquaTrans, PwrIB,
C         &                                XTime, NowVal, Start, ntim, ntem, nr,
C         &                                dt, GPDepth, Coef2, CritDep, MultRat,
C         &                                COPak)
C
      else
C
C         CALL CorrosionNoSP(c10, Penet1, Penet2, NPitCAM, NPitCRM,
C         &                                binf, BinFact, BinTerm, NeutD1, NeutD2,
C         &                                MSpec, Pit1st, Pat1st, BinStart, Bin,
C         &                                PatStart, PatBin, CRMThk, NDTpVal,
C         &                                NDHASTart, NDAquaX, GDTpVal, GDHASTart,
C         &                                GDAquaX, BDTpVal, BDHASTart, BDAquaX,
C         &                                PwrIB, PatchType, GPDepth, Coef2,
C         &                                Start, COPatM, dt, NumSeg, SegStart,
C         &                                SegEnd, Ndx1, Ndx2, Ndx3, CAMType,
C         &                                CRMType, PitInit, MaxPM, CRMGen,
C         &                                GenRateM, MxCRMRate, CRMGenR, CRMTemp,
C         &                                CRMNoD, NowVal, ntim, ntem, nr, CAMPit,
C         &                                CRMPit)
C
      end if
      end if
      end do !End of Patch Loop
C
C   Write the results for the current waste package to the output files.
C
      CALL WritePack(outfid, binfid, patfid, crmfid, Penet2, InFail,
      &                                NumPatch, bin0, binf, nbin, BinFact, ibin1,
      &                                ibin2, ibin3, ibin4, MSpec, Pack, Pit1st, Pat1st,
      &                                BinStart, Bin, PatStart, PatBin, CRMThk)
      end do !End of Pack Loop
C
C   Close the output files and end.
C
      CLOSE(outfid)
      CLOSE(binfid)
      CLOSE(auxfid)
      END !PROGRAM WAPDEG
C

```

```

C *****
C
C      SUBROUTINE ReadFileInp(Version, inpfid, outfid, binfid, auxfid,
&                             patfid, crmfid, hstfl, SigTmp, SigRH,
&                             Penet1, Penet2, InFail, NumPack, NumPatch,
&                             NPitCAM, NPitCRM, PakGrp, bin0, binf, nbin,
&                             BinFact, BinTerm, ibin1, ibin2, ibin3,
&                             ibin4, iseed, TmpAllow, RHallow, NumTop,
&                             TopDripFrc, NumBot, BotDripFrc, NumSide,
&                             DripStartType, DripStartPar, DripEndType,
&                             DripEndPar, SomeDrips, NewDrips, HaveNeut,
&                             HaveChem, DripChnType, DripChnPar, NeutD1,
&                             NeutD2, CermType, CermPar, MSpec,
&                             DTypeStack, DParmStack, EndStack, CDFStack,
&                             CRMMMax, NumCRMD, varshare, vg, NDFeat,
&                             GDFeat, BDFeat, NDCRMT, GDCRMT, BDCRMT,
&                             ErrFound)
C
C      Read primary input file and perform data checks.
C      Detect errors (return with ErrFound = true if any found) and set
C      defaults if necessary (issue warning if any set).
C      Input : Version, inpfid, outfid, binfid, auxfid, patfid, crmfid
C      Output: hstfl, SigTmp, SigRH, Penet1, Penet2, InFail, NumPack,
C              NumPatch, NPitCAM, NPitCRM, PakGrp, bin0, binf, nbin,
C              BinFact, BinTerm, ibin1, ibin2, ibin3, ibin4, iseed,
C              TmpAllow, RHallow, NumTop, TopDripFrc, NumBot, BotDripFrc,
C              NumSide, DripStartType, DripStartPar, DripEndType,
C              DripEndPar, SomeDrips, NewDrips, HaveNeut, HaveChem,
C              DripChnType, DripChnPar, NeutD1, NeutD2, CermType, CermPar,
C              MSpec, DTypeStack, DParmStack, EndStack, CDFStack, CRMMMax,
C              NumCRMD, varshare, vg, NDFeat, GDFeat, BDFeat, NDCRMT,
C              GDCRMT, BDCRMT, ErrFound
C      Local : dum, vers, intstr1, intstr2, realstr, inpflname, outflname,
C              binflname, auxflname, patflname, crmflname, CDFName, Err1,
C              Err2, Warn1, Warn2, testex, OnlyGenCRM, DotPos, intvar, NW,
C              NumHist, PackTot, CDFStackEnd, Cond, ModBranch, i, NumD,
C              DType, StackIt, var1, var2, vard, DParm, RateMax, binrat,
C              Logbin0, bin1, bin2, bin3, bin4, TopAngle, PctTopDrip,
C              BotAngle, PctBotDrip, NCRMD, ATemp
C
C      Arguments, and the problem size parameters from the include file.
C
C      IMPLICIT NONE
C      include 'parm306.inc'
C
C      character Version*5
C      character*(NAMELEN) hstfl(MAXHIST)
C
C      logical SomeDrips, NewDrips, HaveNeut, HaveChem, NeutD1, NeutD2,

```

```

&      ErrFound
c
integer inpfid, outfid, binfid, auxfid, patfid, crmfid
integer NumPack, NumPatch, NPitCAM, NPitCRM, PakGrp(*)
integer izeed, nbin, ibin1, ibin2, ibin3, ibin4
integer NumTop, NumSide, NumBot
integer DripStartType, DripChnType, DripEndType, CermType
integer MSpec(*), DTypeStack(*), EndStack(*), NumCRMD(*)
c
double precision SigTmp(*), SigRH(*)
double precision Penet1, Penet2, InFail
double precision bin0, binf, BinFact, BinTerm
double precision TmpAllow, RHallow
double precision TopDripFrc, BotDripFrc
double precision DripStartPar(*), DripChnPar(*), DripEndPar(*),
&      CermPar(*)
double precision DParmStack(MAXDISTS, 4), CDFStack(MAXPAIRS, 2)
double precision CRMMax(*), varshare, vg(2, 3, 4)
double precision NDFeat(*), GDFeat(*), BDFeat(*)
double precision NDCRMT(*), GDCRMT(*), BDCRMT(*)
c
c Local variables
c Note dimensions for DType and DParm below are adequate as long as
c MAXCRMDISTS is at least 2.
c
character dum*80, vers*5, intstr1*6, intstr2*6, realstr*11
character*(NAMELEN + 4) inpfname, outfname, binfname,
&      auxfname, patfname, crmfname
character*(NAMELEN) CDFName
character*74 Err1, Err2, Warn1, Warn2
logical testex, OnlyGenCRM
integer DotPos, intvar, NW, NumHist, PackTot, CDFStackEnd, Cond,
&      ModBranch, i, NumD, DType(MAXCRMDISTS + 3), StackIt, NCRMD
double precision var1, var2, vard(4), DParm(MAXCRMDISTS + 3, 4),
&      RateMax
double precision binrat, Logbin0, bin1, bin2, bin3, bin4
double precision TopAngle, PctTopDrip, BotAngle, PctBotDrip
double precision ATemp(MAXCRMDISTS)
c
c Inquire for inp file name for processing
c
write(*, *) 'WapDeg Version = ', Version
write(*, *)
write(*, *) 'Name of input file? '
read(*, '(A30)') inpfname
c
c If no input file is specified, default names are chosen for the
c input, output, bin, aux, pat filenames. If only an input file prefix
c or an input file prefix and suffix is entered, the prefix is

```

c identified, and output, bin, and auxilliary files are named with that
c prefix and the appropriate suffix.

```
c
  IF (inpflname(1: 1) .eq. ' ') then
    inpflname = 'wap306.inp'
    outflname = 'wap306.out'
    binflname = 'wap306.bin'
    auxflname = 'wap306.aux'
    patflname = 'wap306.pat'
    crmflname = 'wap306.crm'
    write(*, *) 'Default file selected: ',inpflname
  else
    CALL GetDotPosit(inpflname, NAMELEN, DotPos)
    DotPos = DotPos - 1
    inpflname = inpflname(1: DotPos) // '.inp'
    outflname = inpflname(1: DotPos) // '.out'
    binflname = inpflname(1: DotPos) // '.bin'
    auxflname = inpflname(1: DotPos) // '.aux'
    patflname = inpflname(1: DotPos) // '.pat'
    crmflname = inpflname(1: DotPos) // '.crm'
  end if
  inquire(FILE = inpflname, EXIST = testex)
  IF (.not. testex) then
    Err1 = 'ERROR. Input file not available. Program STOP'
    Err2 = 'Name = ' // inpflname
    CALL WriteMessage(Err1, Err2, .true., 0, 0, 0)
    ErrFound = .true.
    RETURN
  else
    ErrFound = .false.
    write(*, *) 'Reading input file: ',inpflname
  end if
  CDFStackEnd = 0
```

c
c Here, the input, output, bin, and auxilliary files are opened. The
c input file will be closed by this subroutine(if there are no errors).
c The other files will closed by the main program.

```
c
  OPEN (inpfid, FILE = inpflname, STATUS = 'OLD')
  OPEN (outfid, FILE = outflname, STATUS = 'UNKNOWN')
  OPEN (binfid, FILE = binflname, STATUS = 'UNKNOWN')
  OPEN (auxfid, FILE = auxflname, STATUS = 'UNKNOWN')
  OPEN (patfid, FILE = patflname, STATUS = 'UNKNOWN')
  OPEN (crmfid, FILE = crmflname, STATUS = 'UNKNOWN')
```

c
c The initial (not program controlled) header statements are copied to
c the output file until the START OF PARAMETERS line is encountered.

```
c
  read(inpfid, 8000) dum
```

```

do while (dum(1: 4) .NE. 'STAR')
  write(outfid, 8000) dum
  write(auxfid, 8000) dum
  read (inpfid, 8000) dum
end do
write(auxfid, 9900)
9900 format('Defaults and Warning Messages',/)
NW = 0
write(outfid, 8001) 'START OF PARAMETERS'

c
c Here the version of the input file is checked against that of the
c version of WapDeg being executed.
c
  read(inpfid, 9000) vers
  write(outfid, 8002) vers
9000 format(A5)
  if(Version .NE. vers) then
    Err1 = 'ERROR. Input file does not match code version. ' //
    &      'Program STOP'
    Err2 = 'Version from input ' // vers // ', from code '
    &      // Version
    CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
    ErrFound = .true.
    RETURN
  end if

c
c Read the number of alternate histories to be input
c
  read(inpfid, *) NumHist
  write(outfid, 80031) NumHist
  if(NumHist .gt. MAXHIST) then
    write(intstr1, '(i6)') MAXHIST
    Err1 = 'ERROR. Too many alternate history files. Max ' //
    &      'allowed = ' // intstr1
    Err2 = 'Program STOP'
    CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
    ErrFound = .true.
    RETURN
  end if

c
c In turn read the name of a history file, ensure its existence, and
c read the standard deviations for temperature and RH.
c
  PackTot = 0
  do i = 1, NumHist
    read(inpfid, '(A30)') hstfl(i)
    write(outfid, 80032) hstfl(i)
    inquire(FILE = hstfl(i), EXIST = testex)
    if(.not. testex) then

```

```
      Err1 = 'ERROR. History file not available. Program STOP'
      Err2 = 'Name = ' // hstfl(i)
      CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
      ErrFound = .true.
      RETURN
    end if

c
    read(inpfid, *) PakGrp(i), SigTmp(i), SigRH(i)
    write(outfid, 80033) PakGrp(i), SigTmp(i), SigRH(i)
    if(PakGrp(i) .lt. 0) then
      Err1 = 'ERROR. Number of packages per group cannot be ' //
&         'negative. Program STOP.'
      Err2 = ' '
      ErrFound = .true.
      CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
      RETURN
    end if
    PackTot = PackTot + PakGrp(i)

c
    var1 = SigTmp(i)
    if(var1 .lt. 0.D0) then
      var1 = 0.D0
      write(intstr1, '(i6)') i
      Warn1 = 'Temperature standard deviation defaulted to 0 for' //
&           'history file number' // intstr1
      Warn2 = ' '
      NW = NW + 1
      CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
    end if
    SigTmp(i) = var1

c
    var1 = SigRH(i)
    if(var1 .lt. 0.D0) then
      var1 = 0.D0
      write(intstr1, '(i6)') i
      Warn1 = 'RH standard deviation defaulted to 0 for' //
&           'history file number' // intstr1
      Warn2 = ' '
      NW = NW + 1
      CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
    end if
    SigRH(i) = var1
  end do

c
  if(PackTot .eq. 0) then
    Err1 = 'ERROR. Number of packages per group must be at ' //
&         'least 1 for at least'
    Err2 = 'one group. Program STOP.'
    ErrFound = .true.
```

```
      CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
      RETURN
    end if

c
c Read barrier thicknesses in cm and convert outer barrier thickness
c to micrometers and inner barrier thickness to millimeters. The
c fail % is the percent of inner barrier gone which will cause
c structural failure of package. Its converted to a fraction.
c
      read(inpfid, *) Penet1, Penet2
      write(outfid, 8004) Penet1, Penet2
      var1 = Penet1
      var2 = Penet2
      if(var1 .le. 0.D0) then
        var1 = 10.D0
        Warn1 = 'Outer barrier thickness defaulted to 10 cm.'
        Warn2 = ' '
        NW = NW + 1
        CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
      end if
      if(var2 .le. 0.D0) then
        var2 = 2.D0
        Warn1 = 'Inner barrier thickness defaulted to 2 cm.'
        Warn2 = ' '
        NW = NW + 1
        CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
      end if
      Penet1 = 10000.D0 * var1
      Penet2 = 10.D0 * var2

c
c Fail % is the percent of inner barrier gone which will cause
c structural failure of package. Its converted to a fraction.
c Variance share is fraction allocated to pack, patches get the rest.
c
      read(inpfid, *) InFail, varshare
      write(outfid, 80041) InFail, varshare
      var1 = InFail
      if(var1 .le. 0.D0 .or. var1 .gt. 100.D0) then
        var1 = 75.D0
        Warn1 = 'Percent thickness of inner barrier required ' //
        & 'to maintain structural'
        Warn2 = 'integrity defaulted to 75%.'
        NW = NW + 1
        CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
      end if
      InFail = 0.01D0 * var1

c
      if(varshare .lt. 0.D0 .or. varshare .gt. 1.D0) then
        Err1 = 'ERROR. Share of the variance assigned to ' //
```



```

&      'packages is required to be'
      Err2 = 'between 0 and 1. Program STOP'
      ErrFound = .true.
      CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
      RETURN
    end if

c
c Problem size parameters: number of packages, patches per package, and
c pits per patch.
c
      read(inpfid, *) NumPack, NumPatch, NPitCAM, NPitCRM
      write(outfid, 8005) NumPack, NumPatch, NPitCAM, NPitCRM
      intvar = NumPack
      if(intvar .gt. MAXPACK) then
        write(intstr1, '(i6)') intvar
        write(intstr2, '(i6)') MAXPACK
        Err1 = 'ERROR. Number of packages = ' // intstr1 //
&        ', exceeds current dimension = ' // intstr2
        Err2 = 'Program STOP'
        ErrFound = .true.
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
      else if(intvar .le. 0) then
        Warn1 = 'Number of packages defaulted to 400.'
        Warn2 = ' '
        NW = NW + 1
        CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
        intvar = 400
      end if
      NumPack = intvar

c
c Normalize number of packages per history group, if necessary.
c
      if(NumPack .ne. PackTot) then
        write(intstr1, '(i6)') NumPack
        Warn1 = 'Number of packages per group normalized to ' //
&        intstr1 // ', results are:'
        intvar = 1
        Cond = 0
        Warn2 = ' '
        do i = 1, NumHist - 1
          var1 = dble(PakGrp(i)) * dble(NumPack) / dble(PackTot)
          PakGrp(i) = int(var1 + 0.5D0)
          Cond = Cond + PakGrp(i)
          write(intstr1, '(i4)') PakGrp(i)
          Warn2(intvar: intvar + 3) = intstr1
          intvar = intvar + 4
        end do
        PakGrp(NumHist) = max(NumPack - Cond, 0)

```

```

        write(intstr1, '(i4)') PakGrp(NumHist)
        Warn2(intvar: intvar + 3) = intstr1
        NW = NW + 1
        CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
    end if

C
    intvar = NumPatch
    if(intvar .le. 0) then
        Warn1 = 'Number of patches/package defaulted to 1201.'
        Warn2 = ' '
        NW = NW + 1
        CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
        intvar = 1201
    end if
    NumPatch = intvar

C
    intvar = NPitCAM
    if(intvar .gt. MAXPITCAM) then
        write(intstr1, '(i6)') intvar
        write(intstr2, '(i6)') MAXPITCAM
        Err1 = 'ERROR. Number of pits/patch = ' // intstr1 //
        &      ', exceeds current dimension = ' // intstr2
        Err2 = 'for the outer barrier. Program STOP'
        ErrFound = .true.
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
    else if(intvar .le. 0) then
        Err1 = 'ERROR. Number of pits/patch on the outer barrier ' //
        &      'must be positive.'
        Err2 = 'Program STOP'
        ErrFound = .true.
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
    end if

C
C Pits on CRM checked for too big here, later if can be zero.
C
    intvar = NPitCRM
    if(intvar .gt. MAXPITCRM) then
        write(intstr1, '(i6)') intvar
        write(intstr2, '(i6)') MAXPITCRM
        Err1 = 'ERROR. Number of pits/patch = ' // intstr1 //
        &      ', exceeds current dimension = ' // intstr2
        Err2 = 'for the inner barrier. Program STOP'
        ErrFound = .true.
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
    else if(intvar .lt. 0) then
        Err1 = 'ERROR. Number of pits/patch on the inner barrier ' //

```

```

&      'cannot be negative.'
      Err2 = 'Program STOP'
      ErrFound = .true.
      CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
      RETURN
    end if

c
c Read start time, end time, and number of bins.
c
      read(inpfid, *) bin0, binf, nbin
      write(outfid, 8006) bin0, binf, nbin
      var1 = bin0
      IF (var1 .lt. 1.D0) then
        Warn1 = 'Binning start time defaulted to one year.'
        Warn2 = ' '
        NW = NW + 1
        CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
        var1 = 1.D0
      end if
      bin0 = var1

c
      var1 = binf
      IF (var1 .le. bin0) then
        Warn1 = 'Binning end time defaulted to 100,000 years.'
        Warn2 = ' '
        NW = NW + 1
        CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
        var1 = 1.D5
      end if
      binf = var1

c
      if(binf .le. bin0) then
        write(realstr, '(1p11.4)') binf
        Err1 = 'ERROR. Bin ending time = ' // realstr //
        &      ', is used for problem simulation time.'
        write(realstr, '(1p11.4)') bin0
        Err2 = 'It must exceed bin start time = ' // realstr //
        &      ', Program STOP.'
        ErrFound = .true.
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
      end if

c
      intvar = nbin
      if(intvar .gt. MAXBIN) then
        write(intstr1, '(i6)') intvar
        write(intstr2, '(i6)') MAXBIN
        Err1 = 'ERROR. Number of bins = ' // intstr1 //
        &      ', exceeds current dimension = ' // intstr2

```

```

    Err2 = 'Program STOP.'
    ErrFound = .true.
    CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
    RETURN
else if(intvar .le. 0) then
    Warn1 = 'Number of bins defaulted to 1200.'
    Warn2 = ' '
    NW = NW + 1
    CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
    intvar = 1200
end if
nbin = intvar
BinFact = nbin / LOG(bin0 / binf)
BinTerm = BinFact * LOG(bin0)
c
c Read four times at which the number of penetrated pits is to be
c reported to the OUT file.
c
    read(inpfid, *) bin1, bin2, bin3, bin4
    write(outfid, 8007) bin1, bin2, bin3, bin4
    var1 = bin1
    if(var1 .lt. bin0) then
        Warn1 = 'First bin report time defaulted to 10% of ' //
        & 'binning end time.'
        Warn2 = ' '
        NW = NW + 1
        CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
        var1 = 0.1 * binf
    end if
    bin1 = var1
c
    var1 = bin2
    if(var1 .lt. bin0) then
        Warn1 = 'Second bin report time defaulted to 20% of ' //
        & 'binning end time.'
        Warn2 = ' '
        NW = NW + 1
        CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
        var1 = 0.2 * binf
    end if
    bin2 = var1
c
    var1 = bin3
    if(var1 .lt. bin0) then
        Warn1 = 'Third bin report time defaulted to 50% of ' //
        & 'binning end time.'
        Warn2 = ' '
        NW = NW + 1
        CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)

```

```

        var1 = 0.5 * binf
    end if
    bin3 = var1
c
    var1 = bin4
    if(var1 .lt. bin0) then
        Warn1 = 'Last bin report time defaulted to binning end ' //
        & 'time.'
        Warn2 = ' '
        NW = NW + 1
        CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
        var1 = binf
    end if
    bin4 = var1
c
c Convert bin time locations to bin numbers
c
    binrat = dble(nbin) / LOG(bin4 / bin0)
    Logbin0 = LOG(bin0)
    ibin1 = 1 + int(binrat * (LOG(bin1) - Logbin0))
    ibin2 = 1 + int(binrat * (LOG(bin2) - Logbin0))
    ibin3 = 1 + int(binrat * (LOG(bin3) - Logbin0))
    ibin4 = 1 + int(binrat * (LOG(bin4) - Logbin0))
c
c Read seed for the random number generator.
c
    read(inpfid, *) iseed
    write(outfid, 8008) iseed
c
c Read maximum change allowed in temp and RH over a time step.
c
    read(inpfid, *) TmpAllow, RHallow
    write(outfid, 8009) TmpAllow, RHallow
    var1 = TmpAllow
    if(var1 .lt. 0.D0) then
        Warn1 = 'No restrictions on time step size, due to user-' //
        & 'specified maximum'
        Warn2 = 'temperature change, are applied.'
        NW = NW + 1
        CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
        var1 = 0.D0
    end if
    TmpAllow = var1
c
    var1 = RHallow
    if(var1 .lt. 0.D0) then
        Warn1 = 'No restrictions on time step size, due to user-' //
        & 'specified maximum'
        Warn2 = 'relative humidity change, are applied.'

```

```

        NW      = NW + 1
        CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
        var1 = 0.D0
    end if
    RHallow = var1
c
c Read definition for top of package in terms of an angle from the
c center of the cylinder and the percent of top patches seeing drips.
c
    read(inpfid, *) TopAngle, PctTopDrip
    write(outfid, 8010) TopAngle, PctTopDrip
    var1 = TopAngle
    if(var1 .lt. 0.D0 .or. var1 .gt. 360.D0) then
        Err1 = 'ERROR. Angle defining top of package must be ' //
        &      'between 0 and 360.'
        Err2 = 'Program STOP'
        ErrFound = .true.
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
    end if
    NumTop = int((TopAngle / 360.D0) * NumPatch + 0.5D0)
c
    var1 = PctTopDrip
    if(var1 .lt. 0.D0 .or. var1 .gt. 100.D0) then
        Err1 = 'ERROR. Percent of patches on top of package that ' //
        &      'see drips must be'
        Err2 = 'between 0 and 100. Program STOP'
        ErrFound = .true.
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
    end if
    TopDripFrc = PctTopDrip / 100.D0
c
c Read definition for top of package in terms of an angle from the
c center of the cylinder and the percent of top patches seeing drips.
c
    read(inpfid, *) BotAngle, PctBotDrip
    write(outfid, 8011) BotAngle, PctBotDrip
    var1 = BotAngle
    if(var1 .lt. 0.D0 .or. var1 .gt. 360.D0 - TopAngle) then
        write(realstr, '(f6.2,5x)') 360.D0 - TopAngle
        Err1 = 'ERROR. Angle defining bottom of package must be ' //
        &      'between 0 and ' // realstr
        Err2 = 'Program STOP'
        ErrFound = .true.
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
    end if
    NumBot = int((BotAngle / 360.D0) * NumPatch + 0.5D0)

```

```

c
var1 = PctBotDrip
if(var1 .lt. 0.D0 .or. var1 .gt. 100.D0) then
  Err1 = 'ERROR. Percent of patches on bottom of package that ' //
  & 'see drips must be'
  Err2 = 'between 0 and 100. Program STOP'
  ErrFound = .true.
  CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
  RETURN
end if
BotDripFrc = PctBotDrip / 100.D0

c
do while (NumTop + NumBot .gt. NumPatch)
  if(NumTop .gt. 1) then
    NumTop = NumTop - 1
  else
    NumBot = NumBot - 1
  end if
end do
NumSide = NumPatch - NumTop - NumBot
Warn1 = 'The number of top, bottom, and side patches per ' //
& 'package are'
Warn2 = ' '
write(intstr1, '(i6)') NumTop
Warn2(2: 9) = intstr1 // ', '
write(intstr1, '(i6)') NumBot
Warn2(10: 17) = intstr1 // ', '
write(intstr1, '(i6)') NumSide
Warn2(18: 37) = intstr1 // ', respectively.'
CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, 0)

c
c Read drip start, end, and change times only if possibility of drips.
c
c if(TopDripFrc .gt. 0.D0 .or. BotDripFrc .gt. 0.D0) then
  SomeDrips = .true.

c
c Read distribution for dripping start time.
c
  CALL InitDPArray(vard, 4)
  CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, intvar,
  & vard, CDFName)
  if(ErrFound) then
    CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
    RETURN
  end if
  if(intvar .eq. DSTCDF) then
    CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
    & CDFStackEnd, vard)
    if(ErrFound) then

```

```

        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
    end if
else if(intvar .eq. DSTDEF) then
    intvar = DSTFXD          !Fixed at time zero, if default
end if
DripStartType = intvar
DripStartPar(1) = vard(1)
DripStartPar(2) = vard(2)
DripStartPar(3) = vard(3)
DripStartPar(4) = vard(4)
c
c Read distribution for dripping end time.
c
    CALL InitDPArray(vard, 4)
    CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, intvar,
    &          vard, CDFName)
    if(ErrFound) then
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
    end if
    if(intvar .eq. DSTCDF) then
        CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
    &          CDFStackEnd, vard)
        if(ErrFound) then
            CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
            RETURN
        end if
    else if(intvar .eq. DSTDEF) then
        intvar = DSTFXD
        vard(1) = 1.1D0 * binf    !Stop after simulation, if default
    end if
    DripEndType = intvar
    DripEndPar(1) = vard(1)
    DripEndPar(2) = vard(2)
    DripEndPar(3) = vard(3)
    DripEndPar(4) = vard(4)
c
c Read water condition at start time, neutral (T) or not (F) and
c whether the condition changes at some point in time (T or F).
c
    read(inpfid, *) NeutD1, NewDrips
    write(outfid, 8012) NeutD1, NewDrips
    HaveNeut = NeutD1
    HaveChem = .not. NeutD1
c
c Only read data for water condition change if NewDrips = true.
c
    if(NewDrips) then

```



```

C
C Read distribution for time when dripping water chemistry changes.
C
      CALLInitDPArray(vard, 4)
      CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, intvar,
        &          vard, CDFName)
      if(Errfound) then
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
      end if
      if(intvar .eq. DSTCDF) then
        CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
          &          CDFStackEnd, vard)
        if(Errfound) then
          CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
          RETURN
        end if
      else if(intvar .eq. DSTDEF) then
        intvar = DSTFXD !No logical default, set to time zero
      end if
      DripChnType = intvar
      DripChnPar(1) = vard(1)
      DripChnPar(2) = vard(2)
      DripChnPar(3) = vard(3)
      DripChnPar(4) = vard(4)

C
C Since have change, new water condition is opposite of initial.
C
      NeutD2 = .not. NeutD1
      HaveNeut = .true.
      HaveChem = .true.
    end if
  else
    SomeDrips = .false.
    HaveNeut = .false.
    HaveChem = .false.
  end if

C
C Read distribution for time delay range induced by ceramic protection.
C
      CALLInitDPArray(vard, 4)
      CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, intvar, vard,
        &          CDFName)
      if(Errfound) then
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
      end if
      if(intvar .eq. DSTCDF) then
        CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,

```

```

&          CDFStackEnd, vard)
  if(Errfound) then
    CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
    RETURN
  end if
  else if(intvar .eq. DSTDEF) then
    intvar = DSTFXD          !Set to time 0, if default
  end if
  CermType = intvar
  CermPar(1) = vard(1)
  CermPar(2) = vard(2)
  CermPar(3) = vard(3)
  CermPar(4) = vard(4)
c
c Read segment header for model appropriate for patches NOT seeing
c drips, for the OUTER barrier. Then read the model name and its
c supporting distributions.
c
  read(inpfid, 8000) dum
  write(outfid, 8000) dum
  CALL UpCase(dum, 20)
  if(dum(1: 20) .ne. '[NO DRIP MODEL, CAM]') then
    Err1 = 'ERROR. Missing segment header = [No Drip Model, CAM]' //
    &      ' which must follow'
    Err2 = 'ceramic life distribution. Program STOP'
    ErrFound = .true.
    CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
    RETURN
  end if
  CALL InitIntArray(MSpec, 6)
c
  Cond = 0
  CALL ReadModelType(inpfid, outfid, auxfid, CDFStack, vg, varshare,
&          ErrFound, CDFStackEnd, RateMax, Cond, Err1,
&          Err2, ModBranch, NumD, DType, DParm, NW,
&          NCRMD, ATemp)
  if(Errfound) then
    CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
    RETURN
  end if
  if(ModBranch .ne. CAMGPM) then
    Err1 = 'ERROR. CAM corrosion w/o drips must be modeled ' //
    &      'with General+PitMultiples.'
    Err2 = 'Program STOP'
    ErrFound = .true.
    CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
    RETURN
  end if
  MSpec(1) = ModBranch

```

```

do i = 1, NumD
  DTypeStack(i) = DType(i)
  DParmStack(i, 1) = DParm(i, 1)
  DParmStack(i, 2) = DParm(i, 2)
  DParmStack(i, 3) = DParm(i, 3)
  DParmStack(i, 4) = DParm(i, 4)
end do
EndStack(1) = NumD
c
c Read segment header for model appropriate for patches NOT seeing
c drips, for the INNER barrier. Then read the model name and its
c supporting distributions.
c
  read(inpfid, 8000) dum
  write(outfid, 8000) dum
  CALL UpCase(dum, 20)
  if(dum(1: 20) .ne. '[NO DRIP MODEL, CRM]') then
    Err1 = 'ERROR. Missing segment header = [No Drip Model, CRM]' //
    &      ' which must follow'
    Err2 = 'segment [No Drip Model, CAM]. Program STOP'
    ErrFound = .true.
    CALL WriteMessage(Err1, Err2, .true., outfild, auxfid, 0)
    RETURN
  end if
  CALL InitDPArray(ATemp, MAXCRMDISTS)
  CALL ReadModelType(inpfid, outfild, auxfid, CDFStack, vg, varshare,
&      ErrFound, CDFStackEnd, RateMax, Cond, Err1,
&      Err2, ModBranch, NumD, DType, DParm, NW,
&      NCRMD, ATemp)
  if(Errfound) then
    CALL WriteMessage(Err1, Err2, .true., outfild, auxfid, 0)
    RETURN
  end if
  if(ModBranch .ne. CRMGONLY .and. ModBranch .ne. CRMGRONLY .and.
& ModBranch .ne. CRMEXP) then
    Err1 = 'ERROR. CRM corrosion w/o drips must be modeled ' //
    &      'with General Arrhenius,'
    Err2 = 'General Rate, or with Exponential PitRate(TSPA95). ' //
    &      'Program STOP'
    ErrFound = .true.
    CALL WriteMessage(Err1, Err2, .true., outfild, auxfid, 0)
    RETURN
  end if
  if(ModBranch .eq. CRMEXP) then
    if(SomeDrips) then
      Err1 = 'ERROR. Exponential Pit Rate of TSPA95 vintage ' //
      &      'can only be run when'
      Err2 = 'there is no possibility of drips. Program STOP'
      ErrFound = .true.
    end if
  end if

```

```

        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
    end if
    if(NPitCRM .ne. NPitCAM) then
        Err1 = 'ERROR. Exponential Pit Rate of TSPA95 vintage ' //
        &      'requires an equal number'
        Err2 = 'of pits on both barriers. Program STOP'
        ErrFound = .true.
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
    end if
    HaveNeut = .false.
    HaveChem = .false.
end if
if(ModBranch .eq. CRMGRONLY) then
    NumCRMD(1) = NCRMD
    do i = 1, NCRMD
        NDCRMT(i) = ATemp(i) + 273.15D0
    end do
end if
MSpec(2) = ModBranch
CRMMax(1) = RateMax
do i = 1, NumD
    DTypeStack(EndStack(1) + i) = DType(i)
    DParmStack(EndStack(1) + i, 1) = DParm(i, 1)
    DParmStack(EndStack(1) + i, 2) = DParm(i, 2)
    DParmStack(EndStack(1) + i, 3) = DParm(i, 3)
    DParmStack(EndStack(1) + i, 4) = DParm(i, 4)
end do
EndStack(2) = EndStack(1) + NumD
c
c . Read segment header for thresholds, etc appropriate for patches NOT
c seeing drips. Then read distributions and fixed values.
c
    read(inpfid, 8000) dum
    write(outfid, 8000) dum
    CALL UpCase(dum, 18)
    if(dum(1: 18) .ne. '[NO DRIP FEATURES]') then
        Err1 = 'ERROR. Missing segment header = [No Drip Features]' //
        &      ' which must follow'
        Err2 = 'segment [No Drip Model, CRM]. Program STOP'
        ErrFound = .true.
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
    end if
    StackIt = 3
    CALL ReadFeatures(inpfid, outfid, auxfid, Penet1, DTypeStack,
    &                  DParmStack, EndStack, CDFStack, NDFeat,
    &                  ErrFound, StackIt, Cond, Err1, Err2, NW,

```

```
&                                CDFStackEnd)
  if(ErrFound) then
    CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
    RETURN
  end if
c
c If there is a possibility of NEUTRAL drips, read segment header for
c model for patches SEEING NEUTRAL drips, for the OUTER barrier. Then
c read the model name and its supporting distributions.
c
  OnlyGenCRM = .true.
  if(HaveNeut) then
    read(inpfid, 8000) dum
    write(outfid, 8000) dum
    CALL UpCase(dum, 25)
    if(dum(1: 25) .ne. '[NEUTRAL DRIP MODEL, CAM]') then
      Err1 = 'ERROR. Missing segment header = [Neutral Drip ' //
      &      'Model, CAM] which must'
      Err2 = 'follow the [No Drip Features] segment. Program STOP'
      ErrFound = .true.
      CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
      RETURN
    end if
    Cond = 1
    CALL ReadModelType(inpfid, outfid, auxfid, CDFStack, vg,
      &      varshare, ErrFound, CDFStackEnd, RateMax,
      &      Cond, Err1, Err2, ModBranch, NumD, DType,
      &      DParm, NW, NCRMD, ATemp)
    if(Errfound) then
      CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
      RETURN
    end if
    if(ModBranch .ne. CAMGPM) then
      Err1 = 'ERROR. CAM corrosion with neutral drips must be ' //
      &      'modeled with the'
      Err2 = 'CAMGeneral+PitMultiples corrosion model. Program STOP'
      ErrFound = .true.
      CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
      RETURN
    end if
    MSpec(3) = ModBranch
    do i = 1, NumD
      DTypeStack(EndStack(3) + i) = DType(i)
      DParmStack(EndStack(3) + i, 1) = DParm(i, 1)
      DParmStack(EndStack(3) + i, 2) = DParm(i, 2)
      DParmStack(EndStack(3) + i, 3) = DParm(i, 3)
      DParmStack(EndStack(3) + i, 4) = DParm(i, 4)
    end do
    EndStack(4) = EndStack(3) + NumD
```

```

c
c Read segment header for model appropriate for patches SEEING NEUTRAL
c dripping water, for the INNER barrier. Then read the model name and
c its supporting distributions.
c
      read(inpfid, 8000) dum
      write(outfid, 8000) dum
      CALL UpCase(dum, 25)
      if(dum(1: 25) .ne. '[NEUTRAL DRIP MODEL, CRM]') then
        Err1 = 'ERROR. Missing segment header = [Neutral Drip ' //
          &      'Model, CRM] which must'
        Err2 = 'follow the segment [Neutral Drip Model, CAM]. ' //
          &      'Program STOP'
        &
        ErrFound = .true.
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
      end if
      CALL InitDPArray(ATemp, MAXCRMDISTS)
      CALL ReadModelType(inpfid, outfid, auxfid, CDFStack, vg,
        &      varshare, ErrFound, CDFStackEnd, RateMax,
        &      Cond, Err1, Err2, ModBranch, NumD, DType,
        &      DParm, NW, NCRMD, ATemp)
      if(Errfound) then
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
      end if
      if(ModBranch .ne. CRMGONLY .and. ModBranch .ne. CRMGRONLY .and.
        &      ModBranch .ne. CRMGPGL .and. ModBranch .ne. CRMGRPGL .and.
        &      ModBranch .ne. CRMGLOG .and. ModBranch .ne. CRMGRLOG .and.
        &      ModBranch .ne. CRMGLNL .and. ModBranch .ne. CRMGRNLNL) then
        Err1 = 'ERROR. CRM corrosion with neutral drips must be ' //
          &      'modeled with general-only'
        Err2 = 'or general plus a pit growth law. Program STOP'
        ErrFound = .true.
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
      end if
      if(ModBranch .ne. CRMGONLY .and. ModBranch .ne. CRMGRONLY) then
        if(NPitCRM .eq. 0) then
          Err1 = 'ERROR. CRM pitting corrosion requires a positive '//
            &      'number of pits.'
          Err2 = 'Program STOP'
          ErrFound = .true.
          CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
          RETURN
        end if
        OnlyGenCRM = .false.
      end if
      if(MSpec(2) .eq. CRMGONLY) then

```

```

      if (ModBranch .ne. CRMGONLY .and. ModBranch .ne. CRMGPGL .and.
&      ModBranch .ne. CRMGLOG .and. ModBranch .ne. CRMGLNL) then
        ErrFound = .true.
      end if
    else if (MSpec(2) .eq. CRMGRONLY) then
      if (ModBranch .ne. CRMGRONLY .and. ModBranch .ne. CRMGRPGL .and.
&      ModBranch .ne. CRMGRLOG .and. ModBranch .ne. CRMGRNL) then
        ErrFound = .true.
      end if
    end if
    if (ErrFound) then
      Err1 = 'ERROR. CRM general corrosion models must be the '//
&      'same under all conditions.'
      Err2 = 'Program STOP'
      CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
      RETURN
    end if
    if (ModBranch .ne. CRMGONLY .and. ModBranch .ne. CRMGRONLY) then
      OnlyGenCRM = .false.
    end if
    if (ModBranch .eq. CRMGRONLY .or. ModBranch .eq. CRMGRPGL .or.
&      ModBranch .eq. CRMGRLOG .or. ModBranch .eq. CRMGRNL) then
      NumCRMD(2) = NCRMD
      do i = 1, NCRMD
        GDCRMT(i) = ATemp(i) + 273.15D0
      end do
    end if
    MSpec(4) = ModBranch
    CRMMax(2) = RateMax
    do i = 1, NumD
      DTypeStack(EndStack(4) + i) = DType(i)
      DParmStack(EndStack(4) + i, 1) = DParm(i, 1)
      DParmStack(EndStack(4) + i, 2) = DParm(i, 2)
      DParmStack(EndStack(4) + i, 3) = DParm(i, 3)
      DParmStack(EndStack(4) + i, 4) = DParm(i, 4)
    end do
    EndStack(5) = EndStack(4) + NumD
c
c Read segment header for thresholds, etc appropriate for patches
c SEEING NEUTRAL drips. Then read distributions and fixed values.
c
      read(inpfid, 8000) dum
      write(outfid, 8000) dum
      CALL UpCase(dum, 23)
      if (dum(1: 23) .ne. '[NEUTRAL DRIP FEATURES]') then
        Err1 = 'ERROR. Missing segment header = [Neutral Drip ' //
&        'Features] which must follow'
        Err2 = 'the [Neutral Drip Model, CRM] segment. Program STOP'
        ErrFound = .true.

```

```

        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
    end if
    StackIt = 6
    CALL ReadFeatures(inpfid, outfid, auxfid, Penet1, DTypeStack,
&                DParmStack, EndStack, CDFStack, GDFeat,
&                ErrFound, StackIt, Cond, Err1, Err2, NW,
&                CDFStackEnd)
    if(ErrFound) then
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
    end if
end if

c
c If there is a possibility of NON-NEUTRAL drips, read segment header
c for model for patches SEEING NON-NEUTRAL drips for the OUTER barrier.
c Then read the model name and its supporting distributions.
c
    if(HaveChem) then
        read(inpfid, 8000) dum
        write(outfid, 8000) dum
        CALL UpCase(dum, 26)
        if(dum(1: 26) .ne. '[CHEMICAL DRIP MODEL, CAM]') then
            Err1 = 'ERROR. Missing segment header = [Chemical Drip ' //
&                'Model, CAM] which must be'
            Err2 = 'present if non-neutral drips are possible. ' //
&                'Program STOP'
            ErrFound = .true.
            CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
            RETURN
        end if
        Cond = 2
        CALL ReadModelType(inpfid, outfid, auxfid, CDFStack, vg,
&                varshare, ErrFound, CDFStackEnd, RateMax,
&                Cond, Err1, Err2, ModBranch, NumD, DType,
&                DParm, NW, NCRMD, ATemp)
        if(Errfound) then
            CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
            RETURN
        end if
        if(ModBranch .ne. CAMGPM .and. ModBranch .ne. CAMGPGL) then
            Err1 = 'ERROR. CAM corrosion with non-neutral drips must ' //
&                'be modeled with one'
            Err2 = 'of the CAM corrosion models. Program STOP'
            ErrFound = .true.
            CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
            RETURN
        end if
        if(MSpec(3) .gt. 0) then

```



```

      MSpec(5) = ModBranch
    else
      MSpec(3) = ModBranch
    end if
    do i = 1, NumD
      DTypeStack(EndStack(StackIt) + i) = DType(i)
      DParmStack(EndStack(StackIt) + i, 1) = DParm(i, 1)
      DParmStack(EndStack(StackIt) + i, 2) = DParm(i, 2)
      DParmStack(EndStack(StackIt) + i, 3) = DParm(i, 3)
      DParmStack(EndStack(StackIt) + i, 4) = DParm(i, 4)
    end do
    EndStack(StackIt + 1) = EndStack(StackIt) + NumD
    StackIt = StackIt + 1
c
c Read segment header for model for patches SEEING NON-NEUTRAL dripping
c water, for the INNER barrier. Then read the model name and its
c supporting distributions.
c
      read(inpfid, 8000) dum
      write(outfid, 8000) dum
      CALL UpCase(dum, 26)
      if(dum(1: 26) .ne. '[CHEMICAL DRIP MODEL, CRM]') then
        Err1 = 'ERROR. Missing segment header = [Chemical Drip ' //
          & 'Model, CRM] which must'
        Err2 = 'be present if non-neutral drips are possible. ' //
          & 'Program STOP'
        ErrFound = .true.
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
      end if
      CALL InitDPArray(ATemp, MAXCRMDISTS)
      CALL ReadModelType(inpfid, outfid, auxfid, CDFStack, vg,
        & varshare, ErrFound, CDFStackEnd, RateMax,
        & Cond, Err1, Err2, ModBranch, NumD, DType,
        & DParm, NW, NCRMD, ATemp)
      if(Errfound) then
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
      end if
      if(ModBranch .ne. CRMGONLY .and. ModBranch .ne. CRMGRONLY .and.
        & ModBranch .ne. CRMGPGL .and. ModBranch .ne. CRMGRPGL .and.
        & ModBranch .ne. CRMGLOG .and. ModBranch .ne. CRMGRLOG .and.
        & ModBranch .ne. CRMGLNL .and. ModBranch .ne. CRMGRNLNL) then
        Err1 = 'ERROR. CRM corrosion with neutral drips must be ' //
          & 'modeled with general-only'
        Err2 = 'or general plus a pit growth law. Program STOP'
        ErrFound = .true.
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
      end if

```

```
end if
if (ModBranch .ne. CRMGONLY .and. ModBranch .ne. CRMGRONLY) then
  if (NPitCRM .eq. 0) then
    Err1 = 'ERROR. CRM pitting corrosion requires a positive '//
    & 'number of pits.'
    Err2 = 'Program STOP'
    ErrFound = .true.
    CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
    RETURN
  end if
  OnlyGenCRM = .false.
end if
if (MSpec(2) .eq. CRMGONLY) then
  if (ModBranch .ne. CRMGONLY .and. ModBranch .ne. CRMGPGL .and.
  & ModBranch .ne. CRMGLOG .and. ModBranch .ne. CRMGLNL) then
    ErrFound = .true.
  end if
  else if (MSpec(2) .eq. CRMGRONLY) then
    if (ModBranch .ne. CRMGRONLY .and. ModBranch .ne. CRMGRPGL .and.
    & ModBranch .ne. CRMGRLOG .and. ModBranch .ne. CRMGRNL) then
      ErrFound = .true.
    end if
  end if
  if (ErrFound) then
    Err1 = 'ERROR. CRM general corrosion models must be the '//
    & 'same under all conditions.'
    Err2 = 'Program STOP'
    CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
    RETURN
  end if
  if (MSpec(4) .gt. 0) then
    MSpec(6) = ModBranch
    if (MSpec(4) .ne. CRMGONLY .and. MSpec(6) .ne. CRMGONLY .and.
    & MSpec(4) .ne. CRMGRONLY .and. MSpec(6) .ne. CRMGRONLY) then
      if (MSpec(4) .ne. MSpec(6)) then
        Err1 = 'ERROR. CRM pit corrosion model can be of only ' //
        & 'one type for both kinds'
        Err2 = 'of drips. Program STOP'
        ErrFound = .true.
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
      end if
    end if
  end if
  else
    MSpec(4) = ModBranch
  end if
  if (ModBranch .eq. CRMGRONLY .or. ModBranch .eq. CRMGRPGL .or.
  & ModBranch .eq. CRMGRLOG .or. ModBranch .eq. CRMGRNL) then
    NumCRMD(3) = NCRMD
```

```

      do i = 1, NCRMD
        BDCRMT(i) = ATemp(i) + 273.15D0
      end do
    end if
    CRMMax(3) = RateMax
    do i = 1, NumD
      DTypeStack(EndStack(StackIt) + i) = DType(i)
      DParmStack(EndStack(StackIt) + i, 1) = DParm(i, 1)
      DParmStack(EndStack(StackIt) + i, 2) = DParm(i, 2)
      DParmStack(EndStack(StackIt) + i, 3) = DParm(i, 3)
      DParmStack(EndStack(StackIt) + i, 4) = DParm(i, 4)
    end do
    EndStack(StackIt + 1) = EndStack(StackIt) + NumD
    StackIt = StackIt + 1
  c
  c Read segment header for thresholds, etc appropriate for patches
  c SEEING NEUTRAL drips. Then read distributions and fixed values.
  c
    read(inpfid, 8000) dum
    write(outfid, 8000) dum
    CALL UpCase(dum, 24)
    if(dum(1: 24) .ne. '[CHEMICAL DRIP FEATURES]') then
      Err1 = 'ERROR. Missing segment header = [Chemical Drip ' //
        & 'Features] which must be'
      Err2 = 'present if non-neutral drips are possible. ' //
        & 'Program STOP'
      ErrFound = .true.
      CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
      RETURN
    end if
    StackIt = StackIt + 1
    CALL ReadFeatures(inpfid, outfid, auxfid, Penet1, DTypeStack,
      & DParmStack, EndStack, CDFStack, BDFeat,
      & ErrFound, StackIt, Cond, Err1, Err2, NW,
      & CDFStackEnd)
    if(ErrFound) then
      CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
      RETURN
    end if
  end if
  c
  c If all inner barrier corrosion models and general corrosion only,
  c then set number of inner barrier pits to zero.
  c
    if(OnlyGenCRM) then
      NPitCRM = 0
    end if
  c
  c Close input file. If get to here, didn't return with an error.

```

```

c
    CLOSE(inpfid)
    write(*, 7100) NW
7100 format(' No errors detected.',i5,' defaults applied.',/)
    if(NW .gt. 0) write(auxfid, *) ' '
c
    write(outfid,7000) bin1, bin2, bin3, bin4, bin1, bin2, bin3, bin4
7000 format(242('='),/,
    & ' Pack Advection Diffusion Structure 1st Penet 1st Penet',
    & ' [ Outer Barrier ]',
    & ' [ Inner Barrier - No Drips ]',
    & ' [ Inner Barrier - Drips ]',
    & ' [Cumulative number of pits at times(y)]',
    & ' [Cumulative num of patches at times(y)]',/,
    & ' No Start Start Failure Outer Inner ',
    & ' [First Pit Penetration Time (y)]',
    & ' [ First Patch Failure Time (y) ]',
    & ' [ 1st Pit Fail (y) 1st Patch Fail (y)]',
    & ' 4(1x,1PE8.1E1,1x),1x,4(1x,1PE8.1E1,1x),/,
    & ' Time(y) Time(y) Time(y) Time(y) Time(y) ',
    & ' [ Top Side Bottom ]',
    & ' [ Top Side Bottom ]',
    & ' [ Top Bottom Top Bottom ]',/,242('='))
c
8000 format(A80)
8001 format(A19)
8002 format(A5, T35,'|Version number of code')
80031 format(i10, T35,'|Number of alternate histories')
80032 format(A30, T35,'|History file name')
80033 format(i10,2(F10.2,1x),
    & T35,'|Group size and temp and RH std deviations')
8004 format(2(F10.2,1x),
    & T35,'|Thickness of outer/inner barriers(cm)')
80041 format(2(F10.2,1x),
    & T35,'|% for struct fail, variance share packs')
8005 format(4(I7,1x) ,T35,'|Simulated # of packages, patches and pits')
8006 format(2(F10.2,1x),I10,1x,
    & T35,'|Start and end times, and number of bins')
8007 format(4(1PE8.1E1),
    & T35,'|Time output (4) locations (yrs.)')
8008 format(I10, T35,'|Random number seed')
8009 format(2(F10.2,1x),
    & T35,'|Absolute change allowed in T (K), RH (%)')
8010 format(2(F10.2,1x),
    & T35,'|Angle defining top, % tops seeing drips')
8011 format(2(F10.2,1x),
    & T35,'|Angle defining bottom, % bottom in pools')
8012 format(11,2x,12, T35,'|Orig drips neutral, change occur')
c

```

```

      RETURN
      END !SUBROUTINE ReadFileInp
C
C *****
C
      SUBROUTINE ReadFileHST(hstfid, outfid, auxfid, hstfl, binf,
&                          ErrFound, NumStep, tim, tem, r, TmpMx,
&                          TmpMn, RHMax, RHMin, GroupNdx)
C
C  Read the time (tim), temperature (tem), and RH (r) history file and
C  set the simulation time. Convert temps to Kelvin and RHs to percent,
C  find min and max of each.
C
C  Input : hstfid, outfid, auxfid, hstfl, binf, GroupNdx
C  Output: ErrFound, NumStep, tim, tem, r, TmpMx, TmpMn, RHMax, RHMin
C  Local : Err1, Err2, intstr, i, TimRat
C
C  Arguments, and maximum sizes from include file.
C
      IMPLICIT NONE
      INCLUDE 'parm306.inc'
      character*(NAMELEN) hstfl(MAXHIST)
      logical ErrFound
      integer hstfid, outfid, auxfid, NumStep, GroupNdx
      double precision binf, tim(*), tem(*), r(*), TmpMx, TmpMn,
&                          RHMax, RHMin
C
C  Local variables
C
      character Err1*74, Err2*74, intstr*6
      integer i
      double precision TimRat
C
C  Open history file (existence already assured), read to end of file.
C
      OPEN(hstfid, FILE = hstfl(GroupNdx), STATUS = 'OLD')
      i = 1
      read(hstfid, *, end = 101) tim(i), tem(i), r(i)
      do while (tim(i) .le. binf)
        i = i + 1
        if(i .le. MAXDAT) then
          read(hstfid, *, end = 101) tim(i), tem(i), r(i)
        else
          write(intstr, '(i6)') MAXDAT
          Err1 = 'ERROR. Number of data points on history file ' //
&              'exceeds current'
          Err2 = 'dimension = ' // intstr // '. Program STOP'
          CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
          ErrFound = .true.

```

```

        RETURN
      end if
    end do
101 continue
    i = i - 1
    if(i .le. 1) then
      Err1 = 'ERROR. Bad history file ' // hstfl(GroupNdx)
      Err2 = 'Must have at least 2 points. Program STOP'
      CALL WriteMessage(Err1, Err2, .true., outf1d, auxf1d, 0)
      ErrFound = .true.
      RETURN
    end if
c
c If the simulation time (binf) is longer than the time history, then
c extrapolate the last values out, otherwise insert binf into the time
c history for the simulation time steps.
c
    if(tim(i) .lt. binf) then
      NumStep = i + 1
      if(NumStep .le. MAXDAT) then
        tim(NumStep) = binf
        tem(NumStep) = tem(i)
        r(NumStep) = r(i)
      else
        write(intstr, '(i6)') MAXDAT
        Err1 = 'ERROR. Extending history to simulation time adds ' //
          'more time steps than'
        Err2 = 'current dimension = ' // intstr // '. Program STOP'
        CALL WriteMessage(Err1, Err2, .true., outf1d, auxf1d, 0)
        ErrFound = .true.
        RETURN
      end if
    else
      NumStep = i
      TimRat = log(binf / tim(i-1)) / log(tim(i) / tim(i-1))
      tem(NumStep) = tem(i - 1) + (tem(i) - tem(i - 1)) * TimRat
      r(NumStep) = r(i - 1) + (r(i) - r(i - 1)) * TimRat
      tim(NumStep) = binf
    end if
c
c Convert Celsius to Kelvin & RH to percents, find min and max of each.
c
    TmpMx = 0.D0
    TmpMn = 1000.D0
    RHMax = 0.D0
    RHMin = 100.D0
    do i = 1, NumStep
      tem(i) = tem(i) + 273.15D0
      r(i) = 100.D0 * r(i)
    end do

```

```

      if(TmpMx .lt. tem(i)) TmpMx = tem(i)
      if(TmpMn .gt. tem(i)) TmpMn = tem(i)
      if(RHMax .lt. r(i))   RHMax = r(i)
      if(RHMin .gt. r(i))   RHMin = r(i)
    end do
    CLOSE(hstfid)
    RETURN
  END    !SUBROUTINE ReadFileHST

```

```

C
C *****
C
      SUBROUTINE ChkTimeGrid(outfid, auxfid, TmpAllow, RHallow,
&                               ErrFound, NumStep, tim, tem, r, TmpMx,
&                               TmpMn, RHMax, RHMin)
C
C   Since WapDeg equations assume constant temp and RH across a time
C   step, user can force time grid to allow only small changes over a
C   time step.  TmpAllow applies to temp and RHallow applies to RH.
C   If these are so small that more time steps are produced than
C   dimensions allow, an error condition is reported and the program
C   stops.
C
C   Input : outfid, auxfid, TmpAllow, RHallow,
C           NumStep, tim, tem, r, TmpMx, TmpMn, RHMax, RHMin
C   Output: NumStep, tim, tem, r, ErrFound
C   Local : intstring, Err1, Err2, InsertOne, i, j, TmpDiff, TmpAdd,
C           RHDiff, RHAdd, timNew, temNew, rNew, DiffMax, Rat
C
C   Arguments, and size parameters from include file
C
      IMPLICIT NONE
      INCLUDE 'parm306.inc'
      logical ErrFound
      integer outfid, auxfid, NumStep
      double precision TmpAllow, RHallow, tim(*), tem(*), r(*),
&                     TmpMx, TmpMn, RHMax, RHMin
C
C   Local variables
C
      character intstring*5
      character*74 Err1, Err2
      logical InsertOne
      integer i, j
      double precision TmpDiff, TmpAdd, RHDiff, RHAdd,
&                     timNew, temNew, rNew, DiffMax, Rat
C
      ErrFound = .false.
C
C   The next segment is accessed if the input grid is to be examined to

```

```

c ensure a controlled environment change over a time step.  If so,
c insert time steps as necessary.
c
  if(TmpAllow .gt. 0.D0 .OR. RHallow .gt. 0.D0) then
    if(TmpAllow .le. 0.D0) then
      DiffMax = MAX(TmpMx - TmpMn, 1.D0)
      TmpAllow = 2.D0 * DiffMax
    end if
    if(RHallow .le. 0.D0) then
      DiffMax = MAX(RHMax - RHMin, 1.D0)
      RHallow = 2.D0 * DiffMax
    end if
c
    i = 2
    do while (i .le. NumStep)
      TmpDiff = tem(i) - tem(i - 1)
      TmpAdd = sign(TmpAllow, TmpDiff)
      RHDiff = r(i) - r(i - 1)
      RHAdd = sign(RHallow, RHDiff)
      InsertOne = .false.
c
c See if temperature change too big.  If not, check RH change.  If they
c both show too large a change, apply temperature criterion first.
c
      if(abs(TmpDiff) .gt. TmpAllow) then
        temNew = tem(i - 1) + TmpAdd
        Rat = TmpAdd / TmpDiff
c(dl) timNew = (tim(i) - tim(i - 1)) * Rat + tim(i - 1)
        timNew = log(tim(i) / tim(i - 1)) * Rat + log(tim(i - 1))
        timNew = exp(timNew)
        rNew = (r(i) - r(i - 1)) * Rat + r(i - 1)
        InsertOne = .true.
c
      elseif(abs(RHDiff) .gt. RHallow) then
        rNew = r(i - 1) + RHAdd
        Rat = RHAdd / RHDiff
c(dl) timNew = (tim(i) - tim(i - 1)) * Rat + tim(i - 1)
        timNew = log(tim(i) / tim(i - 1)) * Rat + log(tim(i - 1))
        timNew = exp(timNew)
        temNew = (tem(i) - tem(i - 1)) * Rat + tem(i - 1)
        InsertOne = .true.
      end if
c
c If need to insert a time step, slide the current data back one index
c and insert the new values.  If sliding back causes the arrays to
c exceed dimensions, stop with an error message.
c
      if(InsertOne) then
        if(NumStep .lt. MAXDAT) then

```



```

do j = NumStep, i, -1
  tim(j + 1) = tim(j)
  tem(j + 1) = tem(j)
  r(j + 1) = r(j)
end do
NumStep = NumStep + 1
tim(i) = timNew
tem(i) = temNew
r(i) = rNew
i = i + 1

```

c

```

else
  write(intstring, '(i5)') MAXDAT
  Err1 = 'ERROR. Smoothing time grid for small ' //
  &      'environment changes causes more'
  Err2 = 'time steps than dimensions allow = ' // intstring
  CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
  ErrFound = .true.
  RETURN
end if

```

c

```

c Didn't need to insert step for this time interval, increment the
c counter to the next interval.

```

c

```

else
  i = i + 1
end if
end do
end if
RETURN
END      !SUBROUTINE ChkTimeGrid

```

c

```

c *****

```

c

```

SUBROUTINE SetModelParameters(Mean, v5, StanDev, varshare)

```

```

c Set initial values of simulation parameters. Mean and StanDev defined
c here are global. Locally define the covariance matrices, symmetrically
c decompose them into the v-array, multiply by sqrt(varshare) and store
c in the global array v5. Inner barrier values were read in, place
c them in their appropriate array slots.

```

```

c Input : varshare
c Output: Mean, v5, StanDev
c Local : i, j, k, varhmd, varaq, sum, pakshare, patshare, cov, v
c Arguments, and dimension parameters from the include file.

```

c

```

IMPLICIT NONE

```

```
      INCLUDE 'parm306.inc'
      double precision varshare, Mean(*), v5(NCOEFF, *),
&          StanDev(NMODEL, 2)

C
C Local variables
C
      integer i, j, k
      double precision varhmd, varaq, sum, pakshare, patshare
      double precision cov(NCOEFF, NCOEFF)
      double precision v(NCOEFF, NCOEFF)

C
C Parameters for humid air corrosion
C
      Mean(1) = 1.71847489D+01
      Mean(2) = -6.23461525D+02
      Mean(3) = -9.74458217D+02
      Mean(4) = 6.22702061D-01
      cov(1,1) = 6.99341851D+00
      cov(1,2) = -2.31785242D+02
      cov(1,3) = -5.23797460D+02
      cov(1,4) = -2.46076982D-02
      cov(2,2) = 1.04703380D+05
      cov(2,3) = -1.08919389D+04
      cov(2,4) = 2.57308977D+00
      cov(3,3) = 4.74698853D+04
      cov(3,4) = 1.30087278D+00
      cov(4,4) = 6.93193151D-04

C
C The standard deviation for the model is multiplied by sqrt(varshare)
C to give the package to package variability.
C
      pakshare = sqrt(varshare)
      patshare = sqrt(1.D0 - varshare)
      varhmd = 1.27572817D-01
      StanDev(1, 1) = sqrt(varhmd) * pakshare
      StanDev(1, 2) = sqrt(varhmd) * patshare

C
C Decompose the covariance matrix for humid air corrosion
C
      v(1, 1) = sqrt(cov(1, 1))
      do i = 2, 4
        v(i,1) = cov(1, i) / v(1, 1)
        do j = 2, (i - 1)
          sum = cov(j, i)
          do k = 1, (j - 1)
            sum = sum - v(i, k) * v(j, k)
          end do
          v(i, j) = sum / v(j, j)
        end do
      end do
```

```

sum = cov(i, i)
do j = 1, (i - 1)
    sum = sum - v(i, j) * v(i, j)
end do
v(i, i) = sqrt(sum)
end do

```

c
c Parameters for aqueous corrosion
c

```

Mean(5) = 1.11529542D+02
Mean(6) = 5.31987099D-01
Mean(7) = -2.32911746D+04
Mean(8) = -3.19184722D-04
cov(5,5) = 1.16629495D+02
cov(5,6) = -9.42263193D-04
cov(5,7) = -2.47611417D+04
cov(5,8) = -3.79257480D-04
cov(6,6) = 7.41490056D-04
cov(6,7) = 1.77037722D-11
cov(6,8) = 2.74906265D-19
cov(7,7) = 5.26274162D+06
cov(7,8) = 8.03108729D-02
cov(8,8) = 1.24097816D-09

```

c
c The standard deviation for the model is multiplied by sqrt(varshare)
c to give the package to package variability.
c

```

varaq = 3.62188651D-02
StanDev(2, 1) = sqrt(varaq) * pakshare
StanDev(2, 2) = sqrt(varaq) * patshare

```

c
c Decompose the covariance matrix for aqueous corrosion
c

```

v(5, 5) = sqrt(cov(5, 5))
do i = 6, 8
    v(i, 5) = cov(5, i) / v(5, 5)
    do j = 6, (i - 1)
        sum = cov(j, i)
        do k = 5, (j - 1)
            sum = sum - v(i, k) * v(j, k)
        end do
        v(i, j) = sum / v(j, j)
    end do
    sum = cov(i, i)
    do j = 5, (i - 1)
        sum = sum - v(i, j) * v(i, j)
    end do
    v(i, i) = sqrt(sum)
end do

```

```

c
c The decomposed matrices for the humid air model are stored in rows
c 1-4 and columns 1-8 (col 1-4 for package, col 5-8 for patch). The
c storage is similar for the aqueous corrosion model.
c

```

```

      do i = 1, 4
        do j = 1, 4
          v5(i, j) = v(i, j) * pakshare
          v5(i, j + 4) = v(i, j) * patshare
        end do
      end do
      do i = 5, 8
        do j = 1, 4
          v5(i, j) = v(i, j + 4) * pakshare
          v5(i, j + 4) = v(i, j + 4) * patshare
        end do
      end do
      RETURN
    END !SUBROUTINE SetModelParameters

```

```

c
c*****

```

```

c      SUBROUTINE SetSeedPacks(NumPack, iseed, SeedPacks)
c
c      Fills array SeedPacks(NSEED, NumPack) with random integers from
c      random number generator (rng) iran0
c
c      Input : NumPack, iseed, (Parameter NSEED from include file)
c      Output: SeedPacks
c      Local : i, j, iran0, iran0set
c
c      Arguments, and dimension sizes from the include file
c

```

```

      IMPLICIT NONE
      INCLUDE 'parm306.inc'
      integer NumPack, iseed
      integer SeedPacks(NSEED, NumPack)

```

```

c
c      Local variables
c
c      integer i, j, iran0, iran0set
c
c      Initialize random integer generator iran0 and then find seeds.
c

```

```

      iseed = iran0set(iseed)
      do j = 1, NumPack
        do i = 1, NSEED
          SeedPacks(i, j) = iran0()
        end do
      end do

```

```

      end do
      END !SUBROUTINE SetSeedPacks
C
C *****
C
      SUBROUTINE SetupPackage(auxfid, Mean, v5, StanDev, v9, c10,
&                               Penet1, NPitCAM, NPitCRM, binf, nbin,
&                               DripStartType, DripStartPar, DripEndType,
&                               DripEndPar, SomeDrips, NewDrips, HaveNeut,
&                               HaveChem, DripChnType, DripChnPar, NeutD1,
&                               MSpec, DTypeStack, DParmStack, Endstack,
&                               CDFStack, NumCRMD, varshare, vg, Pack,
&                               SeedPacks,
&                               Pit1st, Pat1st, BinStart, Bin, PatStart,
&                               PatBin, CRMThk, c9, NDPakLnA, NDPakK,
&                               NDPakR, NDTpVal, NDHASTart, NDAquaX,
&                               NDFeat, GDPakLnA, GDPakK, GDPakR, GDTpVal,
&                               GDPInit, GDHASTart, GDAquaX, GDFeat,
&                               BDPakLnA, BDPakK, BDPakR, BDTpVal,
&                               BDPInit, BDHASTart,
&                               BDAquaX, BDFeat, OnsetDrip, ChangeDrip,
&                               StopDrip, PwrIB, Coef1, NDCritDep,
&                               GDCritDep, BDCritDep, COPak, BDCAMB,
&                               BDCAMn, GDCRM1, GDCRM2, GDCRM3, BDCRM1,
&                               BDCRM2, BDCRM3)
C
C For the current waste package
C 1. Sample temperature threshold for corrosion initiation
C 2. Sample RH threshold for humid-air/aqueous corrosion transition
C 3. Set RH threshold for humid-air corrosion initiation
C 4. If package has ceramic coating, sample delay time for corrosion
C 5. Sample power for time term for inner barrier corrosion
C 6. If two time periods modeled, sample transition time
C 7. Sample parameters for outer barrier humid-air & aqueous corrosion
C 8. Sample parameters for the inner barrier corrosion model
C 9. Sample pit properties for the outer and inner barriers
C 10. Initialize first penetration times and zero out bin arrays
C 11. Write package properties to the AUX file.
C Input : auxfid, Mean, v5, StanDev, v9, Penet1, NPitCAM, NPitCRM,
C         binf, nbin, DripStartType, DripStartPar, DripEndType,
C         DripEndPar, SomeDrips, NewDrips, HaveNeut, HaveChem,
C         DripChnType, DripChnPar, NeutD1, MSpec, DTypeStack,
C         DParmStack, Endstack, CDFStack, NumCRMD, varshare, vg, Pack,
C         SeedPacks, NDFeat, GDFeat, BDFeat
C Output: c10, Pit1st, Pat1st, BinStart, Bin, PatStart, PatBin, CRMThk,
C         C9, NDPakLnA, NDPakK, NDTpVal, NDHASTart, NDAquaX, GDPakLnA,
C         GDPakK, GDTpVal, GDPInit, GDHASTart, GDAquaX, BDPakLnA,
C         BDPakK, BDTpVal, BDPInit, BDHASTart, BDAquaX, OnsetDrip,
C         ChangeDrip, StopDrip, PwrIB, Coef1, NDCritDep, GDCritDep,

```

```

c      BDCritDep, COPak, BDCAMB, BDCAMn, GDCRM1, GDCRM2, GDCRM3,
c      BDCRM1, BDCRM2, BDCRM3
c Local : GoodLaw, BadLaw, i, j, k, packint, Dist, StackIt, ModNdx,
c      ranval, Parm, Rate
c
c Arguments, and dimension size parameters
c
c      IMPLICIT NONE
c      INCLUDE 'parm306.inc'
c      logical SomeDrips, NewDrips, HaveNeut, HaveChem, NeutDl
c      integer auxfid, NPitCAM, NPitCRM, nbin, NumCRMD(*)
c      integer DripStartType, DripChnType, DripEndType
c      integer MSpec(*), DTypeStack(*), EndStack(*)
c      integer Pack, SeedPacks(NSEED, *)
c      integer Bin(MAXBIN, 3), PatBin(MAXBIN, 5)
c      double precision Mean(*), v5(NCOEFF, *), StanDev(NMODEL, 2), v9
c      double precision Penetl, binf, BinStart, PatStart, C9
c      double precision DripStartPar(*), DripChnPar(*), DripEndPar(*)
c      double precision DParmStack(MAXDISTS, 4), CDFStack(MAXPAIRS, 2)
c      double precision vg(2, 3, 4), CRMThk(*), varshare
c      double precision C10, Pit1st(*), Pat1st(*)
c      double precision NDFeat(*), GDFeat(*), BDFeat(*)
c      double precision NDPakLnA, NDPakK, NDTpVal, NDHASTart, NDAquaX,
c      & NDPakR(*)
c      double precision GDPakLnA, GDPakK, GDTpVal, GDPInit, GDHASTart,
c      & GDAquaX, GDPakR(*)
c      double precision BDPakLnA, BDPakK, BDTpVal, BDPInit, BDHASTart,
c      & BDAquaX, BDPakR(*)
c      double precision OnsetDrip, ChangeDrip, StopDrip, PwrIB
c      double precision Coefl(*), NDCritDep(*), GDCritDep(*),
c      & BDCritDep(*), COPak(*)
c      double precision BDCAMB(*), BDCAMn(*)
c      double precision GDCRM1(*), GDCRM2(*), GDCRM3(*)
c      double precision BDCRM1(*), BDCRM2(*), BDCRM3(*)
c
c Local variables
c
c      logical GoodLaw, BadLaw, granset
c      integer i, j, k, packint(4), Dist, StackIt, ModNdx, ranmzset
c      double precision ranval, Parm(4), ranmz, gran, SampleValue
c
c To ensure that package-level properties agree in two different runs
c (given the same seed, which produces the same seed packs) a random
c number is obtained for each property below whether it is used or not.
c Use seedpack to initialize random number generator for this package.
c
c      do i = 1, 4
c         packint(i) = SeedPacks(i, Pack)
c      end do

```

```

i      = ranmzset(packint)
GoodLaw = granset()
GoodLaw = .false.
BadLaw  = .false.

c
c Get sampled values for the general corrosion CAM model.
c Sample 8 independent numbers from a standard normal distribution.
c These numbers will be used to generate the mean values for the
c general corrosion models for the outer barrier and the pitting
c corrosion model for the inner barrier, for the current package.
c
  do i = 1, 8
    Coef1(i) = gran()
  end do

c
c Find the average values over the package for the parameters for
c humid-air corrosion of the outer barrier. Linear transformations
c of the normal variables to get the average values of the corrosion
c parameters based on the means and covariance structures provided
c at the beginning of this program. Include StanDev to represent
c the portion of the variability from the epsilon term in the
c corrosion model which is allocated to package to package
c variability (see User's Guide).
c
  Coef1(4) = Mean(4) + V5(4, 1) * Coef1(1) + V5(4, 2) * Coef1(2) +
&          V5(4, 3) * Coef1(3) + V5(4, 4) * Coef1(4)
  Coef1(3) = Mean(3) + V5(3, 1) * Coef1(1) + V5(3, 2) * Coef1(2) +
&          V5(3, 3) * Coef1(3)
  Coef1(2) = Mean(2) + V5(2, 1) * Coef1(1) + V5(2, 2) * Coef1(2)
  Coef1(1) = Mean(1) + V5(1, 1) * Coef1(1) + StanDev(1, 1) * gran()

c
c Find the average values over the package for the parameters for
c aqueous corrosion of the outer barrier. Aqueous corrosion is done
c the same way as humid air was above.
c
  Coef1(8) = Mean(8) + V5(8, 1) * Coef1(5) + V5(8, 2) * Coef1(6) +
&          V5(8, 3) * Coef1(7) + V5(8, 4) * Coef1(8)
  Coef1(7) = Mean(7) + V5(7, 1) * Coef1(5) + V5(7, 2) * Coef1(6) +
&          V5(7, 3) * Coef1(7)
  Coef1(6) = Mean(6) + V5(6, 1) * Coef1(5) + V5(6, 2) * Coef1(6)
  Coef1(5) = Mean(5) + V5(5, 1) * Coef1(5) + StanDev(2, 1) * gran()

c
c Get sample for pit multipliers for the CAM, no drips.
c
  Dist      = DTypeStack(1)
  Parm(1)   = DParmStack(1, 1)
  Parm(2)   = DParmStack(1, 2)
  Parm(3)   = DParmStack(1, 3)
  Parm(4)   = DParmStack(1, 4)

```

```

CALL InitDPArray(NDCritDep, NPitCAM)
CALL LHSSamp(Dist, Parm, NPitCAM, CDFStack, NDCritDep)
c
c If using the exponential pit rate model, find the package mean C0 and
c get NPitCAM values for C0 for non-dripped-on pits on the CRM.
c 1. Use 1/2 the variance for C0 to find the package mean (c9).
c 2. Find 1/4 the variance, to be used when patch mean (v9) is later
c found and now when the C0 distribution(to be used for all patches)
c is sampled. This sample can be found with mean = 0 and is
c theoretically translated to the patch mean (in practice, such
c translation is not necessary).
c Retrieve the value for C1 and store in c10. Finally, find the power
c of time to be used for this package (PwrIB).
c
if(MSpec(2) .eq. CRMEXP) then
  CALL InitDPArray(COPak, NPitCRM)
  if(DTypeStack(EndStack(1) + 1) .eq. DSTNOR) then
    Dist = DTypeStack(EndStack(1) + 1)
    Parm(1) = DParmStack(EndStack(1) + 1, 1)
    Parm(2) = DParmStack(EndStack(1) + 1, 2)
    Parm(3) = DParmStack(EndStack(1) + 1, 3)
    Parm(4) = DParmStack(EndStack(1) + 1, 4)
    Parm(2) = Parm(2) / sqrt(2.D0)
    C9 = Parm(1) + gran() * Parm(2)
    Parm(2) = Parm(2) / sqrt(2.D0)
    Parm(1) = 0.D0
    V9 = Parm(2)
    CALL LHSSamp(Dist, Parm, NPitCRM, CDFStack, COPak)
    do i = 1, NPitCRM
      COPak(i) = EXP(COPak(i))
    end do
  else if(DTypeStack(EndStack(1) + 1) .eq. DSTFXD) then
    C9 = DParmStack(EndStack(1) + 1, 1)
    V9 = 0.D0
    do i = 1, NPitCRM
      COPak(i) = 1.D0
    end do
  end if
  C10 = DParmStack(EndStack(1) + 2, 1)
  ranval = ranmz()
  Dist = DTypeStack(EndStack(1) + 3)
  Parm(1) = DParmStack(EndStack(1) + 3, 1)
  Parm(2) = DParmStack(EndStack(1) + 3, 2)
  Parm(3) = DParmStack(EndStack(1) + 3, 3)
  Parm(4) = DParmStack(EndStack(1) + 3, 4)
  PwrIB = SampleValue(ranval, Dist, Parm, CDFStack)
c
else if(MSpec(2) .eq. CRMGONLY) then
c

```



```

c The NO-DRIP general corrosion rate parameters for the package are
c found using the covariance, the variance error, and the share of
c the variance for the packages.
c This treatment currently assumes the distributions for LnA and K are
c normal, and stored in Parm(1) and Parm(2) will be their means. Their
c standard deviations have been folded into the vg array. DL9999
c
    Parm(1) = DParmStack(EndStack(1) + 1, 1)
    Parm(2) = DParmStack(EndStack(1) + 2, 1)
    j      = 1
    k      = 1
    CALL GetCRMRateParms(vg, NDPakLnA, NDPakK, Parm, j, k)
c
    else if(MSpec(2) .eq. CRMGRONLY) then
c
c The NO-DRIP general corrosion rate for package is to be interpolated
c from sampled values at different temperatures. Sample a untransformed
c value for each given temperature using the variance share.
c
    CALL GetCRMPakRates(varshare, NumCRMD(1), NDPakR)
    end if
c
c Obtain a value for the temperature threshold for corrosion initiation
c for non-dripped-on patches for this package.
c
    ranval = ranmz()
    Dist   = DTypeStack(EndStack(2) + 1)
    Parm(1) = DParmStack(EndStack(2) + 1, 1)
    Parm(2) = DParmStack(EndStack(2) + 1, 2)
    Parm(3) = DParmStack(EndStack(2) + 1, 3)
    Parm(4) = DParmStack(EndStack(2) + 1, 4)
    NDTpVal = SampleValue(ranval, Dist, Parm, CDFStack)
    NDTpVal = NDTpVal + 273.15D0
c
c Obtain a value for the RH at which humid-air corrosion initiation is
c possible, for non-dripped-on patches for this package.
c
    ranval = ranmz()
    Dist   = DTypeStack(EndStack(2) + 2)
    Parm(1) = DParmStack(EndStack(2) + 2, 1)
    Parm(2) = DParmStack(EndStack(2) + 2, 2)
    Parm(3) = DParmStack(EndStack(2) + 2, 3)
    Parm(4) = DParmStack(EndStack(2) + 2, 4)
    NDHStart = SampleValue(ranval, Dist, Parm, CDFStack)
c
c Obtain a value for the RH at which aqueous corrosion initiation is
c possible, for non-dripped-on patches for this package. It is related
c to the humid-air corrosion initiation RH via the correlation specified
c by the user.
    
```

```

c      if(NDFeat(1) .ge. 1.D0) then
c      ranval = ranmz()
c      Dist = DTypeStack(EndStack(2) + 3)
c      Parm(1) = DParmStack(EndStack(2) + 3, 1)
c      Parm(2) = DParmStack(EndStack(2) + 3, 2)
c      Parm(3) = DParmStack(EndStack(2) + 3, 3)
c      Parm(4) = DParmStack(EndStack(2) + 3, 4)
c      NDAquaX = SampleValue(ranval, Dist, Parm, CDFStack)
c      end if

c      If some patches are potentially dripped on with NEUTRAL water, sample
c      the package-level properties for the dripped-on patches.
c
c      if(HaveNeut) then
c
c      Get NPitCAM pit multiples for dripped-on pits.
c
c      Dist = DTypeStack(EndStack(3) + 1)
c      Parm(1) = DParmStack(EndStack(3) + 1, 1)
c      Parm(2) = DParmStack(EndStack(3) + 1, 2)
c      Parm(3) = DParmStack(EndStack(3) + 1, 3)
c      Parm(4) = DParmStack(EndStack(3) + 1, 4)
c      CALL LHSSamp(Dist, Parm, NPitCAM, CDFStack, GDCritDep)

c
c      The NEUTRAL-DRIP general corrosion rate parameters for the
c      package are found using the covariance, the variance error, and the
c      share of the variance for the packages.
c      This treatment currently assumes the distributions for LnA and K are
c      normal, and stored in Parm(1) and Parm(2) will be their means. Their
c      standard deviations have been folded into the vg array. DL9999
c
c      if(MSpec(4) .eq. CRMGONLY .or. MSpec(4) .eq. CRMGPGL .or.
c      & MSpec(4) .eq. CRMGLOG .or. MSpec(4) .eq. CRMGLNL) then
c      Parm(1) = DParmStack(EndStack(4) + 1, 1)
c      Parm(2) = DParmStack(EndStack(4) + 2, 1)
c      j = 1
c      k = 2
c      CALL GetCRMRateParms(vg, GDPakLnA, GDPakK, Parm, j, k)
c      k = 3

c
c      The NEUTRAL-DRIP general corrosion rate for package is to be
c      interpolated from sampled values at different temperatures. Sample a
c      untransformed value for each temperature using the variance share.
c
c      else if(MSpec(4) .eq. CRMGRONLY .or. MSpec(4) .eq. CRMGRPGL .or.
c      & MSpec(4) .eq. CRMGRLOG .or. MSpec(4) .eq. CRMGRNL) then
c      CALL GetCRMPakRates(varshare, NumCRMD(2), GDPakR)
c      k = NumCRMD(2) + 1

```

```
      end if
c
c  Get pit properties for CRM, NEUTRAL drips, if using one of the pit
c  growth laws, each requires two to three parameters.
c
      if(MSpec(4) .eq. CRMGPGL .or. MSpec(4) .eq. CRMGRPGL) then
c
c  Get NPitCRM values for B and n (for B^n term) and randomly pair them.
c  Store in the first two GDCRM arrays.
c
      Dist      = DTypeStack(EndStack(4) + k)
      Parm(1) = DParmStack(EndStack(4) + k, 1)
      Parm(2) = DParmStack(EndStack(4) + k, 2)
      Parm(3) = DParmStack(EndStack(4) + k, 3)
      Parm(4) = DParmStack(EndStack(4) + k, 4)
      CALL LHSSamp(Dist, Parm, NPitCRM, CDFStack, GDCRM1)
      k        = k + 1
      Dist      = DTypeStack(EndStack(4) + k)
      Parm(1) = DParmStack(EndStack(4) + k, 1)
      Parm(2) = DParmStack(EndStack(4) + k, 2)
      Parm(3) = DParmStack(EndStack(4) + k, 3)
      Parm(4) = DParmStack(EndStack(4) + k, 4)
      CALL LHSSamp(Dist, Parm, NPitCRM, CDFStack, GDCRM2)
      CALL RandMix(GDCRM2, NPitCRM)
c
      else if(MSpec(4) .eq. CRMGLOG .or. MSpec(4) .eq. CRMGRLOG) then
c
c  Get NPitCRM values for k, x0, and Q (for k*log(t)*exp(Q/T) + x0) and
c  randomly pair them. Note order k, x0, Q into GDCRM1, 2, 3.
c
      Dist      = DTypeStack(EndStack(4) + k)
      Parm(1) = DParmStack(EndStack(4) + k, 1)
      Parm(2) = DParmStack(EndStack(4) + k, 2)
      Parm(3) = DParmStack(EndStack(4) + k, 3)
      Parm(4) = DParmStack(EndStack(4) + k, 4)
      CALL LHSSamp(Dist, Parm, NPitCRM, CDFStack, GDCRM1)
      k        = k + 1
      Dist      = DTypeStack(EndStack(4) + k)
      Parm(1) = DParmStack(EndStack(4) + k, 1)
      Parm(2) = DParmStack(EndStack(4) + k, 2)
      Parm(3) = DParmStack(EndStack(4) + k, 3)
      Parm(4) = DParmStack(EndStack(4) + k, 4)
      CALL LHSSamp(Dist, Parm, NPitCRM, CDFStack, GDCRM2)
      CALL RandMix(GDCRM2, NPitCRM)
      k        = k + 1
      Dist      = DTypeStack(EndStack(4) + k)
      Parm(1) = DParmStack(EndStack(4) + k, 1)
      Parm(2) = DParmStack(EndStack(4) + k, 2)
      Parm(3) = DParmStack(EndStack(4) + k, 3)
```

```

    Parm(4) = DParmStack(EndStack(4) + k, 4)
    CALL LHSSamp(Dist, Parm, NPitCRM, CDFStack, GDCRM3)
    CALL RandMix(GDCRM3, NPitCRM)
    GoodLaw = .true.

c
    else if(MSpec(4) .eq. CRMGLNL .or. MSpec(4) .eq. CRMGRNL) then
c
c   Get NPitCRM values for A, K, and n (for exp(A - K/T) t**n term) and
c   randomly pair them. Note order A, k, N into GDCRM1, 2, 3.
c
    Dist = DTypeStack(EndStack(4) + k)
    Parm(1) = DParmStack(EndStack(4) + k, 1)
    Parm(2) = DParmStack(EndStack(4) + k, 2)
    Parm(3) = DParmStack(EndStack(4) + k, 3)
    Parm(4) = DParmStack(EndStack(4) + k, 4)
    CALL LHSSamp(Dist, Parm, NPitCRM, CDFStack, GDCRM1)
    k = k + 1
    Dist = DTypeStack(EndStack(4) + k)
    Parm(1) = DParmStack(EndStack(4) + k, 1)
    Parm(2) = DParmStack(EndStack(4) + k, 2)
    Parm(3) = DParmStack(EndStack(4) + k, 3)
    Parm(4) = DParmStack(EndStack(4) + k, 4)
    CALL LHSSamp(Dist, Parm, NPitCRM, CDFStack, GDCRM2)
    CALL RandMix(GDCRM2, NPitCRM)
    k = k + 1
    Dist = DTypeStack(EndStack(4) + k)
    Parm(1) = DParmStack(EndStack(4) + k, 1)
    Parm(2) = DParmStack(EndStack(4) + k, 2)
    Parm(3) = DParmStack(EndStack(4) + k, 3)
    Parm(4) = DParmStack(EndStack(4) + k, 4)
    CALL LHSSamp(Dist, Parm, NPitCRM, CDFStack, GDCRM3)
    CALL RandMix(GDCRM3, NPitCRM)
    GoodLaw = .true.
end if

c
c   Obtain a value for the temperature threshold for corrosion initiation
c   for dripped-on patches on the CAM for this package.
c
    ranval = ranmz()
    Dist = DTypeStack(EndStack(5) + 1)
    Parm(1) = DParmStack(EndStack(5) + 1, 1)
    Parm(2) = DParmStack(EndStack(5) + 1, 2)
    Parm(3) = DParmStack(EndStack(5) + 1, 3)
    Parm(4) = DParmStack(EndStack(5) + 1, 4)
    GDTpVal = SampleValue(ranval, Dist, Parm, CDFStack)
    GDTpVal = GDTpVal + 273.15D0

c
c   Obtain a value for the temperature threshold for pitting corrosion
c   initiation on the CRM for dripped-on patches for this package.

```

```

c
    ranval = ranmz()
    Dist   = DTypeStack(EndStack(5) + 2)
    Parm(1) = DParmStack(EndStack(5) + 2, 1)
    Parm(2) = DParmStack(EndStack(5) + 2, 2)
    Parm(3) = DParmStack(EndStack(5) + 2, 3)
    Parm(4) = DParmStack(EndStack(5) + 2, 4)
    GDPInit = SampleValue(ranval, Dist, Parm, CDFStack)
    GDPInit = GDPInit + 273.15D0

c
c Obtain a value for the RH at which humid-air corrosion initiation is
c possible, for dripped-on patches for this package.
c
    ranval = ranmz()
    Dist   = DTypeStack(EndStack(5) + 3)
    Parm(1) = DParmStack(EndStack(5) + 3, 1)
    Parm(2) = DParmStack(EndStack(5) + 3, 2)
    Parm(3) = DParmStack(EndStack(5) + 3, 3)
    Parm(4) = DParmStack(EndStack(5) + 3, 4)
    GDHStart = SampleValue(ranval, Dist, Parm, CDFStack)

c
c Obtain a value for the RH at which aqueous corrosion initiation is
c possible, for dripped-on patches for this package. It is related
c to the humid-air corrosion initiation RH via the correlation specified
c by the user.
c
    if(GDFeat(1) .ge. 1.D0) then
c
        ranval = ranmz()
        Dist   = DTypeStack(EndStack(5) + 4)
        Parm(1) = DParmStack(EndStack(5) + 4, 1)
        Parm(2) = DParmStack(EndStack(5) + 4, 2)
        Parm(3) = DParmStack(EndStack(5) + 4, 3)
        Parm(4) = DParmStack(EndStack(5) + 4, 4)
        GDAquaX = SampleValue(ranval, Dist, Parm, CDFStack)
        end if
    end if

c
c If some patches are potentially dripped on with NON-NEUTRAL water,
c sample the package-level properties for the dripped-on patches.
c
    if(HaveChem) then
        if(HaveNeut) then
            StackIt = 6
            ModNdx = 5
        else
            StackIt = 3
            ModNdx = 3
        end if
    end if
c

```

```
c  Get pit properties for CAM, NON-NEUTRAL drips, which must include
c  either the pit growth law or pit multiples.
c
      if(MSpec(ModNdx) .eq. CAMGPGL) then
c
c  Get NPitCAM values for B and n (for B^n term) and randomly pair them.
c  Skip over the distribution for general corrosion rate, it will be
c  sampled for each patch.
c
      Dist      = DTypeStack(EndStack(StackIt) + 1)
      Parm(1)   = DParmStack(EndStack(StackIt) + 1, 1)
      Parm(2)   = DParmStack(EndStack(StackIt) + 1, 2)
      Parm(3)   = DParmStack(EndStack(StackIt) + 1, 3)
      Parm(4)   = DParmStack(EndStack(StackIt) + 1, 4)
      CALL LHSSamp(Dist, Parm, NPitCAM, CDFStack, BDCAMB)
      Dist      = DTypeStack(EndStack(StackIt) + 2)
      Parm(1)   = DParmStack(EndStack(StackIt) + 2, 1)
      Parm(2)   = DParmStack(EndStack(StackIt) + 2, 2)
      Parm(3)   = DParmStack(EndStack(StackIt) + 2, 3)
      Parm(4)   = DParmStack(EndStack(StackIt) + 2, 4)
      CALL LHSSamp(Dist, Parm, NPitCAM, CDFStack, BDCAMn)
      CALL RandMix(BDCAMn, NPitCAM)
c
      else if(MSpec(ModNdx) .eq. CAMGPM) then
c
c  Get NPitCAM pit multiples for dripped-on pits.
c
      Dist      = DTypeStack(EndStack(StackIt) + 1)
      Parm(1)   = DParmStack(EndStack(StackIt) + 1, 1)
      Parm(2)   = DParmStack(EndStack(StackIt) + 1, 2)
      Parm(3)   = DParmStack(EndStack(StackIt) + 1, 3)
      Parm(4)   = DParmStack(EndStack(StackIt) + 1, 4)
      CALL LHSSamp(Dist, Parm, NPitCAM, CDFStack, BDCritDep)
      end if
      StackIt = StackIt + 1
      ModNdx  = ModNdx + 1
c
c  The NON-NEUTRAL-DRIP general corrosion rate parameters for the
c  package are found using the covariance, the variance error, and the
c  share of the variance for the packages, and is bounded above.
c  This treatment currently assumes the distributions for LnA and K are
c  normal, and stored in Parm(1) and Parm(2) will be their means. Their
c  standard deviations have been folded into the vg array. DL9999
c
      if(MSpec(ModNdx) .eq. CRMGONLY .or.
&      MSpec(ModNdx) .eq. CRMGPGL .or.
&      MSpec(ModNdx) .eq. CRMGLOG .or.
&      MSpec(ModNdx) .eq. CRMGLNL) then
      Parm(1) = DParmStack(EndStack(StackIt) + 1, 1)
```

```

      Parm(2) = DParmStack(EndStack(StackIt) + 2, 1)
      j       = 1
      k       = 3
      CALL GetCRMRateParms(vg, BDPakLnA, BDPakK, Parm, j, k)
      k = 3

C
C The NON-NEUTRAL-DRIP general corrosion rate for package is to be
C interpolated from sampled values at different temperatures. Sample a
C untransformed value for each temperature using the variance share.
C
      elseif(MSpec(ModNdx) .eq. CRMGRONLY .or.
&          MSpec(ModNdx) .eq. CRMGRPGL .or.
&          MSpec(ModNdx) .eq. CRMGRLOG .or.
&          MSpec(ModNdx) .eq. CRMGRNLN) then
      CALL GetCRMPakRates(varshare, NumCRMD(3), BDPakR)
      k = NumCRMD(3) + 1
      end if

C
C Get pit properties for CRM, NON-NEUTRAL drips which must include one
C of the pit growth laws, each requires two or three parameters.
C
      if(MSpec(ModNdx) .eq. CRMGPGL .or. MSpec(ModNdx) .eq. CRMGRPGL) then

C
C Get NPitCRM values for B and n (for B t^n term) & randomly pair them.
C
      Dist = DTypeStack(EndStack(StackIt) + k)
      Parm(1) = DParmStack(EndStack(StackIt) + k, 1)
      Parm(2) = DParmStack(EndStack(StackIt) + k, 2)
      Parm(3) = DParmStack(EndStack(StackIt) + k, 3)
      Parm(4) = DParmStack(EndStack(StackIt) + k, 4)
      CALL LHSSamp(Dist, Parm, NPitCRM, CDFStack, BDCRM1)
      k = k + 1
      Dist = DTypeStack(EndStack(StackIt) + k)
      Parm(1) = DParmStack(EndStack(StackIt) + k, 1)
      Parm(2) = DParmStack(EndStack(StackIt) + k, 2)
      Parm(3) = DParmStack(EndStack(StackIt) + k, 3)
      Parm(4) = DParmStack(EndStack(StackIt) + k, 4)
      CALL LHSSamp(Dist, Parm, NPitCRM, CDFStack, BDCRM2)
      CALL RandMix(BDCRM2, NPitCRM)

C
      elseif(MSpec(ModNdx) .eq. CRMGLOG .or.
&          MSpec(ModNdx) .eq. CRMGRLOG) then

C
C Get NPitCRM values for k, x0, and Q (for k*log(t)*exp(Q/T) + x0) and
C randomly pair them. Note order k, x0, Q into BDCRM1, 2, 3.
C
      Dist = DTypeStack(EndStack(StackIt) + k)
      Parm(1) = DParmStack(EndStack(StackIt) + k, 1)
      Parm(2) = DParmStack(EndStack(StackIt) + k, 2)

```

```

Parm(3) = DParmStack(EndStack(StackIt) + k, 3)
Parm(4) = DParmStack(EndStack(StackIt) + k, 4)
CALL LHSSamp(Dist, Parm, NPitCRM, CDFStack, BDCRM1)
k      = k + 1
Dist   = DTypeStack(EndStack(StackIt) + k)
Parm(1) = DParmStack(EndStack(StackIt) + k, 1)
Parm(2) = DParmStack(EndStack(StackIt) + k, 2)
Parm(3) = DParmStack(EndStack(StackIt) + k, 3)
Parm(4) = DParmStack(EndStack(StackIt) + k, 4)
CALL LHSSamp(Dist, Parm, NPitCRM, CDFStack, BDCRM2)
CALL RandMix(BDCRM2, NPitCRM)
k      = k + 1
Dist   = DTypeStack(EndStack(StackIt) + k)
Parm(1) = DParmStack(EndStack(StackIt) + k, 1)
Parm(2) = DParmStack(EndStack(StackIt) + k, 2)
Parm(3) = DParmStack(EndStack(StackIt) + k, 3)
Parm(4) = DParmStack(EndStack(StackIt) + k, 4)
CALL LHSSamp(Dist, Parm, NPitCRM, CDFStack, BDCRM3)
CALL RandMix(BDCRM3, NPitCRM)
BadLaw = .true.

```

```

c
elseif(MSpec(ModNdx).eq.CRMGLNL.or.

```

```

&
MSpec(ModNdx).eq.CRMGRLNL)then

```

```

c
c Get NPitCRM values for A, k, and n (for exp(A - K/T) t**n term) and
c randomly pair them. Note order A, k, n into BDCRM1, 2, 3.
c

```

```

Dist   = DTypeStack(EndStack(StackIt) + k)
Parm(1) = DParmStack(EndStack(StackIt) + k, 1)
Parm(2) = DParmStack(EndStack(StackIt) + k, 2)
Parm(3) = DParmStack(EndStack(StackIt) + k, 3)
Parm(4) = DParmStack(EndStack(StackIt) + k, 4)
CALL LHSSamp(Dist, Parm, NPitCRM, CDFStack, BDCRM1)
k      = k + 1
Dist   = DTypeStack(EndStack(StackIt) + k)
Parm(1) = DParmStack(EndStack(StackIt) + k, 1)
Parm(2) = DParmStack(EndStack(StackIt) + k, 2)
Parm(3) = DParmStack(EndStack(StackIt) + k, 3)
Parm(4) = DParmStack(EndStack(StackIt) + k, 4)
CALL LHSSamp(Dist, Parm, NPitCRM, CDFStack, BDCRM2)
CALL RandMix(BDCRM2, NPitCRM)
k      = k + 1
Dist   = DTypeStack(EndStack(StackIt) + k)
Parm(1) = DParmStack(EndStack(StackIt) + k, 1)
Parm(2) = DParmStack(EndStack(StackIt) + k, 2)
Parm(3) = DParmStack(EndStack(StackIt) + k, 3)
Parm(4) = DParmStack(EndStack(StackIt) + k, 4)
CALL LHSSamp(Dist, Parm, NPitCRM, CDFStack, BDCRM3)
CALL RandMix(BDCRM3, NPitCRM)

```



```
      BadLaw = .true.  
    end if  
    StackIt = StackIt + 1
```

```
c  
c Obtain a value for the temperature threshold for corrosion initiation  
c for dripped-on patches for this package.  
c
```

```
      ranval = ranmz()  
      Dist = DTypeStack(EndStack(StackIt) + 1)  
      Parm(1) = DParmStack(EndStack(StackIt) + 1, 1)  
      Parm(2) = DParmStack(EndStack(StackIt) + 1, 2)  
      Parm(3) = DParmStack(EndStack(StackIt) + 1, 3)  
      Parm(4) = DParmStack(EndStack(StackIt) + 1, 4)  
      BDTPVal = SampleValue(ranval, Dist, Parm, CDFStack)  
      BDTPVal = BDTPVal + 273.15D0
```

```
c  
c Obtain a value for the temperature threshold for pitting corrosion  
c initiation on the CRM for dripped-on patches for this package.  
c
```

```
      ranval = ranmz()  
      Dist = DTypeStack(EndStack(StackIt) + 2)  
      Parm(1) = DParmStack(EndStack(StackIt) + 2, 1)  
      Parm(2) = DParmStack(EndStack(StackIt) + 2, 2)  
      Parm(3) = DParmStack(EndStack(StackIt) + 2, 3)  
      Parm(4) = DParmStack(EndStack(StackIt) + 2, 4)  
      BDPInit = SampleValue(ranval, Dist, Parm, CDFStack)  
      BDPInit = BDPInit + 273.15D0
```

```
c  
c Obtain a value for the RH at which humid-air corrosion initiation is  
c possible, for dripped-on patches for this package.  
c
```

```
      ranval = ranmz()  
      Dist = DTypeStack(EndStack(StackIt) + 3)  
      Parm(1) = DParmStack(EndStack(StackIt) + 3, 1)  
      Parm(2) = DParmStack(EndStack(StackIt) + 3, 2)  
      Parm(3) = DParmStack(EndStack(StackIt) + 3, 3)  
      Parm(4) = DParmStack(EndStack(StackIt) + 3, 4)  
      BDHStart = SampleValue(ranval, Dist, Parm, CDFStack)
```

```
c  
c Obtain a value for the RH at which aqueous corrosion initiation is  
c possible, for dripped-on patches for this package. It is related  
c to the humid-air corrosion initiation RH via the correlation specified  
c by the user.  
c
```

```
      if(BDFeat(1) .ge. 1.D0) then
```

```
c  
      ranval = ranmz()  
      Dist = DTypeStack(EndStack(StackIt) + 4)  
      Parm(1) = DParmStack(EndStack(StackIt) + 4, 1)  
      Parm(2) = DParmStack(EndStack(StackIt) + 4, 2)
```

```
      Parm(3) = DParmStack(EndStack(StackIt) + 4, 3)
      Parm(4) = DParmStack(EndStack(StackIt) + 4, 4)
      BDAquaX = SampleValue(ranval, Dist, Parm, CDFStack)
    end if
  end if
```

c

c If dripping is a possibility, sample the time for onset of drips.

c Then if water chemistry changes, sample time of change.

c

```
  if(SomeDrips) then
    ranval = ranmz()
    Dist = DripStartType
    Parm(1) = DripStartPar(1)
    Parm(2) = DripStartPar(2)
    Parm(3) = DripStartPar(3)
    Parm(4) = DripStartPar(4)
    OnsetDrip = SampleValue(ranval, Dist, Parm, CDFStack)
    ranval = ranmz()
    Dist = DripEndType
    Parm(1) = DripEndPar(1)
    Parm(2) = DripEndPar(2)
    Parm(3) = DripEndPar(3)
    Parm(4) = DripEndPar(4)
    StopDrip = SampleValue(ranval, Dist, Parm, CDFStack)
    if(StopDrip .le. OnsetDrip) then
      OnsetDrip = 2.D0 * binf
      StopDrip = OnsetDrip
      ChangeDrip = OnsetDrip
    else
      if(NewDrips) then
        ranval = ranmz()
        Dist = DripChnType
        Parm(1) = DripChnPar(1)
        Parm(2) = DripChnPar(2)
        Parm(3) = DripChnPar(3)
        Parm(4) = DripChnPar(4)
        ChangeDrip = SampleValue(ranval, Dist, Parm, CDFStack)
        if(ChangeDrip .lt. OnsetDrip) then
          ChangeDrip = OnsetDrip
        else if(ChangeDrip .gt. StopDrip) then
          ChangeDrip = 2.D0 * binf
        end if
      else
        ChangeDrip = 2.D0 * binf
      end if
    end if
  else
    OnsetDrip = 2.D0 * binf
    StopDrip = OnsetDrip
```

```

      ChangeDrip = OnsetDrip
    end if

```

```

c
c Initialize penetration time variables, Bin arrays, and CRM array.
c

```

```

      BinStart = binf
      PatStart = binf
      do i = 1, nbin + 1
        Bin(i, 1) = 0
        Bin(i, 2) = 0
        Bin(i, 3) = 0
        PatBin(i, 1) = 0
        PatBin(i, 2) = 0
        PatBin(i, 3) = 0
        PatBin(i, 4) = 0
        PatBin(i, 5) = 0
      end do
      do i = 1, 5
        Pit1st(i) = binf
        Pat1st(i) = binf
      end do
      CALL InitDPArray(CRMThk, MAXCRMT)

```

```

c
c The references to GoodLaw and BadLaw herein indicate whether one the
c pit laws applies to the CRM in the presence of good drips and bad
c drips. Currently we force the pit law parameters to be identical
c for both kinds of drips.
c

```

```

      if(GoodLaw .and. BadLaw) then
        do i = 1, NPitCRM
          BDCRM1(i) = GDCRM1(i)
          BDCRM2(i) = GDCRM2(i)
          BDCRM3(i) = GDCRM3(i)
        end do
      end if

```

```

c
c Write package properties to the auxilliary file. Column labels are
c written if this is the first package.
c

```

```

      if(.not. SomeDrips) then
        if(MSpec(2) .eq. CRMEXP) then
          if(Pack .eq. 1) write(auxfid, 7001)
          write(auxfid, 9000) Pack, NDTpVal, NDHASTart, NDAquaX,
&              (Coef1(i), i = 1, 8), c9, PwrIB
          else
            if(Pack .eq. 1) write(auxfid, 7002)
            write(auxfid, 9005) Pack, NDTpVal, NDHASTart, NDAquaX,
&              NDPakLnA, NDPakK, (Coef1(i), i = 1, 8)
            end if

```

```

else
  if(.not. NewDrips) then
    if(HaveNeut) then
      if(Pack .eq. 1) write(auxfid, 7011)
      write(auxfid, 9010) Pack, NDTpVal, NDHASTart, NDAquaX,
&      NDPakLnA, NDPakK, OnsetDrip, GDTpVal,
&      GDPInit, GDHASTart, GDAquaX, GDPakLnA,
&      GDPakK, StopDrip, (Coef1(i), i = 1, 8)
    else
      if(Pack .eq. 1) write(auxfid, 7012)
      write(auxfid, 9010) Pack, NDTpVal, NDHASTart, NDAquaX,
&      NDPakLnA, NDPakK, OnsetDrip, BDTpVal,
&      BDPInit, BDHASTart, BDAquaX, BDPakLnA,
&      BDPakK, StopDrip, (Coef1(i), i = 1, 8)
    end if
  else
    if(NeutD1) then
      if(Pack .eq. 1) write(auxfid, 7021)
      write(auxfid, 9020) Pack, NDTpVal, NDHASTart, NDAquaX,
&      NDPakLnA, NDPakK, OnsetDrip, GDTpVal,
&      GDPInit, GDHASTart, GDAquaX, GDPakLnA,
&      GDPakK, ChangeDrip, BDTpVal, BDPInit,
&      BDHASTart, BDAquaX, BDPakLnA, BDPakK,
&      StopDrip, (Coef1(i), i = 1, 8)
    else
      if(Pack .eq. 1) write(auxfid, 7022)
      write(auxfid, 9020) Pack, NDTpVal, NDHASTart, NDAquaX,
&      NDPakLnA, NDPakK, OnsetDrip, BDTpVal,
&      BDPInit, BDHASTart, BDAquaX, BDPakLnA,
&      BDPakK, ChangeDrip, GDTpVal, GDPInit,
&      GDHASTart, GDAquaX, GDPakLnA, GDPakK,
&      StopDrip, (Coef1(i), i = 1, 8)
    end if
  end if
end if

```

```

C
7001 format(/, ' Package Properties',/,109('='),/,
& ' Pack [      No Drips      ] [      Humid-Air Coefficients      ]',
& ' [      Aqueous Coefficients      ] [ CRM Pitting ]',/,
& ' No TmpThr HASTrt AqStrt  a0      a1      a2      a3 ',
& ' b0      b1      b2      b3      MeanC0      n ',/,
& 109('='))
7002 format(/, ' Package Properties',/,108('='),/,
& ' Pack [      No Drips      ]',
& ' [      Humid-Air Coefficients      ]',
& ' [      Aqueous Coefficients      ]',/,
& ' No TmpThr HASTrt AqStrt CRMLnA CRMk ',
& ' a0      a1      a2      a3 ',
& ' b0      b1      b2      b3      ',/,108('='))

```

```

9000 format(i4,1x,3(1x,f6.2),1x,
&         f6.3,1x,f8.2,1x,f8.2,1x,f7.5,1x,f6.2,1x,f7.5,1x,f7.0,1x,
&         lpe10.3,1x,0pf7.3,1x,0pf6.4)
9005 format(i4,1x,3(1x,f6.2),1x,f6.3,1x,f6.1,1x,
&         f6.3,1x,f8.2,1x,f8.2,1x,f7.5,1x,f6.2,1x,f7.5,1x,f7.0,1x,
&         lpe10.3)
7011 format(/,' Package Properties',/,168('='),/,
& ' Pack [          No Drips          ] Drips ',
& ' [          Neutral Drips          ] Drips ',
& ' [ Humid-Air Coefficients ]',
& ' [ Aqueous Coefficients ]',/,
& ' No TmpThr HASTrt AqStrt CRMLnA CRMk Start ',
& ' TmpThr TmpCRM HASTrt AqStrt CRMLnA CRMk Stop ',
& ' a0      a1      a2      a3 ',
& ' b0      b1      b2      b3 ',/,168('-'))
7012 format(/,' Package Properties',/,168('='),/,
& ' Pack [          No Drips          ] Drips ',
& ' [ NonNeutral Drips          ] Drips ',
& ' [ Humid-Air Coefficients ]',
& ' [ Aqueous Coefficients ]',/,
& ' No TmpThr HASTrt AqStrt CRMLnA CRMk Start ',
& ' TmpThr TmpCRM HASTrt AqStrt CRMLnA CRMk Stop ',
& ' a0      a1      a2      a3 ',
& ' b0      b1      b2      b3 ',/,168('-'))
9010 format(i4,1x,3(1x,f6.2),1x,f6.3,1x,f6.1,1x,f8.0,4(1x,f6.2),1x,
&         f6.3,1x,f6.1,1x,f8.0,1x,
&         f6.3,1x,f8.2,1x,f8.2,1x,f7.5,1x,f6.2,1x,f7.5,1x,f7.0,1x,
&         lpe10.3)
7021 format(/,' Package Properties',/,219('='),/,
& ' Pack [          No Drips          ] Drips ',
& ' [          Neutral Drips          ] Drips ',
& ' [ NonNeutral Drips          ] Drips ',
& ' [ Humid-Air Coefficients ]',
& ' [ Aqueous Coefficients ]',/,
& ' No TmpThr HASTrt AqStrt CRMLnA CRMk Start ',
& ' TmpThr TmpCRM HASTrt AqStrt CRMLnA CRMk Chang ',
& ' TmpThr TmpCRM HASTrt AqStrt CRMLnA CRMk Stop ',
& ' a0      a1      a2      a3 ',
& ' b0      b1      b2      b3 ',/,219('-'))
7022 format(/,' Package Properties',/,219('='),/,
& ' Pack [          No Drips          ] Drips ',
& ' [ NonNeutral Drips          ] Drips ',
& ' [          Neutral Drips          ] Drips ',
& ' [ Humid-Air Coefficients ]',
& ' [ Aqueous Coefficients ]',/,
& ' No TmpThr HASTrt AqStrt CRMLnA CRMk Start ',
& ' TmpThr TmpCRM HASTrt AqStrt CRMLnA CRMk Chang ',
& ' TmpThr TmpCRM HASTrt AqStrt CRMLnA CRMk Stop ',
& ' a0      a1      a2      a3 ',

```

```

&      b0      b1      b2      b3      ',/,219('-'))
9020 format(i4,1x,3(1x,f6.2),1x,f6.3,1x,f6.1,1x,f8.0,4(1x,f6.2),1x,
&      f6.3,1x,f6.1,1x,f8.0,4(1x,f6.2),1x,f6.3,1x,f6.1,1x,f8.0,1x,
&      f6.3,1x,f8.2,1x,f8.2,1x,f7.5,1x,f6.2,1x,f7.5,1x,
&      f7.0,1x,1p10.3)
      RETURN
      END      !SUBROUTINE SetupPackage

```

```

C
C *****
C

```

```

      SUBROUTINE GetPackGrids(outfid, auxfid, SigTmp, SigRH, binf,
&      NeutD1, NeutD2, ErrFound, NumStep, tim,
&      tem, r, Pack, GroupNdx, NDTpVal,
&      NDHASTart, NDAquaX, GDTpVal, GDHASTart,
&      GDAquaX, BDTpVal, BDHASTart, BDAquaX,
&      OnsetDrip, NdxDrip, ChangeDrip, NdxChn,
&      StopDrip, NdxEnd, NDNowVal, NDntim,
&      NDntem, NDnr, YDNowVal, YDntim, YDntem,
&      YDnr, NDTime, YDTimeND1, YDTimeND2,
&      YDTime1, YDTime2, YDTime)

```

```

C
C Determine the temp and RH history for this package using group means.
C Develop no-drip history for this package. Use sampled temperature
C and RH thresholds (ND...) to establish time stepping for corrosion
C from start to finish, and save it(ND...). Then if dripping possible,
C find a history appropriate for dripped on packages, which may use no-
C drip variables, neutral drip variables (GD...), and non-native water
C characteristics (BD...).
C Input : outfid, auxfid, SigTmp, SigRH, binf, NeutD1, NeutD2, NumStep,
C      tim, tem, r, Pack, GroupNdx, NDTpVal, NDHASTart, NDAquaX,
C      GDTpVal, GDHASTart, GDAquaX, BDTpVal, BDHASTart, BDAquaX,
C      OnsetDrip, ChangeDrip, StopDrip
C Output: ErrFound, NdxDrip, NdxChn, NdxEnd, NDNowVal, NDntim, NDntem,
C      NDnr, YDNowVal, YDntim, YDntem, YDnr, NDTime, YDTimeND1,
C      YDTimeND2, YDTime1, YDTime2, YDTime
C Local : i, LastTime, ChnBound, rantmp, ranRH

```

```

C
C Arguments
C

```

```

      implicit none
      logical NeutD1, NeutD2, ErrFound, NDTime, YDTimeND1, YDTimeND2,
&      YDTime1, YDTime2, YDTime
      integer outfid, auxfid, NumStep, Pack, GroupNdx, NdxDrip, NdxChn,
&      NdxEnd, NDNowVal, YDNowVal
      double precision SigTmp(*), SigRH(*), binf, tim(*), tem(*), r(*)
      double precision NDTpVal, NDHASTart, NDAquaX
      double precision GDTpVal, GDHASTart, GDAquaX
      double precision BDTpVal, BDHASTart, BDAquaX
      double precision Onsetdrip, ChangeDrip, StopDrip

```

```

double precision NDntim(*), NDntem(*), NDnr(*)
double precision YDntim(*), YDntem(*), YDnr(*)
c
c Local variable
c
c integer i, LastTime
c double precision ChnBound, rantmp, ranRH, gran
c
c Get two random numbers from a standard normal distribution to be used
c to offset the temp and RH curves from their mean.
c
c rantmp = gran()
c ranRH = gran()
c rantmp = rantmp * SigTmp(GroupNdx)
c ranRH = ranRH * SigRH(GroupNdx)
c
c No drip grid
c
c NDNowVal = NumStep
c do i = 1, NDNowVal
c   NDntim(i) = tim(i)
c   NDntem(i) = tem(i) + rantmp
c   NDnr(i) = r(i) + ranRH
c end do
c LastTime = NDNowVal
c CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
c & NDtpVal, NDHStart, NDAquaX, 1, LastTime,
c & NDNowVal, NDntim, NDntem, NDnr, NDTime)
c if(ErrFound) RETURN
c YDTime = .false.
c YDTimeND1 = .false.
c YDTimeND2 = .false.
c YDTime1 = .false.
c YDTime2 = .false.
c
c if(OnsetDrip .lt. binf) then
c
c Drips start before the end of simulation so we need a drip grid.
c Initialize to the generic grid, possibly with adjusted temp and RH.
c
c YDNowVal = NumStep
c do i = 1, YDNowVal
c   YDntim(i) = tim(i)
c   YDntem(i) = tem(i) + rantmp
c   YDnr(i) = r(i) + ranRH
c end do
c NdxDrip = 1
c
c if(OnsetDrip .gt. 0.D0) then

```

```

c
c Drips begin after time zero, so insert time when drips begin and
c check the corrosion possibilities during the initial no-drip segment.
c
      CALL InsertTime(outfid, auxfid, binf, ErrFound, YDNowVal,
&                      YDntim, YDntem, YDnr, Pack, OnsetDrip,
&                      NdxDrip)
      if(ErrFound) RETURN
      CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
&                      NDTpVal, NDHASTart, NDAquaX, 1,
&                      NdxDrip, YDNowVal, YDntim, YDntem, YDnr,
&                      YDTimeND1)
      if(ErrFound) RETURN
      YDTime = YDTimeND1
      end if
      ChnBound = min(StopDrip, binf)
c
      if(ChangeDrip.gt.OnsetDrip .and. ChangeDrip.lt.ChnBound) then
c
c Water chemistry of drips changes after dripping starts and before
c the end of drips or simulation, partition the time grid into 2 more
c segments: type 1 water segment and type 2 water segment.
c
      CALL InsertTime(outfid, auxfid, binf, ErrFound, YDNowVal,
&                      YDntim, YDntem, YDnr, Pack, ChangeDrip,
&                      NdxChn)
      if(ErrFound) RETURN
      if(NeutD1) then
      CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
&                      GDTpVal, GDHASTart, GDAquaX, NdxDrip,
&                      NdxChn, YDNowVal, YDntim, YDntem, YDnr,
&                      YDTime1)
      if(ErrFound) RETURN
      else
      CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
&                      BDTpVal, BDHASTart, BDAquaX, NdxDrip,
&                      NdxChn, YDNowVal, YDntim, YDntem, YDnr,
&                      YDTime1)
      if(ErrFound) RETURN
      end if
      if(.not. YDTime) YDTime = YDTime1
c
c Now do the next time segment using the new water properties and the
c last nondripping segment, if present.
c
      if(StopDrip .lt. binf) then
      CALL InsertTime(outfid, auxfid, binf, ErrFound, YDNowVal,
&                      YDntim, YDntem, YDnr, Pack, StopDrip,
&                      NdxEnd)

```



```

        if(ErrFound) RETURN
        if(NeutD2) then
            CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
&                                GDTpVal, GDHASTart, GDAquaX, NdxChn,
&                                NdxEnd, YDNowVal, YDntim, YDntem,
&                                YDnr, YDTime2)
            if(ErrFound) RETURN
        else
            CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
&                                BDTpVal, BDHASTart, BDAquaX, NdxChn,
&                                NdxEnd, YDNowVal, YDntim, YDntem,
&                                YDnr, YDTime2)
            if(ErrFound) RETURN
        end if
        if(.not. YDTime) YDTime = YDTime2

```

```

c
c Define and check the nondripping segment.
c

```

```

        LastTime = YDNowVal
        CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
&                                NDTpVal, NDHASTart, NDAquaX, NdxEnd,
&                                LastTime, YDNowVal, YDntim, YDntem,
&                                YDnr, YDTimeND2)
        if(ErrFound) RETURN
        if(.not. YDTime) YDTime = YDTimeND2

```

```

c
c else
c
c Dripping starts and water changes but dripping doesn't stop, so type
c 2 drips persist to end.
c

```

```

        LastTime = YDNowVal
        if(NeutD2) then
            CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
&                                GDTpVal, GDHASTart, GDAquaX, NdxChn,
&                                LastTime, YDNowVal, YDntim, YDntem,
&                                YDnr, YDTime2)
            if(ErrFound) RETURN
        else
            CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
&                                BDTpVal, BDHASTart, BDAquaX, NdxChn,
&                                LastTime, YDNowVal, YDntim, YDntem,
&                                YDnr, YDTime2)
            if(ErrFound) RETURN
        end if
        if(.not. YDTime) YDTime = YDTime2
        NdxEnd = YDNowVal + 1
        end if

```

```

c

```

```

else if(ChangeDrip .ge. ChnBound) then
c
c Water chemistry does not change during dripping from its initial
c condition, dripping stops or simulation ends before change time. So
c adjust time grid from drip onset to end of dripping or end of
c simulation using properties of water type 1.
c
    if(StopDrip .lt. binf) then
        CALL InsertTime(outfid, auxfid, binf, ErrFound, YDNowVal,
            & YDntim, YDntem, YDnr, Pack, StopDrip,
            & NdxEnd)
        if(ErrFound) RETURN
        if(NeutD1) then
            CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
            & GDTpVal, GDHASTart, GDAquaX, NdxDrip,
            & NdxEnd, YDNowVal, YDntim, YDntem,
            & YDnr, YDTime1)
            if(ErrFound) RETURN
        else
            CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
            & BDTpVal, BDHASTart, BDAquaX, NdxDrip,
            & NdxEnd, YDNowVal, YDntim, YDntem,
            & YDnr, YDTime1)
            if(ErrFound) RETURN
        end if
        if(.not. YDTime) YDTime = YDTime1
        LastTime = YDNowVal
        CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
            & NDTpVal, NDHASTart, NDAquaX, NdxEnd,
            & LastTime, YDNowVal, YDntim, YDntem,
            & YDnr, YDTimeEND2)
        if(ErrFound) RETURN
        if(.not. YDTime) YDTime = YDTimeEND2
    else
c
c Dripping does not change and does not stop, so use initial drips to
c end of simulation.
c
        LastTime = YDNowVal
        if(NeutD1) then
            CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
            & GDTpVal, GDHASTart, GDAquaX, NdxDrip,
            & LastTime, YDNowVal, YDntim, YDntem,
            & YDnr, YDTime1)
            if(ErrFound) RETURN
        else
            CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
            & BDTpVal, BDHASTart, BDAquaX, NdxDrip,

```

```
&                                LastTime, YDNowVal, YDntim, YDntem,
&                                YDnr, YDTime1)
      if(ErrFound) RETURN
    end if
    if(.not. YDTime) YDTime = YDTime1
    NdxEnd = YDNowVal + 1
  end if
  NdxChn = YDNowVal + 1
c
c   else
c
c   Water chemistry does not change during dripping.  Its condition
c   changed to its final condition at or prior to the start of dripping.
c   So adjust time grid from drip onset to end of simulation using
c   properties of water type 2.
c
  NdxChn = NdxDrip
  if(StopDrip .lt. binf) then
    CALL InsertTime(outfid, auxfid, binf, ErrFound, YDNowVal,
&                                YDntim, YDntem, YDnr, Pack, StopDrip,
&                                NdxEnd)
    if(ErrFound) RETURN
    if(NeutD2) then
      CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
&                                GDTpVal, GDHASTart, GDAquaX, NdxDrip,
&                                NdxEnd, YDNowVal, YDntim, YDntem,
&                                YDnr, YDTime2)
      if(ErrFound) RETURN
    else
      CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
&                                BDTpVal, BDHASTart, BDAquaX, NdxDrip,
&                                NdxEnd, YDNowVal, YDntim, YDntem,
&                                YDnr, YDTime2)
      if(ErrFound) RETURN
    end if
    if(.not. YDTime) YDTime = YDTime2
  else
    LastTime = YDNowVal
    if(NeutD2) then
      CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
&                                GDTpVal, GDHASTart, GDAquaX, NdxDrip,
&                                LastTime, YDNowVal, YDntim, YDntem,
&                                YDnr, YDTime2)
      if(ErrFound) RETURN
    else
      CALL AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
&                                BDTpVal, BDHASTart, BDAquaX, NdxDrip,
&                                LastTime, YDNowVal, YDntim, YDntem,
&                                YDnr, YDTime2)
```

```

        if(ErrFowVal) RETURN
      end if
      NdxEnd = YDNowVal + 1
      if(.not. YDTime) YDTime = YDTime2
    end if
  end if

c
  else
c
c   Drips either not being considered or start to late for this package.
c   So set the YesDrip grid equal to the NoDrip grid.
c
    YDNowVal = NDNowVal
    do i = 1, YDNowVal
      YDnr(i)   = NDnr(i)
      YDntim(i) = NDntim(i)
      YDntem(i) = NDntem(i)
    end do
    NdxDrip = NDNowVal + 1
    NdxChn  = NDNowVal + 1
    NdxEnd  = NDNowVal + 1
  end if
  RETURN
END      !SUBROUTINE GetPackGrids

c
c *****
c
c   SUBROUTINE SetupPatch(v5, StanDev, v9, Penet1, NPitCAM, NPitCRM,
&   TopDripFrc, BotDripFrc, HaveNeut, NeutD1,
&   NeutD2, CermType, CermPar, MSpec,
&   DTypeStack, DParmStack, EndStack, CDFStack,
&   CRMMMax, NumCRMD, varshare, vg, NDFeat,
&   GDFeat, BDFeat, c9, NDPakLnA, NDPakK,
&   NDCRMT, NDPakR, GDPakLnA, GDPakK, GDCRMT,
&   GDPakR, GDPInit, BDPakLnA, BDPakK, BDCRMT,
&   BDPakR, BDPInit, NdxDrip, NdxChn, NdxEnd,
&   Coef1, NDNowVal, NDntim, NDntem, NDnr,
&   YDNowVal, YDntim, YDntem, YDnr, NDTime,
&   YDTimeND1, YDTimeND2, YDTime1, YDTime2,
&   YDTime, PatchType, GPDepth, Coef2, Start,
&   C0PatM, dt, NumSeg, SegStart, SegEnd, Ndx1,
&   Ndx2, Ndx3, DoPatch, CAMType, CRMType,
&   PitInit, NDCritDep, GDCritDep, BDCritDep,
&   MaxPM, BDCAMB, BDCAMn, GDCRM1, GDCRM2,
&   GDCRM3, BDCRM1, BDCRM2, BDCRM3, C0Pak,
&   CAMPit, CRMGen, GenRateM, MxCRMRate,
&   CRMGenR, CRMTemp, CRMNoD, CRMPit, NowVal,
&   ntim, ntem, nr)

```

```

c   For each patch,
c   1. Determine if patch sees drips or pooled water, or not.
c   2. Based on the answer, retrieve the appropriate time grid.
c   3. If ceramic protection, truncate grid at sampled time.
c   4. Create grid-centered history and find start time.
c   5. Sample corrosion parameters based on average values for package.
c   6. Set the properties for each time segment.
c   Input : v5, StanDev, v9, Penet1, NPitCAM, NPitCRM, TopDripFrc,
c           BotDripFrc, HaveNeut, NeutD1, NeutD2, CermType, CermPar,
c           MSpec, DTypeStack, DParmStack, EndStack, CDFStack, CRMMax,
c           NumCRMD, varshare, vg, NDFeat, GDFeat, BDFeat, c9, NDPakLnA,
c           NDPakK, NDCRMT, NDPakR, GDPakLnA, GDPakK, GDCRMT, GDPakR,
c           GDPInit, BDPakLnA, BDPakK, BDCRMT, BDPakR, BDPInit, NdxDrip,
c           NdxChn, NdxEnd, Coef1, NDNowVal, NDntim, NDntem, NDnr,
c           YDNowVal, YDntim, YDntem, YDnr, NDTime, YDTimeND1, YDTimeND2,
c           YDTime1, YDTime2, YDTime, PatchType, NDCritDep, GDCritDep,
c           BDCritDep, BDCAMB, GDCAMn, GDCRM1, GDCRM2, GDCRM3, BDCRM1,
c           BDCRM2, BDCRM3, C0Pak
c   Output: GPDepth, Coef2, Start, C0PatM, dt, NumSeg, SegStart, SegEnd,
c           Ndx1, Ndx2, Ndx3, DoPatch, CAMType, CRMType, PitInit, MaxPM,
c           CAMPit, CRMGen, GenRateM, MxCRMRate, CRMGenR, CRMTemp,
c           CRMNoD, CRMPit, NowVal, ntim, ntem, nr
c   Local : DrippedOn, NDTimeG1, NDTimeG2, YDTime1G, YDTime2G, NoTime,
c           GoodDrip, i, j, k, Dist, StackIt, SegND1, SegW1, SegW2,
c           SegND2, ranval, CermTime, Parm, InvNorm, S, PatLnA, PatK,
c           Rate
c
c   Arguments, and dimension sizes from the include file.
c

```

```

IMPLICIT NONE
INCLUDE 'parm306.inc'
logical HaveNeut, NeutD1, NeutD2
logical NDTime, YDTimeND1, YDTimeND2, YDTime1, YDTime2, YDTime,
&      DoPatch
integer NPitCAM, NPitCRM, CermType, NdxDrip, NdxChn, NdxEnd,
&      NDNowVal, YDNowVal, PatchType, Ndx1, Ndx2, Ndx3, NumSeg,
&      NowVal
integer MSpec(*), DTypeStack(*), EndStack(*)
integer SegStart(*), SegEnd(*), CAMType(*), CRMType(*)
integer NumCRMD(*), CRMNoD(*)
double precision v5(NCOEFF, *), StanDev(NMODEL, 2)
double precision v9, Penet1, TopDripFrc, BotDripFrc, c9, Start,
&      C0PatM
double precision NDPakLnA, GDPakLnA, BDPakLnA
double precision NDPakK, GDPakK, BDPakK
double precision NDPakR(*), GDPakR(*), BDPakR(*)
double precision NDCRMT(*), GDCRMT(*), BDCRMT(*)
double precision GDPInit, BDPInit
double precision NDFeat(*), GDFeat(*), BDFeat(*)

```

```

double precision CermPar(*), Coef1(*)
double precision DParmStack(MAXDISTS, 4), CDFStack(MAXPAIRS, 2)
double precision vg(2, 3, 4), CRMMax(*), varshare
double precision NDntim(*), NDntem(*), NDnr(*)
double precision YDntim(*), YDntem(*), YDnr(*)
double precision GPDepth(*), Coef2(*), dt(*)
double precision NDCritDep(*), GDCritDep(*), BDCritDep(*)
double precision BDCAMB(*), BDCAMn(*), MaxPM(*)
double precision GDCRM1(*), GDCRM2(*), GDCRM3(*)
double precision BDCRM1(*), BDCRM2(*), BDCRM3(*)
double precision COPak(*), PitInit(*), CRMGen(MAXSEG, 2),
&      GenRateM(MAXSEG, 2), MxCRMRate(*)
double precision CRMGenR(MAXSEG, MAXCRMDISTS),
&      CRMTemp(MAXSEG, MAXCRMDISTS)
double precision ntim(*), ntem(*), nr(*)
double precision CAMPit(MAXPITCAM, MAXSEG, 2),
&      CRMPit(MAXPITCRM, MAXSEG, 3)

```

```

c
c Local variables
c

```

```

logical DrippedOn, NDTimeG1, NDTimeG2, YDTime1G, YDTime2G, NoTime,
&      GoodDrip
integer i, j, k, Dist, StackIt, SegND1, SegW1, SegW2, SegND2
double precision ranval, ranmz, gran
double precision CermTime, SampleValue, InvNor
double precision Parm(4), InvNorm, S, PatLnA, PatK, Rate

```

```

c
c Determine whether this patch is dripped on.
c

```

```

ranval = ranmz()
. if(PatchType .eq. TOPTYPE) then
  if(ranval .lt. TopDripFrc) then
    DrippedOn = .true.
  else
    DrippedOn = .false.
  end if
else if(PatchType .eq. SIDETYPE) then
  DrippedOn = .false.
else if(PatchType .eq. BOTTYPE) then
  if(ranval .lt. BotDripFrc) then
    DrippedOn = .true.
  else
    DrippedOn = .false.
  end if
end if

```

```

c
c Grab the appropriate time grid, place into generic arrays.
c

```

```

if(.not. DrippedOn) then

```

```

NowVal = NDNowVal
do i = 1, NowVal
  ntim(i) = NDntim(i)
  ntem(i) = NDntem(i)
  nr(i)    = NDnr(i)
end do
NDTimeG1 = NDTime
NDTimeG2 = .false.
YDTime1G = .false.
YDTime2G = .false.
Ndx1     = NowVal + 1
Ndx2     = NowVal + 1
Ndx3     = NowVal + 1
else
  NowVal = YDNowVal
  do i = 1, NowVal
    ntim(i) = YDntim(i)
    ntem(i) = YDntem(i)
    nr(i)    = YDnr(i)
  end do
  NDTimeG1 = YDTimeND1
  NDTimeG2 = YDTimeND2
  YDTime1G = YDTime1
  YDTime2G = YDTime2
  Ndx1     = NdxDrip
  Ndx2     = NdxChn
  Ndx3     = NdxEnd
end if

```

```

c
c Find time for this patch until ceramic coating fails.
c
  ranval = ranmz()
  CermTime = SampleValue(ranval, CermType, CermPar, CDFStack)
c
c If corrosion delayed due to ceramic coating, eliminate times prior
c to then from the time grid.
c
  if(CermTime .gt. 0.D0) then
    CALL DelayTime(CermTime, NowVal, ntim, ntem, nr, Ndx1, Ndx2,
      &              Ndx3, NoTime)
    if(NoTime) RETURN
    if(Ndx1 .le. 1) NDTimeG1 = .false.
    if(Ndx2 .le. 1) YDTime1G = .false.
    if(Ndx3 .le. 1) YDTime2G = .false.
  end if
c
c Slide the time scale so corrosion starts at time 0 and calculate the
c average RH and temperature during the time steps. 'Start' saves
c when corrosion actually began for this patch. Note that if CermTime

```

c was positive, routine DelayTime set ntim(1) = CermTime and adjusted
c the time indexes accordingly.

c
c Start = ntim(1)
c do i = NowVal, 2, -1
c dt(i) = ntim(i) - ntim(i - 1)
c ntim(i) = ntim(i) - Start
c ntem(i) = (ntem(i) + ntem(i - 1)) / 2.D0
c nr(i) = (nr(i) + nr(i - 1)) / 2.D0
c end do
c ntim(1) = 0.D0

c
c Since the general corrosion rate for the CAM is always required,
c sample 8 independent numbers from a standard normal distribution.
c These numbers will be used to generate the values for the general
c corrosion model parameters for the patch.

c
c do i = 1, 8
c Coef2(i) = gran()
c end do

c
c Find parameters for general humid air corrosion on the current patch.
c Linear transformations of the normal variables to get the values of
c the corrosion parameters for this patch based on the package average
c and the covariance structure provided in SetModelParameters. Include
c StanDev to represent the portion of the variability from the epsilon
c term in the corrosion model.

c
c Coef2(4) = Coef1(4) + V5(4, 5) * Coef2(1) + V5(4, 6) * Coef2(2) +
c & V5(4, 7) * Coef2(3) + V5(4, 8) * Coef2(4)
c Coef2(3) = Coef1(3) + V5(3, 5) * Coef2(1) + V5(3, 6) * Coef2(2) +
c & V5(3, 7) * Coef2(3)
c Coef2(2) = Coef1(2) + V5(2, 5) * Coef2(1) + V5(2, 6) * Coef2(2)
c Coef2(1) = Coef1(1) + V5(1, 5) * Coef2(1) + StanDev(1, 2) * gran()

c
c Find parameters for general aqueous corrosion on the current patch.
c Aqueous corrosion is treated the same way that humid air was above.

c
c Coef2(8) = Coef1(8) + V5(8, 5) * Coef2(5) + V5(8, 6) * Coef2(6) +
c & V5(8, 7) * Coef2(7) + V5(8, 8) * Coef2(8)
c Coef2(7) = Coef1(7) + V5(7, 5) * Coef2(5) + V5(7, 6) * Coef2(6) +
c & V5(7, 7) * Coef2(7)
c Coef2(6) = Coef1(6) + V5(6, 5) * Coef2(5) + V5(6, 6) * Coef2(6)
c Coef2(5) = Coef1(5) + V5(5, 5) * Coef2(5) + StanDev(2, 2) * gran()

c
c Set remaining properties for patch for each time segment. First
c determine segment indexing and the number of segments represented.

c
c SegND1 = 0


```

SegW1 = 0
SegW2 = 0
SegND2 = 0
i = 0
if(NDTimeG1) then
  i = i + 1
  SegND1 = i
  GPDepth(i) = 0.00
end if
if(YDTime1G) then
  i = i + 1
  SegW1 = i
  GPDepth(i) = 0.00
end if
if(YDTime2G) then
  i = i + 1
  SegW2 = i
  GPDepth(i) = 0.00
end if
if(NDTimeG2) then
  i = i + 1
  SegND2 = i
  GPDepth(i) = 0.00
end if
NumSeg = i

```

c If NDTIMEG1 or NDTIMEG2 are true then either the patch is not dripped
c on and there is time available for corrosion or the patch is dripped
c on and there is time for corrosion either prior to onset of dripping
c or after dripping ceases.

```

c
c SpallDep1 = 0.00
c if(NDTimeG1 .or. NDTIMEG2) then
c   if(SegND1 .gt. 0) then
c     SegStart(SegND1) = 1
c     SegEnd(SegND1) = min(NowVal, Ndx1)
c     CAMType(SegND1) = MSPEC(1)
c     GenRateM(SegND1, 1) = NDFEAT(5)
c     GenRateM(SegND1, 2) = NDFEAT(6)
c     MxCRMRate(SegND1) = CRMMAX(1)
c   end if
c   if(SegND2 .gt. 0) then
c     SegStart(SegND2) = Ndx3
c     SegEnd(SegND2) = NowVal
c     CAMType(SegND2) = MSPEC(1)
c     GenRateM(SegND2, 1) = NDFEAT(5)
c     GenRateM(SegND2, 2) = NDFEAT(6)
c     MxCRMRate(SegND2) = CRMMAX(1)
c   end if

```

```

c
c CAM not dripped on => WapDeg uses pit multiples for localization
c factors. Resample the largest pit multiple to be used for this patch.
c

```

```

      Dist          = DTypeStack(1)
      Parm(1)       = DParmStack(1, 1)
      Parm(2)       = DParmStack(1, 2)
      Parm(3)       = DParmStack(1, 3)
      Parm(4)       = DParmStack(1, 4)
      ranval        = 1.D0 - ranmz() / dble(NPitCAM)
      NDCritDep(NPitCAM) = SampleValue(ranval, Dist, Parm, CDFStack)
      if(SegND1 .gt. 0) then      !Note reverse order storage here
        MaxPM(SegND1) = NDCritDep(NPitCAM)
        do i = 1, NPitCAM
          CAMPit(i, SegND1, 1) = NDCritDep(NPitCAM - i + 1)
        end do
      end if
      if(SegND2 .gt. 0) then
        MaxPM(SegND2) = NDCritDep(NPitCAM)
        do i = 1, NPitCAM
          CAMPit(i, SegND2, 1) = NDCritDep(NPitCAM - i + 1)
        end do
      end if

```

```

c
c CRM not dripped on => use sampled general rate only (with its share
c of the variance accounted for) or can use TSPA95 exponential model.
c

```

```

      if(MSpec(2) .eq. CRMGONLY) then

```

```

c
c The NO-DRIP general corrosion rate parameters for the patch are found
c using the covariance, variance error, and the share of variance for
c the patches, and the package means.
c This treatment currently assumes the distributions for LnA and K are
c normal, and stored in Parm(1) and Parm(2) will be their means. Their
c standard deviations have been folded into the vg array. DL9999
c

```

```

      Parm(1) = NDPakLnA
      Parm(2) = NDPakK
      j       = 2
      k       = 1
      CALL GetCRMRateParms(vg, PatLnA, PatK, Parm, j, k)
      if(SegND1 .gt. 0) then
        CRMType(SegND1) = MSpec(2)
        CRMGen(SegND1, 1) = PatLnA
        CRMGen(SegND1, 2) = PatK
        PitInit(SegND1) = 0.D0
      end if
      if(SegND2 .gt. 0) then
        CRMType(SegND2) = MSpec(2)

```

```

        CRMGen(SegND2, 1) = PatLnA
        CRMGen(SegND2, 2) = PatK
        PitInit(SegND2) = 0.D0
    end if
c
    else if(MSpec(2) .eq. CRMGRONLY) then
c
c The NO-DRIP general corrosion rate for package is to be interpolated
c from sampled values at different temperatures. Sample a untransformed
c value for each given temperature using the variance share.
c
        ranval = gran() * sqrt(1.D0 - varshare)
        do i = 1, NumCRMD(1)
            Dist = DTypeStack(EndStack(1) + i)
            Parm(1) = DParmStack(EndStack(1) + i, 1)
            Parm(2) = DParmStack(EndStack(1) + i, 2)
            Parm(3) = DParmStack(EndStack(1) + i, 3)
            Parm(4) = DParmStack(EndStack(1) + i, 4)
            CALL GetCRMRates(ranval, NDPakR(i), Dist, Parm, CDFStack,
                & Rate)
            if(SegND1 .gt. 0) then
                CRMGenR(SegND1, i) = Rate
                CRMTemp(SegND1, i) = NDCRMT(i)
            end if
            if(SegND2 .gt. 0) then
                CRMGenR(SegND2, i) = Rate
                CRMTemp(SegND2, i) = NDCRMT(i)
            end if
        end do
        if(SegND1 .gt. 0) then
            CRMType(SegND1) = MSpec(2)
            PitInit(SegND1) = 0.D0
            CRMNoD(SegND1) = NumCRMD(1)
        end if
        if(SegND2 .gt. 0) then
            CRMType(SegND2) = MSpec(2)
            PitInit(SegND2) = 0.D0
            CRMNoD(SegND2) = NumCRMD(1)
        end if

    else if(MSpec(2) .eq. CRMEXP) then
        CRMType(1) = MSpec(2)
c
c Find the patch mean C0 and resample extremes of C0 distribution.
c
        if(DTypeStack(EndStack(1) + 1) .eq. DSTNOR) then
            C0PatM = C9 + V9 * gran()
            ranval = 1.D0 - ranmz() / dble(NPitCRM)
            InvNorm = InvNor(ranval)

```

```

      COPak(NPitCRM) = exp(InvNorm * V9)
      ranval        = ranmz() / dble(NPitCRM)
      InvNorm       = InvNor(ranval)
      COPak(1)      = exp(InvNorm * V9)
    else if(DTypeStack(EndStack(1) + 1) .eq. DSTFXD) then
      COPatM = C9
    end if
    do i = 1, NPitCRM
      CRMPit(i, 1, 1) = COPak(i)
    end do
  end if

c
c Determine the galvanic protection depth and set the spalling depth.
c
  CALL GetGPDepth(NDFeat(2), NDFeat(3), S)
  if(SegND1 .gt. 0) then
    GPDepth(SegND1) = S
  end if
  if(SegND2 .gt. 0) then
    GPDepth(SegND2) = S
  end if
  SpallDep1 = NDFeat(4)
end if

c
c Set properties for dripped-on patch INITIAL WATER condition, provided
c there is time in the segment.
c
  SpallDep2 = 0.D0
  if(DrippedOn .and. YDTimelG) then
    SegStart(SegW1) = max(1, Ndx1)
    SegEnd(SegW1)   = min(NowVal, Ndx2)
  end if

c
c Use the model and properties for INITIAL WATER condition for this
c segment. ModSpec will point to the appropriate model and StackIt will
c point to the distribution stack.
c
  if(NeutD1) then
    StackIt      = 4
    CAMType(SegW1) = MSpec(3)
    CRMType(SegW1) = MSpec(4)
    GenRateM(SegW1, 1) = GDFeat(5)
    GenRateM(SegW1, 2) = GDFeat(6)
    MxCRMRate(SegW1) = CRMMax(2)
    GoodDrip      = .true.
  else
    if(HaveNeut) then
      StackIt      = 7
      CAMType(SegW1) = MSpec(5)
      CRMType(SegW1) = MSpec(6)
    end if
  end if

```

```

      else
        StackIt      = 4
        CAMType(SegW1) = MSpec(3)
        CRMType(SegW1) = MSpec(4)
      end if
      GenRateM(SegW1, 1) = BDFeat(5)
      GenRateM(SegW1, 2) = BDFeat(6)
      MxCRMRate(SegW1) = CRMMax(3)
      GoodDrip = .false.
    end if
  c
  c If using pit growth law, substitute B and n into appropriate segment.
  c If using pit multiples, resample the largest before substituting.
  c
    if(CAMType(SegW1) .eq. CAMGPM) then
      Dist = DTypeStack(EndStack(StackIt - 1) + 1)
      Parm(1) = DParmStack(EndStack(StackIt - 1) + 1, 1)
      Parm(2) = DParmStack(EndStack(StackIt - 1) + 1, 2)
      Parm(3) = DParmStack(EndStack(StackIt - 1) + 1, 3)
      Parm(4) = DParmStack(EndStack(StackIt - 1) + 1, 4)
      ranval = 1.D0 - ranmz() / dble(NPitCAM)
      S = SampleValue(ranval, Dist, Parm, CDFStack)
      MaxPM(SegW1) = S
      if(GoodDrip) then
        GDCritDep(NPitCAM) = S      !Note reverse order storage here
        do i = 1, NPitCAM
          CAMPit(i, SegW1, 1) = GDCritDep(NPitCAM - i + 1)
        end do
      else
        BDCritDep(NPitCAM) = S
        do i = 1, NPitCAM
          CAMPit(i, SegW1, 1) = BDCritDep(NPitCAM - i + 1)
        end do
      end if
    else
      if(.not. GoodDrip) then
        do i = 1, NPitCAM
          CAMPit(i, SegW1, 1) = BDCAMB(i)
          CAMPit(i, SegW1, 2) = BDCAMn(i)
        end do
      end if
    end if
  c
  c The NEUTRAL-DRIP general corrosion rate parameters for the patch are
  c found using the covariance, variance error, and the share of variance
  c for the patches, and the package means.
  c This treatment currently assumes the distributions for LnA and K are
  c normal, and stored in Parm(1) and Parm(2) will be their means. Their
  c standard deviations have been folded into the vg array. DL9999

```

c

```

if(GoodDrip) then
  PitInit(SegW1) = GDPInit

  if(CRMType(SegW1) .eq. CRMGONLY .or.
&   CRMType(SegW1) .eq. CRMGPGL .or.
&   CRMType(SegW1) .eq. CRMGLOG .or.
&   CRMType(SegW1) .eq. CRMGLNL) then
    Parm(1) = GDPakLnA
    Parm(2) = GDPakK
    j       = 2
    k       = 2
    CALL GetCRMRateParms(vg, PatLnA, PatK, Parm, j, k)
    CRMGen(SegW1, 1) = PatLnA
    CRMGen(SegW1, 2) = PatK

  else if(CRMType(SegW1) .eq. CRMGRONLY .or.
&   CRMType(SegW1) .eq. CRMGRPGL .or.
&   CRMType(SegW1) .eq. CRMGRLOG .or.
&   CRMType(SegW1) .eq. CRMGRNL) then
    ranval = gran() * sqrt(1.00 - varshare)
    do i = 1, NumCRMD(2)
      Dist = DTypeStack(EndStack(StackIt) + i)
      Parm(1) = DParmStack(EndStack(StackIt) + i, 1)
      Parm(2) = DParmStack(EndStack(StackIt) + i, 2)
      Parm(3) = DParmStack(EndStack(StackIt) + i, 3)
      Parm(4) = DParmStack(EndStack(StackIt) + i, 4)
      CALL GetCRMRates(ranval, GDPakR(i), Dist, Parm, CDFStack,
&                      Rate)
      CRMGenR(SegW1, i) = Rate
      CRMTemp(SegW1, i) = GDCRMT(i)
    end do
    CRMNoD(SegW1) = NumCRMD(2)
  end if

else
  PitInit(SegW1) = BDPInit

  if(CRMType(SegW1) .eq. CRMGONLY .or.
&   CRMType(SegW1) .eq. CRMGPGL .or.
&   CRMType(SegW1) .eq. CRMGLOG .or.
&   CRMType(SegW1) .eq. CRMGLNL) then
    Parm(1) = BDPakLnA
    Parm(2) = BDPakK
    j       = 2
    k       = 3
    CALL GetCRMRateParms(vg, PatLnA, PatK, Parm, j, k)
    CRMGen(SegW1, 1) = PatLnA
    CRMGen(SegW1, 2) = PatK
  
```

```

      else if(CRMType(SegW1) .eq. CRMGRONLY .or.
&          CRMType(SegW1) .eq. CRMGRPGL .or.
&          CRMType(SegW1) .eq. CRMGRLOG .or.
&          CRMType(SegW1) .eq. CRMGRNLN) then
        ranval = gran() * sqrt(1.00 - varshare)
        do i = 1, NumCRMD(3)
          Dist = DTypeStack(EndStack(StackIt) + i)
          Parm(1) = DParmStack(EndStack(StackIt) + i, 1)
          Parm(2) = DParmStack(EndStack(StackIt) + i, 2)
          Parm(3) = DParmStack(EndStack(StackIt) + i, 3)
          Parm(4) = DParmStack(EndStack(StackIt) + i, 4)
          CALL GetCRMRates(ranval, BDPakR(i), Dist, Parm, CDFStack,
&                          Rate)
          CRMGenR(SegW1, i) = Rate
          CRMTemp(SegW1, i) = BDCRMT(i)
        end do
        CRMNoD(SegW1) = NumCRMD(3)
      end if
    end if

```

```

C
c Substitute appropriate pit growth parameters.
C

```

```

      if(CRMType(SegW1) .eq. CRMGPGL .or.
&      CRMType(SegW1) .eq. CRMGRPGL) then
        if(GoodDrip) then
          do i = 1, NPitCRM
            CRMPit(i, SegW1, 1) = GDCRM1(i)
            CRMPit(i, SegW1, 2) = GDCRM2(i)
          end do
        else
          do i = 1, NPitCRM
            CRMPit(i, SegW1, 1) = BDCRM1(i)
            CRMPit(i, SegW1, 2) = BDCRM2(i)
          end do
        end if
      else if(CRMType(SegW1) .eq. CRMGLOG .or.
&      CRMType(SegW1) .eq. CRMGRLOG) then
        if(GoodDrip) then
          do i = 1, NPitCRM
            CRMPit(i, SegW1, 1) = GDCRM1(i)
            CRMPit(i, SegW1, 2) = GDCRM2(i)
            CRMPit(i, SegW1, 3) = GDCRM3(i)
          end do
        else
          do i = 1, NPitCRM
            CRMPit(i, SegW1, 1) = BDCRM1(i)
            CRMPit(i, SegW1, 2) = BDCRM2(i)
            CRMPit(i, SegW1, 3) = BDCRM3(i)
          end do
        end if
      end if

```

```

        end do
      end if
      else if (CRMTType(SegW1) .eq. CRMGLNL .or.
&      CRMTType(SegW1) .eq. CRMGRNLNL) then
        if (GoodDrip) then
          do i = 1, NPitCRM
            CRMPit(i, SegW1, 1) = GDCRM1(i)
            CRMPit(i, SegW1, 2) = GDCRM2(i)
            CRMPit(i, SegW1, 3) = GDCRM3(i)
          end do
        else
          do i = 1, NPitCRM
            CRMPit(i, SegW1, 1) = BDCRM1(i)
            CRMPit(i, SegW1, 2) = BDCRM2(i)
            CRMPit(i, SegW1, 3) = BDCRM3(i)
          end do
        end if
      end if
    end if
  c
  c Determine the galvanic protection depth and set the spalling depth.
  c
    if (GoodDrip) then
      CALL GetGPDepth(GDFeat(2), GDFeat(3), GPDepth(SegW1))
      SpallDep2 = GDFeat(4)
    else
      CALL GetGPDepth(BDFeat(2), BDFeat(3), GPDepth(SegW1))
      SpallDep2 = BDFeat(4)
    end if
  end if
  c
  c Set properties for dripped-on patch NEW WATER condition, provided
  c there is time in the segment.
  c
  c SpallDep3 = 0.D0
  c
  if (DrippedOn .and. YDTime2) then
    SegStart(SegW2) = max(1, Ndx2)
    SegEnd(SegW2) = min(NowVal, Ndx3)
  c
  c Use the model and properties for NEW WATER condition for this
  c segment. ModSpec will point to the appropriate model and StackIt will
  c point to the distribution stack.
  c
    if (NeutD2) then
      StackIt = 4
      CAMType(SegW2) = MSpec(3)
      CRMTType(SegW2) = MSpec(4)
      GenRateM(SegW2, 1) = GDFeat(5)
      GenRateM(SegW2, 2) = GDFeat(6)
      MxCRMRate(SegW2) = CRMMax(2)
    end if
  end if

```



```

        GoodDrip          = .true.
    else
        if(HaveNeut) then
            StackIt        = 7
            CAMType(SegW2) = MSpec(5)
            CRMType(SegW2) = MSpec(6)
        else
            StackIt        = 4
            CAMType(SegW2) = MSpec(3)
            CRMType(SegW2) = MSpec(4)
        end if
        GenRateM(SegW2, 1) = BDFeat(5)
        GenRateM(SegW2, 2) = BDFeat(6)
        MxCRMRate(SegW2)   = CRMMax(3)
        GoodDrip = .false.
    end if

c
c If using pit growth law, substitute into segment arrays.  If using
c pit multiples, resample the largest before substituting.
c
    if(CAMType(SegW2) .eq. CAMGPM) then
        Dist = DTypeStack(EndStack(StackIt - 1) + 1)
        Parm(1) = DParmStack(EndStack(StackIt - 1) + 1, 1)
        Parm(2) = DParmStack(EndStack(StackIt - 1) + 1, 2)
        Parm(3) = DParmStack(EndStack(StackIt - 1) + 1, 3)
        Parm(4) = DParmStack(EndStack(StackIt - 1) + 1, 4)
        ranval = 1.00 - ranmz() / dble(NPitCAM)
        S = SampleValue(ranval, Dist, Parm, CDFStack)
        MaxPM(SegW2) = S
        if(GoodDrip) then
            GDCritDep(NPitCAM) = S      !Note reverse order storage here
            do i = 1, NPitCAM
                CAMPit(i, SegW2, 1) = GDCritDep(NPitCAM - i + 1)
            end do
        else
            BDCritDep(NPitCAM) = S
            do i = 1, NPitCAM
                CAMPit(i, SegW2, 1) = BDCritDep(NPitCAM - i + 1)
            end do
        end if
    else
        if(.not. GoodDrip) then
            do i = 1, NPitCAM
                CAMPit(i, SegW2, 1) = BDCAMB(i)
                CAMPit(i, SegW2, 2) = BDCAMn(i)
            end do
        end if
    end if
c

```

c The NONNEUTRAL-DRIP general corrosion rate parameters for the patch
 c are found using the covariance, variance error, and the share of
 c variance for the patches, and the package means.
 c This treatment currently assumes the distributions for LnA and K are
 c normal, and stored in Parm(1) and Parm(2) will be their means. Their
 c standard deviations have been folded into the vg array. DL9999
 c

```

      if(GoodDrip) then
        PitInit(SegW2) = GDPInit

        if(CRMType(SegW2) .eq. CRMGONLY .or.
          & CRMType(SegW2) .eq. CRMGPGL .or.
          & CRMType(SegW2) .eq. CRMGLOG .or.
          & CRMType(SegW2) .eq. CRMGLNL) then
          Parm(1) = GDPakLnA
          Parm(2) = GDPakK
          j = 2
          k = 2
          CALL GetCRMRateParms(vg, PatLnA, PatK, Parm, j, k)
          CRMGen(SegW2, 1) = PatLnA
          CRMGen(SegW2, 2) = PatK

        else if(CRMType(SegW2) .eq. CRMGRONLY .or.
          & CRMType(SegW2) .eq. CRMGRPGL .or.
          & CRMType(SegW2) .eq. CRMGRLOG .or.
          & CRMType(SegW2) .eq. CRMGRNL) then
          ranval = gran() * sqrt(1.D0 - varshare)
          do i = 1, NumCRMD(2)
            Dist = DTypeStack(EndStack(StackIt) + i)
            Parm(1) = DParmStack(EndStack(StackIt) + i, 1)
            Parm(2) = DParmStack(EndStack(StackIt) + i, 2)
            Parm(3) = DParmStack(EndStack(StackIt) + i, 3)
            Parm(4) = DParmStack(EndStack(StackIt) + i, 4)
            CALL GetCRMRates(ranval, GDPakR(i), Dist, Parm, CDFStack,
              & Rate)
            CRMGenR(SegW2, i) = Rate
            CRMTemp(SegW2, i) = GDCRMT(i)
          end do
          CRMNoD(SegW2) = NumCRMD(2)
        end if

      else
        PitInit(SegW2) = BDPInit

        if(CRMType(SegW2) .eq. CRMGONLY .or.
          & CRMType(SegW2) .eq. CRMGPGL .or.
          & CRMType(SegW2) .eq. CRMGLOG .or.
          & CRMType(SegW2) .eq. CRMGLNL) then
          Parm(1) = BDPakLnA

```

```

      Parm(2) = BDPakK
      j      = 2
      k      = 3
      CALL GetCRMRateParms(vg, PatLnA, PatK, Parm, j, k)
      CRMGen(SegW2, 1) = PatLnA
      CRMGen(SegW2, 2) = PatK

      else if (CRMType(SegW2) .eq. CRMGRONLY .or.
&           CRMType(SegW2) .eq. CRMGRPGL .or.
&           CRMType(SegW2) .eq. CRMGRLOG .or.
&           CRMType(SegW2) .eq. CRMGRNLN) then
        ranval = gran() * sqrt(1.D0 - varshare)
        do i = 1, NumCRMD(3)
          Dist = DTypeStack(EndStack(StackIt) + i)
          Parm(1) = DParmStack(EndStack(StackIt) + i, 1)
          Parm(2) = DParmStack(EndStack(StackIt) + i, 2)
          Parm(3) = DParmStack(EndStack(StackIt) + i, 3)
          Parm(4) = DParmStack(EndStack(StackIt) + i, 4)
          CALL GetCRMRate(ranval, BDPakR(i), Dist, Parm, CDFStack,
&                      Rate)
          CRMGenR(SegW2, i) = Rate
          CRMTemp(SegW2, i) = BDCRMT(i)
        end do
        CRMNoD(SegW2) = NumCRMD(3)
      end if
    end if

c
c   Substitute appropriate pit growth parameters.
c
      if (CRMType(SegW2) .eq. CRMGPGL .or.
&       CRMType(SegW2) .eq. CRMGRPGL) then
        if (GoodDrip) then
          do i = 1, NPitCRM
            CRMPit(i, SegW2, 1) = GDCRM1(i)
            CRMPit(i, SegW2, 2) = GDCRM2(i)
          end do
        else
          do i = 1, NPitCRM
            CRMPit(i, SegW2, 1) = BDCRM1(i)
            CRMPit(i, SegW2, 2) = BDCRM2(i)
          end do
        end if
      else if (CRMType(SegW2) .eq. CRMGLLOG .or.
&           CRMType(SegW2) .eq. CRMGRLOG) then
        if (GoodDrip) then
          do i = 1, NPitCRM
            CRMPit(i, SegW2, 1) = GDCRM1(i)
            CRMPit(i, SegW2, 2) = GDCRM2(i)
            CRMPit(i, SegW2, 3) = GDCRM3(i)

```

```

        end do
      else
        do i = 1, NPitCRM
          CRMPit(i, SegW2, 1) = BDCRM1(i)
          CRMPit(i, SegW2, 2) = BDCRM2(i)
          CRMPit(i, SegW2, 3) = BDCRM3(i)
        end do
      end if
      else if (CRMType(SegW2) .eq. CRMGLNL .or.
        & CRMType(SegW2) .eq. CRMGRNL) then
        if (GoodDrip) then
          do i = 1, NPitCRM
            CRMPit(i, SegW2, 1) = GDCRM1(i)
            CRMPit(i, SegW2, 2) = GDCRM2(i)
            CRMPit(i, SegW2, 3) = GDCRM3(i)
          end do
        else
          do i = 1, NPitCRM
            CRMPit(i, SegW2, 1) = BDCRM1(i)
            CRMPit(i, SegW2, 2) = BDCRM2(i)
            CRMPit(i, SegW2, 3) = BDCRM3(i)
          end do
        end if
      end if
    end if
  end if
  c Determine the galvanic protection depth and set the spalling depth.
  c
  if (GoodDrip) then
    CALL GetGPDepth(GDFeat(2), GDFeat(3), GPDepth(SegW2))
    SpallDep3 = GDFeat(4)
  else
    CALL GetGPDepth(BDFeat(2), BDFeat(3), GPDepth(SegW2))
    SpallDep3 = BDFeat(4)
  end if
end if
c
c Only need to simulate patch if there is time available.
c
if (.not. DrippedOn .and. NDTime .or. DrippedOn .and. YDTime) then
  DoPatch = .true.
else
  DoPatch = .false.
end if
RETURN
END      !SUBROUTINE SetupPatch
c
c *****
c
SUBROUTINE CorrosionNoSP(c10, Penet1, Penet2, NPitCAM, NPitCRM,

```

```

&      binf, BinFact, BinTerm, NeutD1, NeutD2,
&      MSpec, Pit1st, Pat1st, BinStart, Bin,
&      PatStart, PatBin, CRMThk, NDTpVal,
&      NDHASTart, NDAquaX, GDTpVal, GDHASTart,
&      GDAquaX, BDTpVal, BDHASTart, BDAquaX,
&      PwrIB, PatchType, GPDepth, Coef2, Start,
&      COPatM, dt, NumSeg, SegStart, SegEnd,
&      Ndx1, Ndx2, Ndx3, CAMType, CRMType,
&      PitInit, MaxPM, CRMGen, GenRateM,
&      MxCRMRate, CRMGenR, CRMTemp, CRMNoD,
&      NowVal, ntim, ntem, nr, CAMPit, CRMPit)
c
c Calculate
c 1. General corrosion depth history for this patch - no spalling
c 2. Outer barrier pit penetration times (by calling PitOuter)
c 3. Inner barrier pit penetration times (by calling PitInner)
c Input : c10, Penet1, Penet2, NPitCAM, NPitCRM, binf, BinFact,
c      BinTerm, NeutD1, NeutD2, MSpec, Pit1st, Pat1st,
c      BinStart, Bin, PatStart, PatBin, CRMThk, NDTpVal, NDHASTart,
c      NDAquaX, GDTpVal, GDHASTart, GDAquaX, BDTpVal, BDHASTart,
c      BDAquaX, PwrIB, PatchType, GPDepth, Coef2, Start, COPatM, dt,
c      NumSeg, SegStart, SegEnd, Ndx1, Ndx2, Ndx3, CAMType, CRMType,
c      PitInit, MaxPM, CRMGen, GenRateM, MxCRMRate, CRMGenR,
c      CRMTemp, CRMNoD, NowVal, ntim, ntem, nr, CAMPit, CRMPit
c Output: Pit1st, Pat1st, BinStart, Bin, PatStart, PatBin, CRMThk
c Local : CAMThru, GotGPT(3), PwrEq1, LastSegSame, DrippedOn, Count,
c      Pit, Cnt1st, GPNdx, K4PIT, Slide, PatNdx, SegNo, i, LostSeg,
c      CRMMod, PatchBin, Temp, RH, ratecam, pwr, PDTrans, GPTIME,
c      ActGPT, CAM1st, PatGone, EndDep, TmInit, RMOuter, RMInner,
c      TpVal, HAINit, AquaTrans, PatchD, ctim, c10Tdt, c10T, rcam,
c      invpwr, SumC0, DTSave, PitTmp, NumPit, FirstCAMSeg, ReducePit
c
c Arguments, and dimension sizes from the include file.
c
c      IMPLICIT NONE
c      INCLUDE 'parm306.inc'
c      logical NeutD1, NeutD2
c      integer NPitCAM, NPitCRM, NowVal, PatchType, NumSeg, Ndx1, Ndx2,
c      &      Ndx3
c      integer Bin(MAXBIN, 3), PatBin(MAXBIN, 5)
c      integer MSpec(*), SegStart(*), SegEnd(*)
c      integer CAMType(*), CRMType(*), CRMNoD(*)
c      double precision c10, Penet1, Penet2, binf, BinFact, BinTerm
c      double precision BinStart, PatStart
c      double precision Pit1st(*), Pat1st(*)
c      double precision NDTpVal, NDHASTart, NDAquaX
c      double precision GDTpVal, GDHASTart, GDAquaX
c      double precision BDTpVal, BDHASTart, BDAquaX
c      double precision PwrIB, Start, COPatM

```

```

double precision GPDepth(*), Coef2(*), dt(*), CRMGen(MAXSEG, 2),
& CRMGenR(MAXSEG, MAXCRMDISTS),
& CRMTemp(MAXSEG, MAXCRMDISTS)
double precision PitInit(*), MaxPM(*), GenRateM(MAXSEG, 2)
double precision ntim(*), ntem(*), nr(*), CRMThk(*), MxCRMRate(*)
double precision CAMPit(MAXPITCAM, MAXSEG, 2),
& CRMPit(MAXPITCRM, MAXSEG, 3)

```

c

c Local variables

c

```

logical CAMThru, GotGPT(MAXSEG), PwrEq1, LastSegSame, AreEqual,
& DrippedOn, ReducePit
integer Count, Pit, Cntlst, GPNdx(MAXSEG), K4PIT(MAXPIT), Slide,
& PatNdx, SegNo, i, LostSeg, CRMMod, PatchBin, NumPit,
& FirstCAMSeg
double precision Temp, RH, ratecam, pwr, CAM1st, PatGone, EndDep,
& TmInit, PitTmp
double precision PDTrans(MAXSEG), GPTtime(MAXSEG), ActGPT(MAXSEG)
double precision RMOuter, RMiner, TpVal, HAINit, AquaTrans
double precision PatchD(MAXDAT1), ctim(MAXDAT1), c1OTdt(MAXDAT1)
double precision c1OT(MAXDAT1), rcam(MAXDAT1), invpwr(MAXDAT1)
double precision SumCO(MAXDAT1), DTSave(MAXPIT)

```

c

```

CAM1st = ntim(NowVal)
Cntlst = NowVal + 1
PatNdx = NowVal + 1
PatGone = ntim(NowVal) + 1.D0
DrippedOn = .false.
CALL InitDPArray(PatchD, NowVal)
do SegNo = 1, NumSeg
  GotGPT(SegNo) = .false.
  GPNdx(SegNo) = 0
  GPTtime(SegNo) = 0.D0
  ActGPT(SegNo) = 0.D0
  PDTrans(SegNo) = Penet1
end do

```

c

c Determine the general corrosion depth during each time step.

c

```

do SegNo = 1, NumSeg
  RMOuter = GenRateM(SegNo, 1)
  RMiner = GenRateM(SegNo, 2)
  do Count = SegStart(SegNo) + 1, SegEnd(SegNo)
    Temp = ntem(Count)
    RH = nr(Count)
    if(Count .le. Ndx1 .or. Count .gt. Ndx3) then !No drip phases
      TpVal = NDTpVal
      HAINit = NDHASTart
      AquaTrans = NDAquaX
    end if
  end do
end do

```

```

      if(CRMType(SegNo) .eq. CRMEXP) then
        cl0T(Count) = EXP(C0PatM + C10 / Temp) * RMiner
        cl0Tdt(Count) = cl0T(Count) * dt(Count)
      end if

c
      else if(Count .le. Ndx2) then                                !Initial water
        DrippedOn = .true.
        if(NeutD1) then
          TpVal = GDTpVal
          HAINit = GDHASTart
          AquaTrans = GDAquaX
        else
          TpVal = BDTpVal
          HAINit = BDHASTart
          AquaTrans = BDAquaX
        end if

c
      else                                                        !New water
        DrippedOn = .true.
        if(NeutD2) then
          TpVal = GDTpVal
          HAINit = GDHASTart
          AquaTrans = GDAquaX
        else
          TpVal = BDTpVal
          HAINit = BDHASTart
          AquaTrans = BDAquaX
        end if
      end if

c
c   Relative humidity and temperature in range for CAM corrosion?
c
c   if(RH .gt. HAINit .and. Temp .lt. TpVal) then
c
c   Depending corrosion mode, set rate and power term for time exponent.
c
      IF (RH .lt. AquaTrans) then
        ratecam = EXP(Coef2(1) + Coef2(2)/Temp + Coef2(3)/RH)
        pwr = Coef2(4)
      else
        ratecam = EXP(Coef2(5) + Coef2(7)/Temp + Coef2(8)*Temp*Temp)
        pwr = Coef2(6)
      end if
      ratecam = ratecam * RMOuter

c
c   Save needed properties of the patch history. If galvanic protection
c   in force, find time step (GPNdx) at which its depth reached and the
c   time remaining in that step (GPTime).
c

```

```

ctim(Count) = (PatchD(Count-1)/ratecam)**(1.0/pwr)+dt(Count)
PatchD(Count) = ratecam * ctim(Count)**pwr
EndDep      = PatchD(Count)
rcam(Count) = ratecam
invpwr(Count) = 1.00 / pwr
if(GPDepth(SegNo) .gt. 0.00) then
  if(PatchD(Count) .ge. GPDepth(SegNo) .and.
    &      .not. GotGPT(SegNo)) then
    GPIdx(SegNo) = Count
    GPTime(SegNo) = (GPDepth(SegNo) / rcam(Count)) **
    &      invpwr(Count)
    GPTime(SegNo) = ctim(Count) - GPTime(SegNo)
    ActGPT(SegNo) = ntim(Count) - GPTime(SegNo)
    GotGPT(SegNo) = .true.
  end if
end if
if(PatchD(Count) .ge. Penet1) EXIT
else
  PatchD(Count) = PatchD(Count - 1)
end if
end do

c
c Find corrosion depth at transition times, if patch not gone.
c
  if(EndDep .lt. Penet1) then
    PDTrans(SegNo) = EndDep
  else
    EXIT
  end if
end do

c
c If patch galvanically protected on inner barrier until outer depth
c reaches GPDepth, but outer depth never gets that far, then no pits
c will penetrate the outer (let alone the inner barrier). Only apply
c this test if there is only one time interval.
c
  if(NumSeg .eq. 1) then
    if(GPDepth(1) .gt. 0.00 .and. .not. GotGPT(1)) RETURN
  end if
  do SegNo = 1, NumSeg
    if(.not. GotGPT(SegNo)) GPDepth(SegNo) = 0.00
  end do
  do Pit = 1, NPitCAM
    K4Pit(Pit) = NowVal + 1
    DTSave(Pit) = 0.00
  end do

c
c If not doing TSPA95 simulation & pitting on CAM always characterized
c by multiples, then can do shortcut. That is, identify segment when

```


c breakthru will occur and only do worst pit.

c

```

      ReducePit   = .true.
      FirstCAMSeg = 1
      if(MSpec(2) .eq. CRMEXP) then
        ReducePit = .false.
      else
        do SegNo = 1, NumSeg
          if(CAMType(SegNo) .ne. CAMGPM) then
            ReducePit = .false.
            exit
          end if
        end do
      end if
      if(ReducePit) then
        do SegNo = 1, NumSeg
          if(MaxPM(SegNo) * PDTrans(SegNo) .lt. Penet1) then
            FirstCAMSeg = FirstCAMSeg + 1
          else
            exit
          end if
        end do
        if(FirstCAMSeg .gt. NumSeg) RETURN
      end if

```

c

c Do pitting corrosion on CAM

c

```

      do SegNo = FirstCAMSeg, NumSeg

```

c

```

        if(CAMType(SegNo) .eq. CAMGPM) then
          if(ReducePit) then
            NumPit = 1
          else
            NumPit = NPitCAM
          end if
          CALL OuterPitMult(NumPit, Penet1, GPDepth, Start, SegStart,
&                               SegEnd, NowVal, ntim, CAMPit, GPNdx, GPTIME,
&                               ActGPT, PatchD, SegNo, PDTrans, K4PIT,
&                               DTSave, CAM1st, Cntlst, rcam, ctim, invpwr,
&                               FirstCAMSeg)

```

c

```

        else if(CAMType(SegNo) .eq. CAMGPGL) then
          CALL OuterPitLaw(NPitCAM, Penet1, NowVal, Start, ntim,
&                               SegStart, SegEnd, CAMPit, GPNdx, GPTIME,
&                               ActGPT, PatchD, SegNo, PDTrans, K4PIT,
&                               DTSave, CAM1st, Cntlst)
        end if
        if(CAM1st .lt. ntim(NowVal)) then
          CAMThru = .true.

```

```

      EXIT
    else
      CAMThru = .false.
    end if
  end do
  if(.not. CAMThru) RETURN

```

```

c
c At least one pit penetrated the CAM (at time CAM1st). If we are NOT
c accessing the TSPA95 model, then corrosion of the CRM starts at this
c time and we slide the time scale so corrosion starts at time 0.
c Else, each pit creates a new pit on the CRM and, since only one time
c segment is allowed in this case, the values for K4Pit and DTSave are
c appropriate for initialization for the inner barrier.
c

```

```

      LostSeg = 0
      if(MSpec(2) .ne. CRMEXP) then
        if(ntim(Cnt1st) - CAM1st .le. 0.D0) Cnt1st = Cnt1st + 1
        Slide = Cnt1st - 2
        do SegNo = 1, NumSeg
          SegEnd(SegNo) = SegEnd(SegNo) - Slide
          if(SegEnd(SegNo) .lt. 2) then
            LostSeg = LostSeg + 1
          else
            SegStart(SegNo) = max(SegStart(SegNo) - Slide, 1)
          end if
        end do
        if(LostSeg .eq. NumSeg) RETURN
        do i = Cnt1st, NowVal
          ntim(i - Slide) = ntim(i) - CAM1st
          ntem(i - Slide) = ntem(i)
          nr(i - Slide) = nr(i)
        end do
        ntim(1) = 0.D0
        NowVal = NowVal - Slide
        Ndx1 = Ndx1 - Slide
        Ndx2 = Ndx2 - Slide
        Ndx3 = Ndx3 - Slide
        do Pit = 1, NPitCRM
          K4Pit(Pit) = 0
          DTSave(Pit) = 0.D0
        end do
        do i = 2, NowVal
          dt(i) = ntim(i) - ntim(i - 1)
        end do
        CALL InitDPArray(PatchD, NowVal)
        CRMod = CRMType(LostSeg + 1)
        if(CRMod .eq. CRMGONLY .or. CRMod .eq. CRMGPGI .or.
&      CRMod .eq. CRMGLOG .or. CRMod .eq. CRMGLENL) then
          CALL InnerGenHist(Penet2, NumSeg, SegStart, SegEnd, CRMGen,

```

```

&          GenRateM, MxCRMRate, ntim, ntem, LostSeg,
&          PatchD, PDTrans, PatGone, PatNdx)
      else
        CALL InnerGenRate(Penet2, NumSeg, SegStart, SegEnd, GenRateM,
&          MxCRMRate, CRMNoD, CRMGenR, CRMTemp, ntim,
&          ntem, LostSeg, PatchD, PDTrans, PatGone,
&          PatNdx)
      end if
      TmInit = CAM1st + Start
      CALL GetCRMThick(Penet2, binf, CRMThk, NowVal, ntim, PatchD,
&          TmInit, PatGone)
c
c  See if temperature sufficiently high to initiate pitting in CRM, if
c  pitting to be considered there. If not, change to general corrosion.
c  The following assumes nonincreasing temperatures with time, but the
c  thresholds can be different for different water chemistries.
c
      PitTmp = ntem(2)
      do SegNo = LostSeg + 1, NumSeg
        if(CRMType(SegNo) .ne. CRMGONLY) then
          if(PitInit(SegNo) .le. PitTmp) then
            EXIT
          else
            CRMType(SegNo) = CRMGONLY
          end if
        end if
        if(SegNo .lt. NumSeg) then
          PitTmp = ntem(SegStart(SegNo + 1) + 1)
        end if
      end do
      end if
c
c  CAM1st now becomes start time for inner barrier corrosion. Convert it
c  to the global time scale and update earliest penetration.
c
      CAM1st = CAM1st + Start
      if(PatchType .eq. TOPTYPE) then
        Pit1st(1) = min(CAM1st, Pit1st(1))
      else if(PatchType .eq. SIDETYPE) then
        Pit1st(2) = min(CAM1st, Pit1st(2))
      else if(PatchType .eq. BOTTYPE) then
        Pit1st(3) = min(CAM1st, Pit1st(3))
      end if
c
c  PatGone is when the patch fails for the CRM. Update earliest time
c  arrays if failure before end of time.
c
      if(PatNdx .le. NowVal) then
        PatGone = PatGone + CAM1st

```

```
PatStart = MIN(PatStart, PatGone)
PatchBin = 1 + BinFact * log(PatGone) - BinTerm
BinStart = MIN(BinStart, PatGone)
if(PatchType .eq. TOPTYPE) then
  if(DrippedOn) then
    Pat1st(1) = min(PatGone, Pat1st(1))
    Pit1st(4) = min(PatGone, Pit1st(4))
    PatBin(PatchBin, 1) = PatBin(PatchBin, 1) + 1
  else
    Pat1st(3) = min(PatGone, Pat1st(3))
    PatBin(PatchBin, 3) = PatBin(PatchBin, 3) + 1
  end if
else if(PatchType .eq. SIDETYPE) then
  Pat1st(4) = min(PatGone, Pat1st(4))
  PatBin(PatchBin, 4) = PatBin(PatchBin, 4) + 1
else if(PatchType .eq. BOTTYPE) then
  if(DrippedOn) then
    Pat1st(2) = min(PatGone, Pat1st(2))
    Pit1st(5) = min(PatGone, Pit1st(5))
    PatBin(PatchBin, 2) = PatBin(PatchBin, 2) + 1
  else
    Pat1st(5) = min(PatGone, Pat1st(5))
    PatBin(PatchBin, 5) = PatBin(PatchBin, 5) + 1
  end if
end if
end if
c
c Simulate inner barrier corrosion.
c
  LastSegSame = .false.
  do SegNo = LostSeg + 1, NumSeg
    CRMMod = CRMType(SegNo)
c
    if((CRMMod .eq. CRMGONLY) .or. (CRMMod .eq. CRMGRONLY)) then
c
c General corrosion only.
c
      if(PatNdx .le. SegEnd(SegNo)) then
c
c Patch gets thru in this time segment. Have already binned the result
c in the IF-bloc above, so quit.
c
        RETURN
      else
c
c Patch doesn't penetrate during this segment. Set all total pit
c penetration depths to that of patch at end of segment. This
c assumes that any previous pits are stifled and will be over-run by
c general corrosion. There might be a problem here if it were possible
```

```
c  to have three segments characterized by pitting, no pitting, pitting.
c  But this is not allowed in current WapDeg.
c
      EndDep = PatchD(SegEnd(SegNo))
      do Pit = 1, NPitCRM
        DTSave(Pit) = EndDep
      end do
    end if
c
cKM
      else if((CRMMod .eq. CRMGPGL) .or. (CRMMod .eq. CRMGRPGL)) then
c
c  Use the pit growth law to find incremental pit depths (Scully).
c  Note that if two kinds of drips fall in two segments, WapDeg earlier
c  required that the pit law parameters were the same in both segments.
c  So, set the reference time for the exponential law, TmInit.
c
      if(LastSegSame) then
        TmInit = ntim(SegStart(SegNo - 1))
      else
        TmInit = ntim(SegStart(SegNo))
      end if
c
      CALL InnerPitLaw(Penet2, NPitCRM, BinFact, BinTerm, Pit1st,
&                    BinStart, Bin, PatchType, SegStart, SegEnd,
&                    NowVal, ntim, CRMPit, PatchD, SegNo, K4Pit,
&                    PDTrans, CAM1st, TmInit, DTSave)
c
c  If whole patch failed in this segment, decrement the pits bin array
c  in the bin where failure occurred, and return. Guard against input
c  that may have resulted in negative pit growth by also decrementing
c  from bins pits that broke through after patch failure.
c
      if(PatNdx .le. SegEnd(SegNo)) then
        do Pit = 1, NPitCRM
          i = K4Pit(Pit)
          if(i .le. PatchBin .and. i .gt. 0) then
            Bin(PatchBin, PatchType) = Bin(PatchBin, PatchType) - 1
          else if(i .gt. PatchBin) then
            Bin(i, PatchType) = Bin(i, PatchType) - 1
          end if
        end do
        RETURN
      end if
      LastSegSame = .true.
c
      else if((CRMMod .eq. CRMGLG) .or. (CRMMod .eq. CRMGRLOG)) then
c
c  Use log pit growth law to find incremental pit depths (Shoesmith).
```

c Note that if two kinds of drips fall in two segments, WapDeg earlier
 c required that the pit law parameters were the same in both segments.
 c So, set the reference time for the log law, TmInit.

```
c
      if (LastSegSame) then
        TmInit = ntim(SegStart(SegNo - 1))
      else
        TmInit = ntim(SegStart(SegNo))
      end if
      CALL InnerLogLaw(Penet2, NPitCRM, BinFact, BinTerm, Pit1st,
&                      BinStart, Bin, PatchType, SegStart, SegEnd,
&                      NowVal, ntim, ntem, CRMPit, PatchD, SegNo,
&                      K4Pit, PDTrans, CAM1st, TmInit, DTSave)
```

c If whole patch failed in this segment, decrement the pits bin array
 c in the bin where failure occurred, and return. Guard against input
 c that may have resulted in negative pit growth by also decrementing
 c from bins pits that broke through after patch failure.

```
c
      if (PatNdx .le. SegEnd(SegNo)) then
        do Pit = 1, NPitCRM
          i = K4Pit(Pit)
          if (i .le. PatchBin .and. i .gt. 0) then
            Bin(PatchBin, PatchType) = Bin(PatchBin, PatchType) - 1
          else if (i .gt. PatchBin) then
            Bin(i, PatchType) = Bin(i, PatchType) - 1
          end if
        end do
        RETURN
      end if
      LastSegSame = .true.
```

```
c
      else if ((CRMod .eq. CRMGLNL) .or. (CRMod .eq. CRMGRNL)) then
```

c Use Arrhenius pitting law to find total pit depths (LLNL).
 c Note that if two kinds of drips fall in two segments, WapDeg earlier
 c required that the pit law parameters were the same in both segments.
 c So, set the reference time for the exponential law, TmInit.

```
c
      if (LastSegSame) then
        TmInit = ntim(SegStart(SegNo - 1))
      else
        TmInit = ntim(SegStart(SegNo))
      end if
```

```
c
      CALL InnerArrLaw(Penet2, NPitCRM, BinFact, BinTerm, Pit1st,
&                      BinStart, Bin, PatchType, SegStart, SegEnd,
&                      NowVal, ntim, ntem, CRMPit, PatchD, SegNo,
&                      K4Pit, PDTrans, CAM1st, TmInit, DTSave)
```

```

C
C If whole patch failed in this segment, decrement the pits bin array
C in the bin where failure occurred, and return. Guard against input
C that may have resulted in negative pit growth by also decrementing
C from bins pits that broke through after patch failure.
C
      if(PatNdx .le. SegEnd(SegNo)) then
        do Pit = 1, NPitCRM
          i = K4Pit(Pit)
          if(i .le. PatchBin .and. i .gt. 0) then
            Bin(PatchBin, PatchType) = Bin(PatchBin, PatchType) - 1
          else if(i .gt. PatchBin) then
            Bin(i, PatchType) = Bin(i, PatchType) - 1
          end if
        end do
        RETURN
      end if
      LastSegSame = .true.

C
      else if(CRMMod .eq. CRMEXP) then
C
C For a TSPA95 run, we have two choices depending on time exponent.
C
        PwrEq1 = AreEqual(PwrIB, 1.D0, 1.D-3)
        if(PwrEq1) then
          call LinTimeIB(Penet2, NPitCRM, BinFact, BinTerm, BinStart,
            & Bin, PatchType, Start, NowVal, ntim, dt,
            & CRMPit, Cntlst, K4PIT, c10Tdt, c10T,
            & DTSave, SumC0)
C
        else
          call NonLinIB(Penet2, NPitCRM, BinFact, BinTerm, BinStart,
            & Bin, PwrIB, PatchType, Start, NowVal, ntim,
            & dt, CRMPit, Cntlst, K4PIT, c10T, DTSave,
            & SumC0)
        end if
      end if
      end do

      RETURN
      END !SUBROUTINE CorrosionNoSP

C
C *****
C
C SUBROUTINE CorrosionYesSP(c10, Penet2, NumPit, BinFact, BinTerm,
C & SpallDep, RMOuter, RMiner, NewPMs,
C & RMOuter2, RMiner2, Outerlst, BinStart,
C & Bin, AquaTrans, PwrIB, XTime, NowVal,
C & Start, ntim, ntem, nr, dt, GPDepth,

```

```

c      &                                Coef2, CritDep, MultRat, C0Pak)
cc
cc  Calculate
cc  1. General corrosion depth history for the patch while accounting for
cc     spalling,
cc  2. Outer barrier pit penetration times
cc  3. Inner barrier pit penetration times
cc  Input : c10, Penet2, NumPit, BinFact, BinTerm, SpallDep, RMOuter,
cc           RMiner, NewPMs, RMOuter2, RMiner2, Outer1st, BinStart,
cc           Bin, AquaTrans, PwrIB, XTime, NowVal, Start, ntim, ntem,
cc           nr, dt, GPDepth, Coef2, CritDep, MultRat, C0Pak
cc  Output: Outer1st, BinStart, Bin
cc  Local : SPon, GPTimFound, Count, Pit, Csp, Cntlst, K4PIT, Temp, RH,
cc           ratecam, pwr, PatchD, ctim, c10Tdt, c10T, rcam, invpwr,
cc           thrdep, Dtime, ratesp, SpallTim, GPTime, ActGPT, CAM1st,
cc           DTSave, PDTrans
cc
cc  Arguments, and dimension parameters from the include file
cc
c      IMPLICIT NONE
c      INCLUDE 'parm306.inc'
c      logical NewPMs
c      integer Bin(MAXBIN, 3), NowVal, NumPit
c      double precision c10, Penet2, BinFact, BinTerm, SpallDep
c      double precision RMOuter, RMiner, RMOuter2, RMiner2
c      double precision Outer1st, BinStart
c      double precision AquaTrans, PwrIB, XTime, Start
c      double precision ntim(*), ntem(*), nr(*), dt(*), GPDepth
c      double precision Coef2(*), CritDep(*), MultRat(*), C0Pak(*)
cc
cc  Local variables
cc
c      logical SPon, GPTimFound
c      integer istart, Count, Pit, Csp, Cntlst, K4PIT(MAXPIT)
c      double precision Temp, RH, ratecam, pwr
c      double precision PatchD(MAXDAT1), ctim(MAXDAT1), c10Tdt(MAXDAT1)
c      double precision c10T(MAXDAT1), rcam(MAXDAT1), invpwr(MAXDAT1)
c      double precision thrdep, Dtime, ratesp, SpallTim, GPTime, ActGPT,
c      &                                CAM1st, DTSave(MAXPIT), PDTrans
c
c      istart      = 2
c      SPon        = .False.
c      GPTimFound = .False.
c      CAM1st      = ntim(NowVal)
c      Cntlst      = NowVal + 1
c      CALL InitDPArray(PatchD, NowVal)
cc
cc  Find the general corrosion depth for this patch at each time step.
cc  When general corrosion depth reaches SpallDep the corresponding

```



```
cc corrosion time and rate are stored in SpallTim and ratesp.
cc
c      do Count = 2, NowVal
c      Temp = ntem(Count)
c      RH = nr(Count)
cc
cc Depending corrosion mode, set rate and power term for time exponent.
cc
c      IF (RH .lt. AquaTrans) then
c      ratecam = EXP(Coef2(1) + Coef2(2)/Temp + Coef2(3)/RH)
c      pwr = Coef2(4)
c      else
c      ratecam = EXP(Coef2(5) + Coef2(7)/Temp + Coef2(8)*Temp*Temp)
c      pwr = Coef2(6)
c      end if
cc
cc Modify the rate for both barriers, depending on the time. Recall
cc that c10T * CoPak is the inner barrier corrosion rate.
cc
c      if(ntim(Count) .le. XTime) then
c      ratecam = ratecam * RMOuter
c      c10T(Count) = EXP(COPatM + C10 / Temp) * RMiner
c      else
c      ratecam = ratecam * RMOuter2
c      c10T(Count) = EXP(COPatM + C10 / Temp) * RMiner2
c      end if
cc
cc Save needed properties of the patch history. If galvanic protection
cc in force, find time step (istart) at which its depth was reached and
cc the time remaining in that step (GPTime). If 2 time periods are being
cc simulated, find corrosion depth at transition time (PDTrans).
cc
c      if(SPon) then
cc
cc Spalling depth has been reached so find corrosion rate based on
cc 'corrosion' time that spalling depth was reached.
cc
c      rcam(Count) = pwr*SpallDep*(ratecam/SpallDep)**(1.0/pwr)
c      PatchD(Count) = PatchD(Count - 1) + rcam(Count) * dt(Count)
c      if(GPDepth .gt. 0.D0 .and. .not. GPTimFound) then
c      if(PatchD(Count) .ge. GPDepth) then
c      istart = Count
c      GPTime = (PatchD(Count) - GPDepth) / rcam(Count)
c      ActGPT = ntim(Count) - GPTime
c      GPTimFound = .true.
c      end if
c      end if
c      else
cc
```

```
cc Spalling depth not yet reached, so find depth at end of time step.
cc
c      ctim(Count) = (PatchD(Count-1)/ratecam)**(1.0/pwr)+dt(Count)
c      rcam(Count) = ratecam
c      PatchD(Count) = ratecam * ctim(Count)**pwr
cc
cc Reach spall depth during this time step? If so, save corrosion rate
cc that applies after spall depth reached for this time step (ratesp).
cc
c      if(PatchD(Count) .gt. SpallDep) then
c          Spon = .true.
c          Csp = Count
c          SpallTim = (SpallDep / ratecam)**(1.00 / pwr)
c          ratesp = SpallDep * pwr / SpallTim
c          PatchD(Count) = SpallDep + ratesp * (ctim(Count) - SpallTim)
cc
c      if(GPDepth .gt. 0.00 .and. .not. GPTimFound) then
c          if(PatchD(Count) .ge. GPDepth) then
c              istart = Count
c              GPTimFound = .true.
c              if(GPDepth .le. SpallDep) then
c                  GPTime = (GPDepth / rcam(Count)) ** invpwr(Count)
c                  GPTime = ctim(Count) - GPTime
c              else
c                  GPTime = (PatchD(Count) - GPDepth) / ratesp
c              end if
c              ActGPT = ntim(Count) - GPTime
c          end if
c      end if
cc
cc Spalling depth not reached during time step, continue with corrosion
cc rate found as if there is no spalling.
cc
c      else
c          if(GPDepth .gt. 0.00 .and. .not. GPTimFound) then
c              if(PatchD(Count) .ge. GPDepth) then
c                  istart = Count
c                  GPTime = (GPDepth / rcam(Count)) ** invpwr(Count)
c                  GPTime = ctim(Count) - GPTime
c                  ActGPT = ntim(Count) - GPTime
c                  GPTimFound = .true.
c              end if
c          end if
c      end if
cc
c      cl0Tdt(Count) = cl0T(Count) * dt(Count)
c      invpwr(Count) = 1.00 / pwr
c      if(ntim(Count) .le. XTime) PDTrans = PatchD(Count)
```

```
c      end do
cc
cc  If patch galvanically protected on inner barrier until outer depth
cc  reaches GPDepth, but outer depth never gets that far, then no pits
cc  will penetrate the outer (let alone the inner barrier).
cc
c      if(GPDepth .gt. 0.D0) then
c          if(.not. GPTimFound) RETURN
c      end if
cc
cc  If two time periods, for those pits that will NOT penetrate the outer
cc  barrier BEFORE the transition time, adjust critical pit depths.
cc
c      if(NewPMs) then
c          do Pit = 1, NumPit
c              if(CritDep(Pit) .gt. PDTrans) then
c                  CritDep(Pit) = MultRat(Pit) * (CritDep(Pit) - PDTrans) +
c                  & PDTrans
c              else
c                  EXIT
c              end if
c          end do
c      end if
cc
cc  For each pit determine the time step at which the pit will be ready
cc  to corrode the inner barrier (Count -> K4PIT) and the time remaining
cc  in that time step (Dtime -> DTSave).
cc
c      do Pit = NumPit, 1, -1
c          if(CritDep(Pit) .le. GPDepth) then
c              DTSave(Pit) = GPTime
c              K4PIT(Pit) = istart
c              CAM1st = ActGPT
c              Cnt1st = istart
cc
c          else
cc
cc  Pit not galvanically inhibited when it reaches the inner barrier, so
cc  corrosion begins when it penetrates the outer barrier.
cc
c              thrdep = CritDep(Pit)
c              Count = istart
c              do while (PatchD(Count) .le. thrdep .AND. Count .le. NowVal)
c                  Count = Count + 1
c              end do
cc
cc  First if clause is: spalling depth was reached before this interval,
cc  which is typically the case, that's why it appears first
cc  Second clause is: spalling depth is reached after this interval
```

```

cc Third clause is: spalling depth was reached in this interval and
cc spalling depth was reached prior to critical depth.
cc Fourth clause is: spalling depth was reached in this interval and
cc spalling depth was reached after critical depth.
cc
c      IF (Count .le. NowVal) then
c          K4PIT(Pit) = Count
c          if (Count .gt. Csp) then
c              Dtime = (PatchD(Count) - thrdep) / rcam(Count)
cc
c          else if (Count .lt. Csp) then
c              Dtime = ctim(Count) - (thrdep/rcam(Count))*invpwr(Count)
cc
c          else if (thrdep .ge. SpallDep) then
c              Dtime = (PatchD(Count) - thrdep) / ratesp
cc
c          else
c              Dtime = (PatchD(Count) - SpallDep) / ratesp +
c              &          SpallTim - (thrdep / rcam(Count))*invpwr(Count)
c          end if
c          DTSave(Pit) = Dtime
c          if(ntim(Count) - Dtime .lt. CAM1st) then
c              CAM1st = ntim(Count) - Dtime
c              Cnt1st = Count
c          end if
cc
c      else
c          DTSave(Pit) = 0.D0
c          K4PIT(Pit) = NowVal + 1
c      end if
c  end if
cc  end do
cc
cc Compare first CAM penetration time for this patch to the others on
cc this package. Store the earliest.
cc
c      Outer1st = min(CAM1st + Start, Outer1st)
cc
cc Simulate inner barrier corrosion.
cc
c      CALL PitInner(Penet2, NumPit, BinFact, BinTerm, BinStart, Bin,
c      &          PwrIB, NowVal, Start, ntim, dt, COPak, Cnt1st,
c      &          K4PIT, cl0Tdt, cl0T, DTSave)
c
c      RETURN
c      END !SUBROUTINE CorrosionYesSP
c
c *****
c

```

```

      SUBROUTINE WritePack(outfid, binfid, patfid, crmfid, Penet2,
&                          InFail, NumPatch, bin0, binf, nbin, BinFact,
&                          ibin1, ibin2, ibin3, ibin4, MSpec, Pack,
&                          Pitlst, Patlst, BinStart, Bin, PatStart,
&                          PatBin, CRMThk)
c
c Provides waste package degradation results to output files
c Input : outfid, binfid, patfid, crmfid, Penet2, InFail, NumPatch,
c         bin0, binf, nbin, BinFact, ibin1, ibin2, ibin3, ibin4, MSpec,
c         Pack, Pitlst, Patlst, BinStart, Bin, PatStart, PatBin, CRMThk
c Output: (none)
c Local : i, j, ilst, ilstpat, SumPit, SumPat, rat, delt, Tm, TotThick,
c         S, Advect, Diffus, Struct, Outlst, Innlst, Frc
c
c Arguments, and include file for dimensions
c
      IMPLICIT NONE
      include 'parm306.inc'
      integer outfid, binfid, patfid, crmfid, NumPatch, nbin, ibin1,
&          ibin2, ibin3, ibin4
      integer Pack, MSpec(*), Bin(MAXBIN, 3), PatBin(MAXBIN, 5)
      double precision Penet2, InFail, bin0, binf, BinFact, Pitlst(*),
&          Patlst(*), BinStart, PatStart, CRMThk(*)
c
c Local variables
c
      integer i, j, ilst, ilstpat, SumPit(4), SumPat(4)
      double precision rat, delt, Tm, TotThick, S(21)
      double precision Advect, Diffus, Struct, Outlst, Innlst, Frc
c
c Convert the pits per bin array to a cumulative pits per bin array.
c
      ilst = 1 + BinFact * log(BinStart / bin0)
      do j = 1, 3
        do i = (ilst + 1), (nbin + 1)
          Bin(i, j) = Bin(i, j) + Bin(i - 1, j)
        end do
      end do
c
c Convert the patches per bin array to a cumulative array.
c
      ilstpat = 1 + BinFact * log(PatStart / bin0)
      do j = 1, 5
        do i = (ilstpat + 1), (nbin + 1)
          PatBin(i, j) = PatBin(i, j) + PatBin(i - 1, j)
        end do
      end do
c
c Print the columns to the OUT file for the package. Advection time is

```

```

c the earliest time when have patch failure for both top patch being
c dripped on and any bottom patch. Diffusion time is the earliest time
c there is any opening in the side or bottom of the package. Change
c CRM thickness array to mean fraction of thickness gone & interpolate
c to find structural failure time.

```

```

c
  Advect = max(Patlst(1), min(Patlst(2), Patlst(5)))
  Diffus = min(min(min(Patlst(2), Patlst(5)), Pitlst(5)), Patlst(4))
  if(MSpec(2) .eq. CRMEXP) then
    Diffus = BinStart
  end if
  Struct = binf
  TotThick = Penet2 * NumPatch
  do i = 1, MAXCRMT
    CRMThk(i) = CRMThk(i) / TotThick
  end do
  delt = binf / dble(MAXCRMT)
  do i = 1, MAXCRMT
    if(CRMThk(i) .gt. InFail) then
      rat = (InFail - CRMThk(i-1)) / (CRMThk(i) - CRMThk(i-1))
      Tm = rat * log(dble(i)/dble(i-1)) + log(dble(i-1) * delt)
      Struct = exp(Tm)
    end if
  end do
  Outlst = min(Pitlst(1), min(Pitlst(2), Pitlst(3)))
  Innlst = min(Diffus, min(Pitlst(4), min(Patlst(1), Patlst(3))))

```

```

c
c Find total cumulative pits and patches gone at the 4 user-input times
c

```

```

  CALL InitIntArray(SumPit, 4)
  CALL InitIntArray(SumPat, 4)
  do i = 1, 3
    SumPit(1) = SumPit(1) + Bin(ibin1, i)
    SumPit(2) = SumPit(2) + Bin(ibin2, i)
    SumPit(3) = SumPit(3) + Bin(ibin3, i)
    SumPit(4) = SumPit(4) + Bin(ibin4, i)
  end do
  do i = 1, 5
    SumPat(1) = SumPat(1) + PatBin(ibin1, i)
    SumPat(2) = SumPat(2) + PatBin(ibin2, i)
    SumPat(3) = SumPat(3) + PatBin(ibin3, i)
    SumPat(4) = SumPat(4) + PatBin(ibin4, i)
  end do
  write(outfid, 10) Pack, Advect, Diffus, Struct, Outlst, Innlst,
    (Pitlst(i), i = 1, 3), (Patlst(i), i = 3, 5),
    (Pitlst(i), i = 4, 5), (Patlst(i), i = 1, 2),
    (SumPit(i), i = 1, 4), (SumPat(i), i = 1, 4)

```

```

c

```

c Write to the BIN file for this package. If the package did not fail,
c only the header line is written. Otherwise, also write the cumulative
c number of pits per bin starting with the bin that contains the first
c pit failure. The header line includes the package number, the first
c bin number, the last bin number, and the time of first failure.
c For TSPA95 runs write the total pits from all patch locations. Else,
c there can be no pit penetrations in side patches, so only write
c results for pits on top & bottom patches (in separate columns).

```
c
      if (BinStart .ge. binf) then
        ilst = nbin + 2
      end if
      write(binfid, 20) Pack, ilst, nbin + 1, BinStart
      if(MSpec(2) .eq. CRMEXP) then
        do i = ilst, nbin + 1
          write(binfid, 30) Bin(i, 1) + Bin(i, 2) + Bin(i, 3)
        end do
      else
        do i = ilst, nbin + 1
          write(binfid, 30) (Bin(i, j)), j = 1, 3, 2)
        end do
      end if
```

c
c Data is written to PAT file for each package. If the package did not
c fail, only the header line is written. Otherwise write the cumulative
c number of patches per bin starting with the bin that contains the
c first patch failure. The header line includes the package number, the
c first bin number, the last bin number, and the time of first failure.
c Note that the column order is: top-drips, bottom-drips, top-nodrips,
c side, bottom-nodrips.

```
c
      if (PatStart .ge. binf) then
        ilstpat = nbin + 2
      end if
      write(patfid, 20) Pack, ilstpat, nbin + 1, PatStart
      do i = ilstpat, nbin + 1
        write(patfid, 35) (PatBin(i, j)), j = 1, 5)
      end do
```

c
c Write the CRM thickness file. Write package number and number of
c lines to follow to the header line. Write time of first CRM
c corrosion and zero fraction gone, then time and fraction gone for
c each 0.05 additional fraction gone.

```
c
      S(1) = Outlst
      ilst = 1
      ilstpat = 21
      do i = 2, 21
        Frc = dble(i - 1) * 0.05D0
```

```

      do j = ilst, MAXCRMT
        if(CRMThk(j) .ge. Frc) then
          rat = (Frc - CRMThk(j - 1)) / (CRMThk(j) - CRMThk(j - 1))
          Tm = rat * log(dble(j)/dble(j-1)) + log(dble(j-1) * delt)
          S(i) = exp(Tm)
          EXIT
        end if
      end do
      ilst = j
      if(ilst .gt. MAXCRMT) then
        ilstpat = i - 1
        EXIT
      end if
      end do
      if(ilstpat .lt. 21) then
        write(crmfid, 40) Pack, ilstpat + 1
      else
        write(crmfid, 40) Pack, ilstpat
      end if
      do i = 1, ilstpat
        Frc = dble(i - 1) * 0.05D0
        write(crmfid, 50) S(i), Frc
      end do
      if(ilstpat .lt. 21) then
        write(crmfid, 50) binf, CRMThk(MAXCRMT)
      end if
C
10  format(i4,2x,5(1x,f9.1),2x,3(1x,f9.1),3x,3(1x,f9.1),2x,
&    4(f9.1,1x),4(1x,i8,1x),1x,4(3x,i4,3x))
20  format(3(1x,i10),1x,f10.2)
30  format(2i9)
35  format(5i6)
40  format(2i5)
50  format(1x,f10.1,1x,f6.3)
      RETURN
      END !SUBROUTINE WritePack
C
C *****
C
      SUBROUTINE GetDotPosit(flname, NameLen, DotPos)
C
C Find the position in flname that an extension can be appended.
C Input : flname, NameLen
C Output: DotPos
C Local : LenTrim, ForSlash, BackSlash, C, Slash
C
C Arguments
C
      IMPLICIT NONE

```



```

character flname*(*)
integer NameLen, DotPos

c
c Local variables
c
character*1 ForSlash, BackSlash, C
integer LenTrim, Slash

c
c Find length of flname when trailing blanks ignored. Some compilers
c have a Len_Trim function to do this.
c
do LenTrim = NameLen, 1, -1
  IF (flname(LenTrim: LenTrim) .ne. ' ') then
    EXIT
  end if
end do

c
c Look backwards for a slash or back-slash. Actual file name assumed
c to occur after the last one. If none found, Slash will be 0.
c
ForSlash = char(47)
BackSlash = char(92)
do Slash = LenTrim, 1, -1
  C = flname(Slash: Slash)
  if(c .eq. ForSlash .or. c .eq. BackSlash) then
    EXIT
  end if
end do

c
c Look backwards for a period denoting an extension: If none found
c before (back-)slash then none exists in file name so can append
c extension at the end of the file name.
c
do DotPos = LenTrim, Slash + 1, -1
  IF (flname(DotPos: DotPos) .eq. '.') then
    EXIT
  end if
end do
if(DotPos .le. Slash) DotPos = LenTrim + 1
RETURN
END      !SUBROUTINE GetDotPosit

c
c *****
c
SUBROUTINE WriteMessage(Line1, Line2, Screen, u1, u2, NW)

c
c Write a two line message (if 2nd line nonblank) to the screen (if
c Screen = true) and to units u1 and u2 (if they are positive). If NW
c is positive, it is the number of the message written to unit u2, so

```

```

c precede the message with its number.
c Input :Line1, Line2, Screen, u1, u2, NW
c Output:(none)
c Local :ident1, ident2
c
c Arguments
c
c     IMPLICIT NONE
c     logical Screen
c     integer u1, u2, NW
c     character Line1*(*), Line2*(*)
c
c Local variables
c
c     character*5 ident1, ident2
c
c     ident2 = ' '
c     if(Screen) then
c         write(*, 9000) Line1
c         if(Line2 .ne. ' ') write(*, 9000) Line2
c     end if
c     if(u1 .gt. 0) then
c         write(u1, 9000) Line1
c         if(Line2 .ne. ' ') write(u1, 9000) Line2
c     end if
c     if(u2 .gt. 0) then
c         if(NW .gt. 0) then
c             ident1 = ' '
c             write(ident1(2: 3), '(i2)') NW
c             write(u2, 9010) ident1, Line1
c             if(Line2 .ne. ' ') write(u2, 9010) ident2, Line2
c         else
c             write(u2, 9000) Line1
c             if(Line2 .ne. ' ') write(u2, 9000) Line2
c         end if
c     end if
c     9000 format(1x,A)
c     9010 format(A5,A)
c     RETURN
c     END      !SUBROUTINE WriteMessage
c
c *****
c
c     SUBROUTINE ReadModelType(inpfid, outfid, auxfid, CDFStack, vg,
c     &                          varshare, ErrFound, CDFStackEnd, RateMax,
c     &                          Cond, Err1, Err2, ModBranch, NumD, DType,
c     &                          DParm, NW, NCRMD, ATemp)
c
c     Read in corrosion model form and parameters, and check validity.

```

```

c Input : inpfid, outfid, auxfid, varshare, CDFStackEnd, Cond, NW
c Output: vg, ErrFound, CDFStackEnd, RateMax, Err1, Err2,
c         ModBranch, DType, DParm, NW, NCRMD, ATemp
c Local : ModType, Mods, Warn1, Warn2, CDFName, VarName, realstr,
c         intstr, i, ADist, BDist, nDist, mDist, CODist, intvar,
c         AreaTot, AreaLeft, cov, covdef, varerr, vardef, pakshare,
c         patshare, AParm, BParm, nParm, mParm, COParm
c
c Arguments, and parameters from include file
c Note dimensions for DType and DParm below are adequate as long as
c MAXCRMDISTS is at least 2.
c
      implicit none
      include 'parm306.inc'
      character*74 Err1, Err2
      logical ErrFound
      integer inpfid, outfid, auxfid, CDFStackEnd, Cond, ModBranch,
&          DType(*), NumD, NW, NCRMD
      double precision CDFStack(MAXPAIRS, 2), DParm(MAXCRMDISTS + 3, 4)
      double precision varshare, vg(2, 3, 4), RateMax, ATemp(*)
c
c Local variables
c
      character*23 ModType, Mods(NCMODELS)
      character*74 Warn1, Warn2
      character CDFName*(NAMELEN), VarName*11, realstr*11, intstr*4
      integer i, ADist, BDist, nDist, mDist, CODist, intvar
      double precision AreaTot, AreaLeft, cov, covdef, varerr, vardef,
&          pakshare, patshare
      double precision AParm(4), BParm(4), nParm(4), mParm(4), COParm(4)
c
      data Mods / 'CAMGENERAL+PITMULTIPLES', 'CAMGENERAL+PITGROWTHLAW',
&          'CRMGENERALONLY', 'CRMGENERALRATEONLY',
&          'CRMGENERAL+PITGROWTHLAW', 'CRMGENRATE+PITGROWTHLAW',
&          'CRMGENERAL+LOGPITGROWTH', 'CRMGENRATE+LOGPITGROWTH',
&          'CRMGENERAL+ARRHENIUSPIT', 'CRMGENRATE+ARRHENIUSPIT',
&          'CRMEXPONENTIALPITRATE' '/'
c
      ErrFound = .false.
      do i = 1, MAXCRMDISTS + 3
        DType(i) = 0
        DParm(i, 1) = 0.D0
        DParm(i, 2) = 0.D0
        DParm(i, 3) = 0.D0
        DParm(i, 4) = 0.D0
      end do
c
c Read model name, convert it to all upper case, and compare to the
c list of currently supported models.

```

```

c      read(inpfid, 8000) ModType
      write(outfid, 8010) ModType
8000 format(a23)
8010 format(a23,T35,'|Type of corrosion model')
c
c      CALL UpCase(ModType, 23)
c
      ModBranch = 0
      do i = 1, NCMODELS
        if(ModType .eq. Mods(i)) then
          ModBranch = i
          EXIT
        end if
      end do
c
c      If user did not supply a legal name, stop with an error. Otherwise
c      read parameters required by the specified model.
c
      if(ModBranch .eq. 0) then
        Err1 = 'ERROR. Unrecognized model type = ' // ModType
        Err2 = 'Program STOP'
        ErrFound = .true.
c
      else if(ModBranch .eq. CAMGPM) then
c
c      General corrosion plus pit multiples (localization factors) for the
c      CAM requires a pit multiple distribution.
c
      CALL InitDPArray(mParm, 4)
      CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, mDist, ,
        &      mParm, CDFName)
      if(ErrFound) RETURN
      if(mDist .eq. DSTDEF) then
        if(Cond .eq. 0) then
          !No drips
          mParm(1) = 1.5D0
          mParm(2) = 0.25D0
          mParm(3) = 1.D0
          mParm(4) = 3.5D0
        else if(Cond .eq. 1) then
          !Connate drips
          mParm(1) = 1.5D0
          mParm(2) = 0.25D0
          mParm(3) = 1.D0
          mParm(4) = 3.5D0
        else if(Cond .eq. 2) then
          !Tainted drips
          mParm(1) = 4.D0
          mParm(2) = 1.D0
          mParm(3) = 1.D0
          mParm(4) = 12.D0
      end if
    end if
  end do
end sub

```

```

        end if
        mDist = DSTBNOR
        CALL GetCutArea(mParm, AreaTot, AreaLeft, ErrFound)
        mParm(3) = AreaTot
        mParm(4) = AreaLeft
    end if
    DType(1) = mDist
    DParm(1, 1) = mParm(1)
    DParm(1, 2) = mParm(2)
    DParm(1, 3) = mParm(3)
    DParm(1, 4) = mParm(4)
    NumD = 1
c
    else if (ModBranch .eq. CAMGPGL) then
c
c General plus pit growth law for the CAM requires distributions for
c the pit growth law only. This mean B and n (B^n term).
c
        CALL InitDPArray(BParm, 4)
        CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, BDist,
&          BParm, CDFName)
        if (ErrFound) RETURN
        intvar = BDist
        VarName = 'CAMB'
        if (intvar .eq. DSTCDF) then
            CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
&          CDFStackEnd, BParm)
            if (ErrFound) RETURN
        else if (intvar .eq. DSTDEF) then
            CALL DefaultCDFs(VarName, CDFStack, Err1, Err2, ErrFound,
&          CDFStackEnd, BParm)
            intvar = DSTCDF
        end if
        DType(1) = intvar
        DParm(1, 1) = BParm(1)
        DParm(1, 2) = BParm(2)
        DParm(1, 3) = BParm(3)
        DParm(1, 4) = BParm(4)
c
        CALL InitDPArray(nParm, 4)
        CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, nDist,
&          nParm, CDFName)
        if (ErrFound) RETURN
        intvar = nDist
        VarName = 'CAMn'
        if (intvar .eq. DSTCDF) then
            CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
&          CDFStackEnd, nParm)
            if (ErrFound) RETURN

```

```

      else if(intvar .eq. DSTDEF) then
        CALL DefaultCDFs(VarName, CDFStack, Err1, Err2, ErrFound,
&          CDFStackEnd, nParm)
        intvar = DSTCDF
      end if
      DType(2)      = intvar
      DParm(2, 1)   = nParm(1)
      DParm(2, 2)   = nParm(2)
      DParm(2, 3)   = nParm(3)
      DParm(2, 4)   = nParm(4)
      NumD = 2
c
      else if(ModBranch .eq. CRMGONLY .or. ModBranch .eq. CRMGPGI .or.
&        ModBranch .eq. CRMGLOG .or. ModBranch .eq. CRMGMLNL) then
c
c All of these models require general corrosion for the CRM. So read
c two distributions for general corrosion rate, their covariance, and
c the error of variance.
c
      CALL InitDPArray(AParm, 4)
      CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, ADist,
&        AParm, CDFName)
      if(ErrFound) RETURN
      intvar = ADist
      if(intvar .eq. DSTDEF) then
        if(Cond .eq. 0) then
          !No drips
          intvar = DSTNOR
          AParm(1) = -0.74696D0
          AParm(2) = SQRT(3.6816D-4)
        else if(Cond .eq. 1) then
          !Connate drips
          intvar = DSTNOR
          AParm(1) = 9.9119D0
          AParm(2) = SQRT(2.6218D-4)
        else if(Cond .eq. 2) then
          !Tainted drips
          intvar = DSTNOR
          AParm(1) = 9.9119D0
          AParm(2) = SQRT(2.6218D-4)
        end if
      end if
      if(intvar .ne. DSTNOR .and. intvar .ne. DSTFXD) then
        Err1 = 'ERROR. Distribution for log(A) for general ' //
&        'CRM corrosion rate is required'
        Err2 = 'to be normal or fixed. Program STOP'
        ErrFound = .true.
        RETURN
      end if
      DType(1)      = intvar
      DParm(1, 1)   = AParm(1)
      DParm(1, 2)   = AParm(2)

```

```

DParam(1, 3) = AParam(3)
DParam(1, 4) = AParam(4)

c
CALL InitDParamArray(BParam, 4)
CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, BDist,
&      BParam, CDFName)
  if(ErrFound) RETURN
  intvar = BDist
  if(intvar .eq. DSTDEF) then
    if(Cond .eq. 0) then
      !No drips
      intvar = DSTNOR
      BParam(1) = 3.1322D3
      BParam(2) = SQRT(38.789D0)
    else if(Cond .eq. 1) then
      !Connate drips
      intvar = DSTNOR
      BParam(1) = 5818.3D0
      BParam(2) = SQRT(27.623D0)
    else if(Cond .eq. 2) then
      !Tainted drips
      intvar = DSTNOR
      BParam(1) = 5818.3D0
      BParam(2) = SQRT(27.623D0)
    end if
  end if
  if(intvar .ne. DSTNOR .and. intvar .ne. DSTFXD) then
    Err1 = 'ERROR. Distribution for K for general ' //
    &      'CRM corrosion rate is required'
    Err2 = 'to be normal or fixed. Program STOP'
    ErrFound = .true.
    RETURN
  end if
  DType(2) = intvar
  DParam(2, 1) = BParam(1)
  DParam(2, 2) = BParam(2)
  DParam(2, 3) = BParam(3)
  DParam(2, 4) = BParam(4)

c
c Set the default covariance and default variance error, in case both
c parameters were defaulted.
c
  covdef = 0.D0
  vardef = 0.D0
  if(ADist .eq. DSTDEF .and. BDist .eq. DSTDEF) then
    if(Cond .eq. 0) then
      !No drips
      covdef = -0.11892D0
      vardef = 1.0755D0
    else if(Cond .eq. 1) then
      !Connate drips
      covdef = -0.084687D0
      vardef = 0.76593D0
    else if(Cond .eq. 2) then
      !Tainted drips

```

```

        covdef = -0.084687D0
        vardef = 0.76593D0
    end if
end if
c
    read(inpfid, *) cov, varerr
    write(outfid, 8040) cov, varerr
8040    format(2f10.3,T35,'|Covariance & error term for variance')
c
    if(DType(1) .eq. DSTFXD .or. DType(2) .eq. DSTFXD) then
        if(abs(cov) .gt. 0.D0) then
            Warn1 = 'Covariance for CRM rate parameters is ignored ' //
                    'when at least one of '
            Warn2 = 'the parameters is fixed.'
            NW     = NW + 1
            CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
        end if
        cov = 0.D0
    else
        if(abs(cov) .le. 0.D0 .and. abs(covdef) .gt. 0.D0) then
            cov = covdef
            write(realstr, '(1pell.4)') cov
            Warn1 = 'Covariance for CRM rate parameters is ' //
                    'defaulted to ' // realstr
            Warn2 = ' '
            NW     = NW + 1
            CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
        end if
    end if
c
    if(DType(1) .eq. DSTFXD .and. DType(2) .eq. DSTFXD) then
        if(abs(varerr) .gt. 0.D0) then
            Warn1 = 'Variance error for CRM rate parameters is ' //
                    'ignored when both '
            Warn2 = 'parameters are fixed.'
            NW     = NW + 1
            CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
        end if
        varerr = 0.D0
    else
        if(abs(varerr) .le. 0.D0 .and. abs(vardef) .gt. 0.D0) then
            varerr = vardef
            write(realstr, '(1pell.4)') varerr
            Warn1 = 'Error of variance for CRM rate parameters is ' //
                    'defaulted to ' // realstr
            Warn2 = ' '
            NW     = NW + 1
            CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
        end if
    end if

```



```

      end if
c
c Establish the decomposed covariance array, including the variance
c share for packages and patches.
c
      do i = 1, 4
        vg(1, Cond + 1, i) = 0.D0
        vg(2, Cond + 1, i) = 0.D0
      end do
      if(DType(1) .ne. DSTFXD .and. DType(2) .ne. DSTFXD) then
        vg(1, Cond + 1, 1) = AParm(2)
        vg(1, Cond + 1, 2) = cov / vg(1, Cond + 1, 1)
        vg(1, Cond + 1, 3) = sqrt(BParm(2) * BParm(2) -
&                               vg(1, Cond + 1, 2) * vg(1, Cond + 1, 2))
      end if
      if(DType(1) .ne. DSTFXD .or. DType(2) .ne. DSTFXD) then
        vg(1, Cond + 1, 4) = sqrt(varerr)
      end if
      pakshare = sqrt(varshare)
      patshare = sqrt(1.D0 - varshare)
      do i = 1, 4
        vg(2, Cond + 1, i) = vg(1, Cond + 1, i)
        vg(1, Cond + 1, i) = vg(1, Cond + 1, i) * pakshare
        vg(2, Cond + 1, i) = vg(2, Cond + 1, i) * patshare
      end do
c
c Read an upper bound for CRM corrosion rate.
c
      read(inpfid, *) RateMax
      write(outfid, 8045) RateMax
8045 format(1pe10.3,T35,'|Max general CRM corrosion rate')
c
      if(RateMax .le. 0) then
        Err1 = 'ERROR. Maximum general corrosion rate for inner ' //
&            'barrier must be positive.'
        Err2 = 'Program STOP'
        ErrFound = .true.
        RETURN
      end if
      NumD = 2
c
      elseif(ModBranch .eq. CRMGRONLY .or. ModBranch .eq. CRMGRPGL .or.
&           ModBranch .eq. CRMGRLOG .or. ModBranch .eq. CRMGRNLN) then
c
c All of these models require general corrosion rate for the CRM. So
c read temperature-dependent distributions for general corrosion rate
c and the maximum corrosion rate.
c
      read(inpfid, *) NCRMD, RateMax

```

```

write(outfid, 80455) NCRMD, RateMax
80455  format(i10,lpe10.3,T35,'|Num CRM rate dists, Max general rate')
C
if(RateMax .le. 0) then
  Err1 = 'ERROR. Maximum general corrosion rate for inner ' //
  &      'barrier must be positive.'
  Err2 = 'Program STOP'
  ErrFound = .true.
  RETURN
end if

C
if(NCRMD .lt. 2 .or. NCRMD .gt. MAXCRMDISTS) then
  write(intstr, '(i4)') MAXCRMDISTS
  Err1 = 'ERROR. Number of distributions for CRM general ' //
  &      'corrosion rate must be '
  Err2 = 'from 2 to ' // intstr // '. Program STOP'
  ErrFound = .true.
  RETURN
end if
do i = 1, NCRMD
  read(inpfid, *) ATemp(i)
  write(outfid, 8046) ATemp(i)
8046  format(f10.3,T35,'|Temperature appropriate for distribution')
  if(i .gt. 1) then
    if(ATemp(i) .le. ATemp(i - 1)) then
      Err1 = 'ERROR. Order of input here must be according ' //
      &      'to increasing temperature.'
      Err2 = 'Program STOP'
      ErrFound = .true.
      RETURN
    end if
  end if
  CALL InitDPArray(AParm, 4)
  CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, ADist,
  &              AParm, CDFName)
  if(ErrFound) RETURN
  intvar = ADist
  if(intvar .eq. DSTDEF) then
    !Deal with default later DL9999
    elseif (intvar .eq. DSTCDF) then
      CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
      &                  CDFStackEnd, AParm)
      if (ErrFound) RETURN
    end if
    DType(i) = intvar
    DParm(i, 1) = AParm(1)
    DParm(i, 2) = AParm(2)
    DParm(i, 3) = AParm(3)
    DParm(i, 4) = AParm(4)

```

```

        end do
        NumD = NCRMD
    end if

C
C Done if doing general rate only, otherwise read more distributions.
C
    if (ModBranch .eq. CRMGPGL .or. ModBranch .eq. CRMGRPGL) then
C
C General plus pit growth law for CRM requires two distributions for
C the pit growth law (John Scully).
C
        CALL InitDPArray(BParm, 4)
        CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, BDist,
        &               BParm, CDFName)
        if (ErrFound) RETURN
        intvar = BDist
        if (intvar .eq. DSTCDF) then
            CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
            &               CDFStackEnd, BParm)
            if (ErrFound) RETURN
        else if (intvar .eq. DSTDEF) then
            if (Cond .eq. 1) then                !Connate drips
                VarName = 'GDCRMB'
            else if (Cond .eq. 2) then            !Tainted drips
                VarName = 'BDCRMB'
            end if
            CALL DefaultCDFs(VarName, CDFStack, Err1, Err2, ErrFound,
            &               CDFStackEnd, BParm)
            intvar = DSTCDF
        end if
        DType(NumD + 1) = intvar
        DParm(NumD + 1, 1) = BParm(1)
        DParm(NumD + 1, 2) = BParm(2)
        DParm(NumD + 1, 3) = BParm(3)
        DParm(NumD + 1, 4) = BParm(4)
C
        CALL InitDPArray(nParm, 4)
        CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, nDist,
        &               nParm, CDFName)
        if (ErrFound) RETURN
        intvar = nDist
        if (intvar .eq. DSTCDF) then
            CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
            &               CDFStackEnd, nParm)
            if (ErrFound) RETURN
        else if (intvar .eq. DSTDEF) then
            if (Cond .eq. 1) then                !Connate drips
                VarName = 'GDCRMn'
            else if (Cond .eq. 2) then            !Tainted drips

```

```

        VarName = 'BDCRMn'
    end if
    CALL DefaultCDFs(VarName, CDFStack, Err1, Err2, ErrFound,
&         CDFStackEnd, nParm)
    intvar = DSTCDF
    end if
    DType(NumD + 2) = intvar
    DParm(NumD + 2, 1) = nParm(1)
    DParm(NumD + 2, 2) = nParm(2)
    DParm(NumD + 2, 3) = nParm(3)
    DParm(NumD + 2, 4) = nParm(4)
    NumD = NumD + 2
c
    else if (ModBranch .eq. CRMGLOG .or. ModBranch .eq. CRMGRLOG) then
c
c General plus logarithmic pit law for the CRM requires 3 distributions
c for the pit growth law. (David Shoesmith)
c
    CALL InitDPArray(BParm, 4)
    CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, BDist,
&         BParm, CDFName)
    if (ErrFound) RETURN
    intvar = BDist !B aka K100
    if (intvar .eq. DSTCDF) then
&         CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
&         CDFStackEnd, BParm)
    if (ErrFound) RETURN
    else if (intvar .eq. DSTDEF) then
    if (Cond .eq. 1) then !Connate drips
        intvar = DSTNOR
        BParm(1) = 0.75D0
        BParm(2) = 0.29D0 / 1.96D0
    else if (Cond .eq. 2) then !Tainted drips
        intvar = DSTNOR
        BParm(1) = 0.75D0
        BParm(2) = 0.29D0 / 1.96D0
    end if
    end if
    DType(NumD + 1) = intvar
    DParm(NumD + 1, 1) = BParm(1)
    DParm(NumD + 1, 2) = BParm(2)
    DParm(NumD + 1, 3) = BParm(3)
    DParm(NumD + 1, 4) = BParm(4)
c
    CALL InitDPArray(nParm, 4)
    CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, nDist,
&         nParm, CDFName)
    if (ErrFound) RETURN
    intvar = nDist !n aka x0

```

```

      if(intvar .eq. DSTCDF) then
        CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
          & CDFStackEnd, nParm)
        if(ErrFound) RETURN
      else if(intvar .eq. DSTDEF) then
        if(Cond .eq. 1) then
          intvar = DSTNOR
          nParm(1) = 1.2D0
          nParm(2) = 0.8D0 / 1.96D0
          !Connate drips
        else if(Cond .eq. 2) then
          intvar = DSTNOR
          nParm(1) = 1.2D0
          nParm(2) = 0.8D0 / 1.96D0
          !Tainted drips
        end if
      end if
      DType(NumD + 2) = intvar
      DParm(NumD + 2, 1) = nParm(1)
      DParm(NumD + 2, 2) = nParm(2)
      DParm(NumD + 2, 3) = nParm(3)
      DParm(NumD + 2, 4) = nParm(4)
C
      CALL InitDPArray(AParm, 4)
      CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, ADist,
        & AParm, CDFName)
      if(ErrFound) RETURN
      intvar = ADist
      !A aka Q
      if(intvar .eq. DSTCDF) then
        CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
          & CDFStackEnd, AParm)
        if(ErrFound) RETURN
      else if(intvar .eq. DSTDEF) then
        if(Cond .eq. 1) then
          intvar = DSTNOR
          AParm(1) = 6615.D0
          AParm(2) = 2044.D0 / 1.96D0
          !Connate drips
        else if(Cond .eq. 2) then
          intvar = DSTNOR
          AParm(1) = 6615.D0
          AParm(2) = 2044.D0 / 1.96D0
          !Tainted drips
        end if
      end if
      DType(NumD + 3) = intvar
      DParm(NumD + 3, 1) = AParm(1)
      DParm(NumD + 3, 2) = AParm(2)
      DParm(NumD + 3, 3) = AParm(3)
      DParm(NumD + 3, 4) = AParm(4)
      NumD = NumD + 3
C
      else if(ModBranch .eq. CRMGLNL .or. ModBranch .eq. CRMGLNL) then

```

c
 c Exponential pit rate model for the CRM requires distributions for A
 c and K, and one for n in $\exp(A - K/T) t^n$.
 c

```

      CALL InitDPArray(AParm, 4)
      CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, ADist,
&          AParm, CDFName)
      if(ErrFound) RETURN
      intvar = ADist
      if(intvar .eq. DSTCDF) then
        CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
&          CDFStackEnd, AParm)
        if(ErrFound) RETURN
      else if(intvar .eq. DSTDEF) then
        intvar = DSTNOR
        AParm(1) = 13.409D0
        AParm(2) = 1.5092D0
      end if
      DType(NumD + 1) = intvar
      DParam(NumD + 1, 1) = AParm(1)
      DParam(NumD + 1, 2) = AParm(2)
      DParam(NumD + 1, 3) = AParm(3)
      DParam(NumD + 1, 4) = AParm(4)

```

c

```

      CALL InitDPArray(BParm, 4)
      CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, BDist,
&          BParm, CDFName)
      if(ErrFound) RETURN
      intvar = BDist
      if(intvar .eq. DSTCDF) then
        CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
&          CDFStackEnd, BParm)
        if(ErrFound) RETURN
      else if(intvar .eq. DSTDEF) then
        intvar = DSTFXD
        BParm(1) = 5558.7D0
        BParm(2) = 0.D0
      end if
      DType(NumD + 2) = intvar
      DParam(NumD + 2, 1) = BParm(1)
      DParam(NumD + 2, 2) = BParm(2)
      DParam(NumD + 2, 3) = BParm(3)
      DParam(NumD + 2, 4) = BParm(4)

```

c

```

      CALL InitDPArray(nParm, 4)
      CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, nDist,
&          nParm, CDFName)
      if(ErrFound) RETURN
      intvar = nDist

```

```

      if(intvar .eq. DSTCDF) then
        CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
          CDFStackEnd, nParm)
      &
        if(ErrFound) RETURN
      else if(intvar .eq. DSTDEF) then
        intvar = DSTFXD
        nParm(1) = 0.5D0
      end if
      DType(NumD + 3) = intvar
      DParm(NumD + 3, 1) = nParm(1)
      DParm(NumD + 3, 2) = nParm(2)
      DParm(NumD + 3, 3) = nParm(3)
      DParm(NumD + 3, 4) = nParm(4)
      NumD = NumD + 3
    end if
  c
    if(ModBranch .eq. CRMEXP) then
  c
  c TSPA95 exponential pit rate model for the CRM requires a distribution
  c for C0, a value for C1, and a value for time exponent. Note the read
  c of constant C1 is in between reads. Value read in is returned in
  c DParm(2, 1) and if defaulted, value used is returned in DParm(2, 2).
  c Require that the distribution for C0 is normal.
  c
    CALL InitDPArray(COParm, 4)
    CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, CODist,
      COParm, CDFName)
    &
    if(ErrFound) RETURN
    if(CODist .eq. DSTDEF) then
      CODist = DSTNOR
      COParm(1) = 29.0046D0
      COParm(2) = log(10.D0) / 1.645D0
    end if
    if(CODist .ne. DSTNOR .and. CODist .ne. DSTFXD) then
      Err1 = 'ERROR. Distribution for C0 in TSPA95 exponential ' //
        'pit rate model must'
      &
      Err2 = 'be a normal or fixed distribution. Program STOP'
      ErrFound = .true.
      RETURN
    end if
    DType(1) = CODist
    DParm(1, 1) = COParm(1)
    DParm(1, 2) = COParm(2)
    DParm(1, 3) = COParm(3)
    DParm(1, 4) = COParm(4)
  c
    read(inpfid, *) DParm(2, 1)
    write(outfid, 8030) DParm(2, 1)
8030 format(1p11.4,T35,'Temperature coefficient')

```

```

      if(DParm(2, 1) .le. 0.D0) then
        DParm(2, 1) = -11682.27D0
      else
        DParm(2, 1) = -DParm(2, 1)
      end if
c
c Read the power for the time term
c
      CALL InitDPArray(nParm, 4)
      CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, nDist,
        &          nParm, CDFName)
      if(ErrFound) RETURN
      if(nDist .eq. DSTDEF) then
        nDist = DSTFXD
        nParm(1) = 1.0D0
        nParm(2) = 1.0D0
      end if
      DType(3) = nDist
      DParm(3, 1) = nParm(1)
      DParm(3, 2) = nParm(2)
      DParm(3, 3) = nParm(3)
      DParm(3, 4) = nParm(4)
      NumD = 3
    end if
    RETURN
  END      !SUBROUTINE ReadModelType
c
c *****
c
      SUBROUTINE DistRead(inpfid, outfid, ErrFound, Err1, Err2,
        &          DistType, Parm, CDFName)
c
c Read distribution type and parameters and check validity.
c Input : inpfid, outfid
c Output: ErrFound, Err1, Err2, DistType, Parm, CDFName
c Local : Dist, Dists, OK, i, NPairs, A1, A2
c
c Arguments, and parameters from include file
c
      implicit none
      include 'parm306.inc'
      character CDFName*(NAMELEN)
      character*74 Err1, Err2
      logical errFound
      integer inpfid, outfid, DistType
      double precision Parm(4)
c
c Local variables
c

```



```

character*10 Dist, Dists(NODISTS)
logical OK
integer i, NPairs
double precision A1, A2
data Dists / 'DEFAULT'  , 'FIXED'    , 'UNIFORM'  ,
&            'NORMAL'   , 'LOGUNIFORM', 'LOGNORMAL' ,
&            'FILE'     , 'B-NORMAL'  /

c
c Read and write distribution name.
c
  ErrFound = .false.
  read(inpfid, '(a10)') Dist
  write(outfid, 8000) Dist
8000 format(a10,T35,'|Distribution type')

c
c Change input to all upper case and compare to supported list.
c
  CALL UpCase(Dist, 10)
  DistType = 0
  do i = 1, NODISTS
    if(Dist .eq. Dists(i)) then
      DistType = i
      EXIT
    end if
  end do

c
c If unknown distribution, then error. If the text is 'Default', then
c the calling program is responsible for substituting in appropriate
c values. If the text is 'File', WapDeg expects a piecewise linear CDF
c on the file name that follows. Fixed type requires one parameter
c and at this generic point, it can be any value. All other currently
c supported distributions require two values. We can make some checks
c at this time.
c
  CALL InitDPArray(Parm, 4)
  if(DistType .eq. 0) then
    Err1 = 'ERROR. Unrecognized distribution type = ' // Dist
    Err2 = 'Program STOP'
    ErrFound = .true.

c
  else if(DistType .eq. DSTDEF) then !Use default distribution

c
  else if(DistType .eq. DSTFXD) then !Fixed value
    read(inpfid, *) Parm(1)
    write(outfid, 8010) Parm(1)
8010 format(lp11.4,T35,'|Fixed value')

c
  else if(DistType .eq. DSTBNOR) then !Bounded normal, 4 parameters
    read(inpfid, *) Parm(1), Parm(2), Parm(3), Parm(4)

```

```
      write(outfid, 8080) Parm(1), Parm(2), Parm(3), Parm(4)
8080      format(4(1pe9.2,1x),
&          T41,'|Mean, std dev, lower/upper bnd')
      if(Parm(2) .le. 0.D0) then
          Err1 = 'ERROR. Bounded normal distributions require ' //
&              'positive standard deviations'
          Err2 = '(second parameter). Program STOP'
          ErrFound = .true.
      end if
      if(Parm(4) .le. Parm(3)) then
          Err1 = 'ERROR. Bounded normal distributions require ' //
&              'lower bound < upper bound'
          Err2 = 'Program STOP'
          ErrFound = .true.
      end if
      CALL GetCutArea(Parm, A1, A2, ErrFound)
      if(ErrFound) then
          Err1 = 'ERROR. Cutoffs for the BoundedNormal distribution ' //
&              'leave insufficient area '
          Err2 = 'left to sample. Program STOP'
          ErrFound = .true.
      end if
      Parm(3) = A1
      Parm(4) = A2
c
      else if(DistType .eq. DSTCDF) then. !Read CDF from file
          read(inpfid, '(a30)') CDFName
          write(outfid, 8020) CDFName
8020      format(a30,T35,'|File name')
          inquire(file = CDFName, exist = OK)
          if(.not. OK) then
              Err1 = 'ERROR. File containing CDF not available. ' //
&                  'Program STOP'
              Err2 = 'File name = ' // CDFName
              ErrFound = .true.
          else
              open(33, file = CDFName)
              read(33, *, end = 1000, err = 1010) NPairs
              do i = 1, NPairs
                  read(33, *, end = 1000, err = 1010) Parm(1), Parm(2)
              end do
              close(33)
              RETURN
1000      continue
              Err1 = 'ERROR. Encountered end-of-file on CDF file ' //
&                  CDFName
              Err2 = 'It has fewer records than first entry indicates. ' //
&                  'Program STOP '
              ErrFound = .true.
```

```
      RETURN
1010    continue
      Err1 = 'ERROR. CDF file, ' // CDFName // ', has incorrect ' //
      &      'structure.'
      Err2 = 'Need # of records, N, followed by N pairs of ' //
      &      'values. Program STOP '
      ErrFound = .true.
      RETURN
    end if
  C
  else                                     !Other dists require 2 params
    read(inpfid, *) Parm(1), Parm(2)
  C
    if(DistType .eq. DSTUNI) then          !Uniform
      write(outfid, 8030) Parm(1), Parm(2)
8030    format(2(1p11.4,1x),T35,'|Minimum and maximum')
      if(Parm(1) .ge. Parm(2)) then
        Err1 = 'ERROR. For uniform distribution, require ' //
        &      'that first parameter (minimum)'
        Err2 = 'is less than second (maximum). Program STOP'
        ErrFound = .true.
      end if
    else if(DistType .eq. DSTNOR) then      !Normal
      write(outfid, 8040) Parm(1), Parm(2)
8040    format(2(1p11.4,1x),T35,'|Mean and standard deviation')
      if(Parm(2) .le. 0.D0) then
        Err1 = 'ERROR. For normal distribution, require ' //
        &      'positive standard deviation'
        Err2 = '(second parameter). Program STOP'
        ErrFound = .true.
      end if
    else if(DistType .eq. DSTLOGU) then     !Loguniform
8050    write(outfid, 8050) Parm(1), Parm(2)
      format(2(1p11.4,1x),T35,'|Minimum and maximum')
      if(Parm(1) .ge. Parm(2)) then
        Err1 = 'ERROR. For loguniform distribution, require ' //
        &      'that first parameter (minimum)'
        Err2 = 'is less than second (maximum). Program STOP'
        ErrFound = .true.
      else if(Parm(1) .le. 0.D0) then
        Err1 = 'ERROR. For loguniform distribution, require ' //
        &      'two positive bounds.'
        Err2 = 'Program STOP'
        ErrFound = .true.
      end if
    else if(DistType .eq. DSTLOGN) then     !Lognormal
8060    write(outfid, 8060) Parm(1), Parm(2)
      format(2(1p11.4,1x),T35,'|Mean and std dev (for normal)')
      if(Parm(2) .le. 0.D0) then
```

```

      Err1 = 'ERROR. For lognormal distribution, require ' //
&      'positive standard deviation'
      Err2 = '(second parameter) for underlying normal. ' //
&      'Program STOP'
      ErrFound = .true.
    end if
  end if
end if
RETURN
END      !SUBROUTINE DistRead

C
C *****
C
      SUBROUTINE ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
&      CDFStackEnd, vard)
C
C  Read a CDF table from the file CDFName and stack the data pairs into
C  the CDFStack array. Starting and ending indexes in CDFStack are
C  returned in vard.
C  Input : CDFName, CDFStack, CDFStackEnd
C  Output: CDFStack, CDFStackEnd, vard, Err1, Err2, ErrFound
C  Local : NPairs, j
C
C  Arguments, and parameters from include file
C
      IMPLICIT NONE
      include 'parm306.inc'
      character CDFName*(NAMELEN), Err1*74, Err2*74
      logical ErrFound
      integer CDFStackEnd
      double precision CDFStack(MAXPAIRS, 2), vard(2)
C
C  Local variables
C
      integer NPairs, j, i
C
      open(33, file = CDFName)
      read(33, *) NPairs
      j = CDFStackEnd
      if(NPairs + j .gt. MAXPAIRS) then
        Err1 = 'ERROR. Too many data pairs for all look-up ' //
&        'tables combined.'
        Err2 = 'Program STOP'
        ErrFound = .true.
        close(33)
      else
        do i = 1, NPairs
          read(33, *) CDFStack(i + j, 1), CDFStack(i + j, 2)
        end do
      end if

```

```

      close(33)
      vard(1)      = CDFStackEnd + 1
      CDFStackEnd = CDFStackEnd + NPairs
      vard(2)      = CDFStackEnd
    end if
    RETURN
  END      !SUBROUTINE ReadTable

```

```

c
c .....
c

```

```

      SUBROUTINE ReadFeatures(inpfid, outfid, auxfid, Penet1,
&                             DTypeStack, DParmStack, EndStack,
&                             CDFStack, Feat, ErrFound, StackIt, Cond,
&                             Err1, Err2, NW, CDFStackEnd)

```

```

c
c Read properties found in the Features segments.
c Input : inpfid, outfid, auxfid, Penet1, StackIt, Cond, NW,
c         CDFStackEnd
c Output: DTypeStack, DParmStack, EndStack, CDFStack, Feat, ErrFound,
c         Err1, Err2, NW, CDFStackEnd
c Local : Warn1, Warn2, CDFName, VarName, realstr, CorrChk1, CorrChk2,
c         intvar, var1, var2, vard, pctgp, pctprotec, spallamnt
c
c Arguments, and dimensioning parameters from include file
c

```

```

      IMPLICIT NONE
      INCLUDE 'parm306.inc'
      character*74 Err1, Err2
      logical ErrFound
      integer inpfid, outfid, auxfid, StackIt, Cond, NW, CDFStackEnd
      integer EndStack(*), DTypeStack(*)
      double precision DParmStack(MAXDISTS, *), CDFStack(MAXPAIRS, 2)
      double precision Penet1, Feat(*)

```

```

c
c Local variables
c

```

```

      character*74 Warn1, Warn2
      character CDFName*(NAMELEN), VarName*11, realstr*11
      integer intvar, CorrChk1, CorrChk2
      double precision var1, var2, vard(4), pctgp, pctprotec, spallamnt

```

```

c
c Read distribution for thermal protection temperature for CAM.
c

```

```

      CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, intvar, vard,
&                  CDFName)
      if(ErrFound) RETURN
      if(intvar .eq. DSTCDF) then
        CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
&                      CDFStackEnd, vard)

```

```

if(ErrFound) RETURN
else if(intvar .eq. DSTDEF) then
  if(Cond .eq. 0) then                !No drips
    VarName = 'NDTmpThresh'
  else if(Cond .eq. 1) then           !Connate drips
    VarName = 'GDTmpThresh'
  else if(Cond .eq. 2) then           !Tainted drips
    VarName = 'BDTmpThresh'
  end if
  CALL DefaultCDFs(VarName, CDFStack, Err1, Err2, ErrFound,
    & CDFStackEnd, vard)
  intvar = DSTCDF
end if
EndStack(StackIt) = EndStack(StackIt - 1) + 1
DTypeStack(EndStack(StackIt)) = intvar
DParmStack(EndStack(StackIt), 1) = vard(1)
DParmStack(EndStack(StackIt), 2) = vard(2)
DParmStack(EndStack(StackIt), 3) = vard(3)
DParmStack(EndStack(StackIt), 4) = vard(4)

C
C If are reading input for dripping conditions, read distribution for
C temperature adequate to initiate pitting on CRM.
C
  if(Cond .gt. 0) then
    CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, intvar,
      & vard, CDFName)
    if(ErrFound) RETURN
    if(intvar .eq. DSTCDF) then
      CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
        & CDFStackEnd, vard)
      if(ErrFound) RETURN
    else if(intvar .eq. DSTDEF) then
      intvar = DSTUNI
      vard(1) = 60.D0
      vard(2) = 70.D0
    end if
    EndStack(StackIt) = EndStack(StackIt) + 1
    DTypeStack(EndStack(StackIt)) = intvar
    DParmStack(EndStack(StackIt), 1) = vard(1)
    DParmStack(EndStack(StackIt), 2) = vard(2)
    DParmStack(EndStack(StackIt), 3) = vard(3)
    DParmStack(EndStack(StackIt), 4) = vard(4)
  end if

C
C Read distribution for humid-air corrosion initiation RH.
C
  CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, intvar, vard,
    & CDFName)
  if(ErrFound) RETURN

```

```

if(intvar .eq. DSTCDF) then
  CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
    & CDFStackEnd, vard)
  if(ErrFound) RETURN
else if(intvar .eq. DSTDEF) then
  if(Cond .eq. 0) then          !No drips
    VarName = 'NDHAThresh'
  else if(Cond .eq. 1) then      !Connate drips
    VarName = 'GDHAThresh'
  else if(Cond .eq. 2) then      !Tainted drips
    VarName = 'BDHAThresh'
  end if
  CALL DefaultCDFs(VarName, CDFStack, Err1, Err2, ErrFound,
    & CDFStackEnd, vard)
  intvar = DSTCDF
end if
EndStack(StackIt) = EndStack(StackIt) + 1
DTypeStack(EndStack(StackIt)) = intvar
DParmStack(EndStack(StackIt), 1) = vard(1)
DParmStack(EndStack(StackIt), 2) = vard(2)
DParmStack(EndStack(StackIt), 3) = vard(3)
DParmStack(EndStack(StackIt), 4) = vard(4)
CorrChk1 = intvar
c
c Read distribution for aqueous corrosion initiation RH.
c
  CALL DistRead(inpfid, outfid, ErrFound, Err1, Err2, intvar, vard,
    & CDFName)
  if(ErrFound) RETURN
  if(intvar .eq. DSTCDF) then
    CALL ReadTable(CDFName, CDFStack, Err1, Err2, ErrFound,
      & CDFStackEnd, vard)
    if(ErrFound) RETURN
  else if(intvar .eq. DSTDEF) then
    if(Cond .eq. 0) then          !No drips
      VarName = 'NDAQThresh'
    else if(Cond .eq. 1) then      !Connate drips
      VarName = 'GDAQThresh'
    else if(Cond .eq. 2) then      !Tainted drips
      VarName = 'BDAQThresh'
    end if
    CALL DefaultCDFs(VarName, CDFStack, Err1, Err2, ErrFound,
      & CDFStackEnd, vard)
    intvar = DSTCDF
  end if
  EndStack(StackIt) = EndStack(StackIt) + 1
  DTypeStack(EndStack(StackIt)) = intvar
  DParmStack(EndStack(StackIt), 1) = vard(1)
  DParmStack(EndStack(StackIt), 2) = vard(2)

```

```

      DParmStack(EndStack(StackIt), 3) = vard(3)
      DParmStack(EndStack(StackIt), 4) = vard(4)
      CorrChk2 = intvar
c
c Read correlation between humidair initiation distribution and aqueous
c initiation distribution. Only read if both the last two are nonfixed
c distributions.
c
      Feat(1) = 1.D0
      if(CorrChk1 .ne. DSTFXD .and. CorrChk2 .ne. DSTFXD) then
        read(inpfid, *) Feat(1)
        write(outfid, 8000) Feat(1)
8000    format(f10.2,T35,'(Correlation between RH thresholds')
        var1 = Feat(1)
        if(var1 .lt. -1.D0 .or. var1 .gt. 1.D0) then
          Warn1 = 'Correlation between distributions for humid-air ' //
          & 'and aqueous corrosion'
          Warn2 = 'initiation defaulted to 1.'
          NW = NW + 1
          CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
          var1 = 1.D0
        end if
c
c *****Currently hardwire Feat(1) to 1.*****
c
      if(abs(var1 - 1.D0) .gt. 0.D0) then
        Warn1 = 'Correlation between distributions for humid-air ' //
        & 'and aqueous corrosion'
        Warn2 = 'initiation defaulted to 1.'
        NW = NW + 1
        CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
        var1 = 1.D0
      end if
      Feat(1) = var1
    end if
c
c If outer barrier corrosion depth for a patch < 'pctgp' percent of its
c thickness, galvanic protection is assumed for the inner barrier. If
c 'pctgp' > 0, then 'pctprotec' is the percentage of patches on a
c package that have galvanic protection.
c
      read(inpfid, *) pctgp, pctprotec
      write(outfid, 8010) pctgp, pctprotec
8010    format(2f10.2,T35,'( galvanic protect depth, %patches protected')
      var1 = pctgp
      var2 = pctprotec
      if(var1 .gt. 0.D0) then
        if(var2 .le. 0.D0 .or. var2 .gt. 100.D0) then
          Warn1 = 'The percentage of patches subject to galvanic ' //

```



```

&      'protection'
Warn2 = 'is defaulted to 100%'
NW    = NW + 1
CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
var2  = 100.D0
end if
else
  if(var2 .ne. 0.D0) then
    Warn1 = 'No galvanic protection depth was specified, so ' //
    &      'the value'
    write(realstr, '(1pd11.4)') var2
    Warn2 = 'read for the % patches protected (' // realstr //
    &      ') is ignored'
    NW    = NW + 1
    CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
    var2  = 0.D0
  end if
  if(var1 .lt. 0.D0) then
    Warn1 = 'Negative galvanic protection depth was specified.'
    Warn2 = 'So default value of zero is used.'
    NW    = NW + 1
    CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
    var1  = 0.D0
  end if
end if
Feat(2) = var1 * Penet1 / 100.D0
Feat(3) = var2 / 100.D0
c
c Spalling depth is read in centimeters and converted to micrometers.
c
  read(inpfid, *) spallamnt
  write(outfid, 8020) spallamnt
8020 format(f10.2,T35,'|Spalling depth (cm)')
  var1 = spallamnt
  if(var1 .lt. 0.D0) then
    Warn1 = 'Negative spalling depth was specified. So the ' //
    &      'default value of zero'
    Warn2 = 'is used (ie, no spalling).'
    NW    = NW + 1
    CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
    var1  = 0.D0
  end if
  Feat(4) = 10000.D0 * var1
c
c Read multipliers for outer and inner barrier corrosion rates.
c
  read(inpfid, *) Feat(5), Feat(6)
  write(outfid, 8030) Feat(5), Feat(6)
8030 format(2f10.3,T35,'|Scale factors for outer, inner barriers')

```

```

var1 = Feat(5)
if(var1 .lt. 0.D0) then
  Warn1 = 'Multiple of outer barrier corrosion rate is ' //
&      'defaulted to 1.'
  Warn2 = ' '
  NW     = NW + 1
  CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
  var1 = 1.D0
end if
Feat(5) = var1
C
var1 = Feat(6)
if(var1 .lt. 0.D0) then
  Warn1 = 'Multiple of inner barrier corrosion rate is ' //
&      'defaulted to 1.'
  Warn2 = ' '
  NW     = NW + 1
  CALL WriteMessage(Warn1, Warn2, .false., 0, auxfid, NW)
  var1 = 1.D0
end if
Feat(6) = var1
RETURN
END  !SUBROUTINE ReadFeatures
C
C*****
C
SUBROUTINE GetCutArea(P, AreaTot, AreaLeft, ErrFound)
C
C This subroutine calculates the probability that x is between P(3) and
C P(4) for a normal distribution with mean and standard deviation
C equal to P(1) and P(2), respectively. It returns the area to the left
C of P(3) (AreaLeft) and the area enclosed by P(3), P(4) = AreaTot.
C
C Input : P
C Output: AreaTot, AreaLeft, ErrFound, Warn1
C Local : zt, AL, AR, znorm
C
C Arguments
C
C   IMPLICIT NONE
C   logical ErrFound
C   double precision P(4), AreaTot, AreaLeft
C
C Local variables
C
C   double precision  zt, AL, AR, znorm
C
C Interpret std dev equal to zero as the random variable equal to a
C constant. Beyond five standard deviations the algorithm in function

```

c znorm is not accurate so use that as a area cutoff criterion.

c

ErrFound = .false.

AreaTot = 1.D0

AreaLeft = 0.D0

AL = 0.D0

AR = 0.D0

if(P(2) .gt. 0.D0) then

c

c If the low end bound is more than 5 deviations greater than the mean
c or the high end bound is less than 5 deviations smaller than the mean
c there is insignificant area left to sample, so flag an error. The
c function znorm returns Prob(Z < zt).

c

zt = (P(3) - P(1)) / P(2)

if(zt .gt. 5.D0) then

ErrFound = .true.

RETURN

else if(zt .gt. -5.D0) then

AL = znorm(zt)

end if

c

zt = (P(4) - P(1)) / P(2)

if(zt .lt. -5.D0) then

ErrFound = .true.

RETURN

else if(zt .lt. 5.D0) then

AR = 1.D0 - znorm(zt)

end if

end if

AreaLeft = AL

AreaTot = 1.D0 - (AL + AR)

RETURN

END !SUBROUTINE GetCutArea

c

c

c

SUBROUTINE DefaultCDFs(VarName, CDFStack, Err1, Err2, ErrFound,
& CDFStackEnd, vard)

c

c Return CDF look-up table for the requested variable name stacked in
c the CDFStack array. Vard(1), vard(2) point to the beginning and
c ending indexes in the stack and CDFStackEnd = vard(2).

c Input : VarName, CDFStack, CDFStackEnd

c Output: CDFStack, Err1, Err2, ErrFound, CDFStackEnd, vard

c Local : NPairs, i, j, X, Y

c

c Arguments, and dimensioning parameters from the include file

c

```

IMPLICIT NONE
include 'parm306.inc'
character VarName*(*), Err1*74, Err2*74
logical ErrFound
integer CDFStackEnd
double precision CDFStack(MAXPAIRS, 2), vard(*)

c
c Local variables
c
integer NPairs, i, j
double precision X(100), Y(100)

c
ErrFound = .false.

if(VarName(1: 11) .eq. 'NDTmpThresh') then
  Err1 = 'ERROR. Lookup table for the CDF of temperature ' //
  & 'initiation (no drip) not '
  Err2 = 'currently available. Program Stop'
  ErrFound = .true.
  RETURN
c
else if(VarName(1: 11) .eq. 'GDTmpThresh') then
  Err1 = 'ERROR. Lookup table for the CDF of temperature ' //
  & 'initiation (neutral drip) '
  Err2 = 'not currently available. Program Stop'
  ErrFound = .true.
  RETURN
c
else if(VarName(1: 11) .eq. 'BDTmpThresh') then
  Err1 = 'ERROR. Lookup table for the CDF of temperature ' //
  & 'initiation (non-neutral '
  Err2 = 'drip) not currently available. Program Stop'
  ErrFound = .true.
  RETURN
c
else if(VarName(1: 10) .eq. 'NDHAThresh') then
  NPairs = 61
  X(1) = 0.0000D+00
  Y(1) = 0.0000D+00
  X(2) = 1.0000D-02
  Y(2) = 2.0000D+01
  X(3) = 2.0000D-02
  Y(3) = 3.4137D+01
  X(4) = 3.0000D-02
  Y(4) = 3.9310D+01
  X(5) = 4.0000D-02
  Y(5) = 4.1812D+01
  X(6) = 5.0000D-02
  Y(6) = 4.3584D+01

```

X(7)	=	6.0000D-02
Y(7)	=	4.4993D+01
X(8)	=	7.0000D-02
Y(8)	=	4.6163D+01
X(9)	=	8.0000D-02
Y(9)	=	4.7170D+01
X(10)	=	9.0000D-02
Y(10)	=	4.8058D+01
X(11)	=	1.0000D-01
Y(11)	=	4.8856D+01
X(12)	=	1.2000D-01
Y(12)	=	5.0254D+01
X(13)	=	1.4000D-01
Y(13)	=	5.1489D+01
X(14)	=	1.6000D-01
Y(14)	=	5.2678D+01
X(15)	=	1.8000D-01
Y(15)	=	5.3911D+01
X(16)	=	2.0000D-01
Y(16)	=	5.5321D+01
X(17)	=	2.2000D-01
Y(17)	=	5.7035D+01
X(18)	=	2.4000D-01
Y(18)	=	5.8718D+01
X(19)	=	2.6000D-01
Y(19)	=	6.0237D+01
X(20)	=	2.8000D-01
Y(20)	=	6.1645D+01
X(21)	=	3.0000D-01
Y(21)	=	6.3040D+01
X(22)	=	3.2000D-01
Y(22)	=	6.4455D+01
X(23)	=	3.4000D-01
Y(23)	=	6.5730D+01
X(24)	=	3.6000D-01
Y(24)	=	6.6633D+01
X(25)	=	3.8000D-01
Y(25)	=	6.7258D+01
X(26)	=	4.0000D-01
Y(26)	=	6.7730D+01
X(27)	=	4.2000D-01
Y(27)	=	6.8114D+01
X(28)	=	4.4000D-01
Y(28)	=	6.8444D+01
X(29)	=	4.6000D-01
Y(29)	=	6.8738D+01
X(30)	=	4.8000D-01
Y(30)	=	6.9008D+01
X(31)	=	5.0000D-01

Y(31) = 6.9262D+01
X(32) = 5.2000D-01
Y(32) = 6.9504D+01
X(33) = 5.4000D-01
Y(33) = 6.9739D+01
X(34) = 5.6000D-01
Y(34) = 6.9971D+01
X(35) = 5.8000D-01
Y(35) = 7.0201D+01
X(36) = 6.0000D-01
Y(36) = 7.0434D+01
X(37) = 6.2000D-01
Y(37) = 7.0671D+01
X(38) = 6.4000D-01
Y(38) = 7.0917D+01
X(39) = 6.6000D-01
Y(39) = 7.1174D+01
X(40) = 6.8000D-01
Y(40) = 7.1448D+01
X(41) = 7.0000D-01
Y(41) = 7.1741D+01
X(42) = 7.2000D-01
Y(42) = 7.2057D+01
X(43) = 7.4000D-01
Y(43) = 7.2398D+01
X(44) = 7.6000D-01
Y(44) = 7.2759D+01
X(45) = 7.8000D-01
Y(45) = 7.3132D+01
X(46) = 8.0000D-01
Y(46) = 7.3504D+01
X(47) = 8.2000D-01
Y(47) = 7.3869D+01
X(48) = 8.4000D-01
Y(48) = 7.4224D+01
X(49) = 8.6000D-01
Y(49) = 7.4572D+01
X(50) = 8.8000D-01
Y(50) = 7.4919D+01
X(51) = 9.0000D-01
Y(51) = 7.5277D+01
X(52) = 9.1000D-01
Y(52) = 7.5464D+01
X(53) = 9.2000D-01
Y(53) = 7.5657D+01
X(54) = 9.3000D-01
Y(54) = 7.5862D+01
X(55) = 9.4000D-01
Y(55) = 7.6081D+01

```

X(56) = 9.5000D-01
Y(56) = 7.6321D+01
X(57) = 9.6000D-01
Y(57) = 7.6593D+01
X(58) = 9.7000D-01
Y(58) = 7.6914D+01
X(59) = 9.8000D-01
Y(59) = 7.7322D+01
X(60) = 9.9000D-01
Y(60) = 7.7934D+01
X(61) = 1.0000D+00
Y(61) = 9.1019D+01

```

c

```

else if(VarName(1: 10) .eq. 'GDHAThresh' .or.
& VarName(1: 10) .eq. 'BDHAThresh') then
  NPairs = 61
  X(1) = 0.0000D+00
  Y(1) = 0.0000D+00
  X(2) = 1.0000D-02
  Y(2) = 2.0000D+01
  X(3) = 2.0000D-02
  Y(3) = 3.4137D+01
  X(4) = 3.0000D-02
  Y(4) = 3.9310D+01
  X(5) = 4.0000D-02
  Y(5) = 4.1812D+01
  X(6) = 5.0000D-02
  Y(6) = 4.3584D+01
  X(7) = 6.0000D-02
  Y(7) = 4.4993D+01
  X(8) = 7.0000D-02
  Y(8) = 4.6163D+01
  X(9) = 8.0000D-02
  Y(9) = 4.7170D+01
  X(10) = 9.0000D-02
  Y(10) = 4.8058D+01
  X(11) = 1.0000D-01
  Y(11) = 4.8856D+01
  X(12) = 1.2000D-01
  Y(12) = 5.0254D+01
  X(13) = 1.4000D-01
  Y(13) = 5.1489D+01
  X(14) = 1.6000D-01
  Y(14) = 5.2678D+01
  X(15) = 1.8000D-01
  Y(15) = 5.3911D+01
  X(16) = 2.0000D-01
  Y(16) = 5.5321D+01
  X(17) = 2.2000D-01

```

Y(17) - 5.7035D+01
X(18) - 2.4000D-01
Y(18) - 5.8718D+01
X(19) - 2.6000D-01
Y(19) - 6.0226D+01
X(20) - 2.8000D-01
Y(20) - 6.1437D+01
X(21) - 3.0000D-01
Y(21) - 6.2310D+01
X(22) - 3.2000D-01
Y(22) - 6.2939D+01
X(23) - 3.4000D-01
Y(23) - 6.3431D+01
X(24) - 3.6000D-01
Y(24) - 6.3844D+01
X(25) - 3.8000D-01
Y(25) - 6.4210D+01
X(26) - 4.0000D-01
Y(26) - 6.4550D+01
X(27) - 4.2000D-01
Y(27) - 6.4874D+01
X(28) - 4.4000D-01
Y(28) - 6.5191D+01
X(29) - 4.6000D-01
Y(29) - 6.5507D+01
X(30) - 4.8000D-01
Y(30) - 6.5828D+01
X(31) - 5.0000D-01
Y(31) - 6.6160D+01
X(32) - 5.2000D-01
Y(32) - 6.6502D+01
X(33) - 5.4000D-01
Y(33) - 6.6855D+01
X(34) - 5.6000D-01
Y(34) - 6.7211D+01
X(35) - 5.8000D-01
Y(35) - 6.7560D+01
X(36) - 6.0000D-01
Y(36) - 6.7892D+01
X(37) - 6.2000D-01
Y(37) - 6.8204D+01
X(38) - 6.4000D-01
Y(38) - 6.8495D+01
X(39) - 6.6000D-01
Y(39) - 6.8768D+01
X(40) - 6.8000D-01
Y(40) - 6.9025D+01
X(41) - 7.0000D-01
Y(41) - 6.9271D+01


```
X(42) = 7.2000D-01
Y(42) = 6.9509D+01
X(43) = 7.4000D-01
Y(43) = 6.9741D+01
X(44) = 7.6000D-01
Y(44) = 6.9971D+01
X(45) = 7.8000D-01
Y(45) = 7.0203D+01
X(46) = 8.0000D-01
Y(46) = 7.0438D+01
X(47) = 8.2000D-01
Y(47) = 7.0679D+01
X(48) = 8.4000D-01
Y(48) = 7.0932D+01
X(49) = 8.6000D-01
Y(49) = 7.1201D+01
X(50) = 8.8000D-01
Y(50) = 7.1495D+01
X(51) = 9.0000D-01
Y(51) = 7.1824D+01
X(52) = 9.1000D-01
Y(52) = 7.2009D+01
X(53) = 9.2000D-01
Y(53) = 7.2211D+01
X(54) = 9.3000D-01
Y(54) = 7.2436D+01
X(55) = 9.4000D-01
Y(55) = 7.2693D+01
X(56) = 9.5000D-01
Y(56) = 7.2995D+01
X(57) = 9.6000D-01
Y(57) = 7.3364D+01
X(58) = 9.7000D-01
Y(58) = 7.3847D+01
X(59) = 9.8000D-01
Y(59) = 7.4542D+01
X(60) = 9.9000D-01
Y(60) = 7.5789D+01
X(61) = 1.0000D+00
Y(61) = 9.1019D+01
```

c

```
else if(VarName(1: 10) .eq. 'NDAQThresh') then
  NPairs = 61
  X(1) = 0.0000D+00
  Y(1) = 8.0000D+01
  X(2) = 1.0000D-02
  Y(2) = 8.4495D+01
  X(3) = 2.0000D-02
  Y(3) = 8.5416D+01
```

X(4)	=	3.0000D-02
Y(4)	=	8.5944D+01
X(5)	=	4.0000D-02
Y(5)	=	8.6332D+01
X(6)	=	5.0000D-02
Y(6)	=	8.6643D+01
X(7)	=	6.0000D-02
Y(7)	=	8.6904D+01
X(8)	=	7.0000D-02
Y(8)	=	8.7131D+01
X(9)	=	8.0000D-02
Y(9)	=	8.7333D+01
X(10)	=	9.0000D-02
Y(10)	=	8.7516D+01
X(11)	=	1.0000D-01
Y(11)	=	8.7683D+01
X(12)	=	1.2000D-01
Y(12)	=	8.7981D+01
X(13)	=	1.4000D-01
Y(13)	=	8.8245D+01
X(14)	=	1.6000D-01
Y(14)	=	8.8483D+01
X(15)	=	1.8000D-01
Y(15)	=	8.8702D+01
X(16)	=	2.0000D-01
Y(16)	=	8.8907D+01
X(17)	=	2.2000D-01
Y(17)	=	8.9101D+01
X(18)	=	2.4000D-01
Y(18)	=	8.9287D+01
X(19)	=	2.6000D-01
Y(19)	=	8.9469D+01
X(20)	=	2.8000D-01
Y(20)	=	8.9646D+01
X(21)	=	3.0000D-01
Y(21)	=	8.9823D+01
X(22)	=	3.2000D-01
Y(22)	=	9.0000D+01
X(23)	=	3.4000D-01
Y(23)	=	9.0178D+01
X(24)	=	3.6000D-01
Y(24)	=	9.0359D+01
X(25)	=	3.8000D-01
Y(25)	=	9.0544D+01
X(26)	=	4.0000D-01
Y(26)	=	9.0734D+01
X(27)	=	4.2000D-01
Y(27)	=	9.0929D+01
X(28)	=	4.4000D-01

Y(28) = 9.1132D+01
X(29) = 4.6000D-01
Y(29) = 9.1344D+01
X(30) = 4.8000D-01
Y(30) = 9.1564D+01
X(31) = 5.0000D-01
Y(31) = 9.1795D+01
X(32) = 5.2000D-01
Y(32) = 9.2036D+01
X(33) = 5.4000D-01
Y(33) = 9.2286D+01
X(34) = 5.6000D-01
Y(34) = 9.2542D+01
X(35) = 5.8000D-01
Y(35) = 9.2800D+01
X(36) = 6.0000D-01
Y(36) = 9.3058D+01
X(37) = 6.2000D-01
Y(37) = 9.3310D+01
X(38) = 6.4000D-01
Y(38) = 9.3555D+01
X(39) = 6.6000D-01
Y(39) = 9.3791D+01
X(40) = 6.8000D-01
Y(40) = 9.4018D+01
X(41) = 7.0000D-01
Y(41) = 9.4237D+01
X(42) = 7.2000D-01
Y(42) = 9.4447D+01
X(43) = 7.4000D-01
Y(43) = 9.4651D+01
X(44) = 7.6000D-01
Y(44) = 9.4850D+01
X(45) = 7.8000D-01
Y(45) = 9.5044D+01
X(46) = 8.0000D-01
Y(46) = 9.5235D+01
X(47) = 8.2000D-01
Y(47) = 9.5426D+01
X(48) = 8.4000D-01
Y(48) = 9.5620D+01
X(49) = 8.6000D-01
Y(49) = 9.5820D+01
X(50) = 8.8000D-01
Y(50) = 9.6030D+01
X(51) = 9.0000D-01
Y(51) = 9.6256D+01
X(52) = 9.1000D-01
Y(52) = 9.6379D+01

```
X(53) = 9.2000D-01
Y(53) = 9.6511D+01
X(54) = 9.3000D-01
Y(54) = 9.6656D+01
X(55) = 9.4000D-01
Y(55) = 9.6820D+01
X(56) = 9.5000D-01
Y(56) = 9.7012D+01
X(57) = 9.6000D-01
Y(57) = 9.7235D+01
X(58) = 9.7000D-01
Y(58) = 9.7500D+01
X(59) = 9.8000D-01
Y(59) = 9.7840D+01
X(60) = 9.9000D-01
Y(60) = 9.8337D+01
X(61) = 1.0000D+00
Y(61) = 1.0000D+02
```

c

```
else if (VarName(1: 10) .eq. 'GDAQThresh' .or.
& VarName(1: 10) .eq. 'BDAQThresh') then
```

```
  NPairs = 61
  X(1) = 0.0000D+00
  Y(1) = 7.0000D+01
  X(2) = 1.0000D-02
  Y(2) = 7.6490D+01
  X(3) = 2.0000D-02
  Y(3) = 8.0000D+01
  X(4) = 3.0000D-02
  Y(4) = 8.1280D+01
  X(5) = 4.0000D-02
  Y(5) = 8.1956D+01
  X(6) = 5.0000D-02
  Y(6) = 8.2498D+01
  X(7) = 6.0000D-02
  Y(7) = 8.2970D+01
  X(8) = 7.0000D-02
  Y(8) = 8.3399D+01
  X(9) = 8.0000D-02
  Y(9) = 8.3795D+01
  X(10) = 9.0000D-02
  Y(10) = 8.4167D+01
  X(11) = 1.0000D-01
  Y(11) = 8.4519D+01
  X(12) = 1.2000D-01
  Y(12) = 8.5169D+01
  X(13) = 1.4000D-01
  Y(13) = 8.5742D+01
  X(14) = 1.6000D-01
```

Y(14) = 8.6240D+01
X(15) = 1.8000D-01
Y(15) = 8.6670D+01
X(16) = 2.0000D-01
Y(16) = 8.7042D+01
X(17) = 2.2000D-01
Y(17) = 8.7369D+01
X(18) = 2.4000D-01
Y(18) = 8.7660D+01
X(19) = 2.6000D-01
Y(19) = 8.7922D+01
X(20) = 2.8000D-01
Y(20) = 8.8162D+01
X(21) = 3.0000D-01
Y(21) = 8.8384D+01
X(22) = 3.2000D-01
Y(22) = 8.8591D+01
X(23) = 3.4000D-01
Y(23) = 8.8787D+01
X(24) = 3.6000D-01
Y(24) = 8.8973D+01
X(25) = 3.8000D-01
Y(25) = 8.9152D+01
X(26) = 4.0000D-01
Y(26) = 8.9325D+01
X(27) = 4.2000D-01
Y(27) = 8.9496D+01
X(28) = 4.4000D-01
Y(28) = 8.9664D+01
X(29) = 4.6000D-01
Y(29) = 8.9831D+01
X(30) = 4.8000D-01
Y(30) = 9.0000D+01
X(31) = 5.0000D-01
Y(31) = 9.0170D+01
X(32) = 5.2000D-01
Y(32) = 9.0343D+01
X(33) = 5.4000D-01
Y(33) = 9.0520D+01
X(34) = 5.6000D-01
Y(34) = 9.0702D+01
X(35) = 5.8000D-01
Y(35) = 9.0891D+01
X(36) = 6.0000D-01
Y(36) = 9.1087D+01
X(37) = 6.2000D-01
Y(37) = 9.1294D+01
X(38) = 6.4000D-01
Y(38) = 9.1512D+01

X(39) = 6.6000D-01
Y(39) = 9.1744D+01
X(40) = 6.8000D-01
Y(40) = 9.1992D+01
X(41) = 7.0000D-01
Y(41) = 9.2258D+01
X(42) = 7.2000D-01
Y(42) = 9.2542D+01
X(43) = 7.4000D-01
Y(43) = 9.2841D+01
X(44) = 7.6000D-01
Y(44) = 9.3153D+01
X(45) = 7.8000D-01
Y(45) = 9.3472D+01
X(46) = 8.0000D-01
Y(46) = 9.3792D+01
X(47) = 8.2000D-01
Y(47) = 9.4111D+01
X(48) = 8.4000D-01
Y(48) = 9.4429D+01
X(49) = 8.6000D-01
Y(49) = 9.4749D+01
X(50) = 8.8000D-01
Y(50) = 9.5076D+01
X(51) = 9.0000D-01
Y(51) = 9.5418D+01
X(52) = 9.1000D-01
Y(52) = 9.5597D+01
X(53) = 9.2000D-01
Y(53) = 9.5785D+01
X(54) = 9.3000D-01
Y(54) = 9.5984D+01
X(55) = 9.4000D-01
Y(55) = 9.6198D+01
X(56) = 9.5000D-01
Y(56) = 9.6435D+01
X(57) = 9.6000D-01
Y(57) = 9.6703D+01
X(58) = 9.7000D-01
Y(58) = 9.7020D+01
X(59) = 9.8000D-01
Y(59) = 9.7425D+01
X(60) = 9.9000D-01
Y(60) = 9.8024D+01
X(61) = 1.0000D+00
Y(61) = 1.0000D+02

c

```
else if(VarName(1: 4) .eq. 'CAMB') then  
  Err1 = 'ERROR. Lookup table for the CDF for coefficient ' //
```

```

&      'term for pit growth law '
      Err2 = 'on the CAM not currently available.  Program Stop'
      ErrFound = .true.
      RETURN

```

c

```

else if(VarName(1: 4) .eq. 'CAMN') then
  NPairs = 61
  X(1)   = 0.0000D+00
  Y(1)   = 0.0000D+00
  X(2)   = 1.0000D-02
  Y(2)   = 2.1731D-01
  X(3)   = 2.0000D-02
  Y(3)   = 2.9841D-01
  X(4)   = 3.0000D-02
  Y(4)   = 3.5198D-01
  X(5)   = 4.0000D-02
  Y(5)   = 3.7206D-01
  X(6)   = 5.0000D-02
  Y(6)   = 3.8487D-01
  X(7)   = 6.0000D-02
  Y(7)   = 3.9541D-01
  X(8)   = 7.0000D-02
  Y(8)   = 4.0468D-01
  X(9)   = 8.0000D-02
  Y(9)   = 4.1296D-01
  X(10)  = 9.0000D-02
  Y(10)  = 4.2037D-01
  X(11)  = 1.0000D-01
  Y(11)  = 4.2781D-01
  X(12)  = 1.2000D-01
  Y(12)  = 4.4201D-01
  X(13)  = 1.4000D-01
  Y(13)  = 4.5550D-01
  X(14)  = 1.6000D-01
  Y(14)  = 4.6868D-01
  X(15)  = 1.8000D-01
  Y(15)  = 4.8179D-01
  X(16)  = 2.0000D-01
  Y(16)  = 4.9500D-01
  X(17)  = 2.2000D-01
  Y(17)  = 5.0838D-01
  X(18)  = 2.4000D-01
  Y(18)  = 5.2199D-01
  X(19)  = 2.6000D-01
  Y(19)  = 5.3586D-01
  X(20)  = 2.8000D-01
  Y(20)  = 5.5001D-01
  X(21)  = 3.0000D-01
  Y(21)  = 5.6456D-01

```

X(22)	=	3.2000D-01
Y(22)	=	5.7951D-01
X(23)	=	3.4000D-01
Y(23)	=	5.9485D-01
X(24)	=	3.6000D-01
Y(24)	=	6.1078D-01
X(25)	=	3.8000D-01
Y(25)	=	6.2776D-01
X(26)	=	4.0000D-01
Y(26)	=	6.4574D-01
X(27)	=	4.2000D-01
Y(27)	=	6.6500D-01
X(28)	=	4.4000D-01
Y(28)	=	6.8544D-01
X(29)	=	4.6000D-01
Y(29)	=	7.0799D-01
X(30)	=	4.8000D-01
Y(30)	=	7.4734D-01
X(31)	=	5.0000D-01
Y(31)	=	9.9099D-01
X(32)	=	5.2000D-01
Y(32)	=	9.9135D-01
X(33)	=	5.4000D-01
Y(33)	=	9.9171D-01
X(34)	=	5.6000D-01
Y(34)	=	9.9207D-01
X(35)	=	5.8000D-01
Y(35)	=	9.9243D-01
X(36)	=	6.0000D-01
Y(36)	=	9.9279D-01
X(37)	=	6.2000D-01
Y(37)	=	9.9315D-01
X(38)	=	6.4000D-01
Y(38)	=	9.9351D-01
X(39)	=	6.6000D-01
Y(39)	=	9.9387D-01
X(40)	=	6.8000D-01
Y(40)	=	9.9423D-01
X(41)	=	7.0000D-01
Y(41)	=	9.9459D-01
X(42)	=	7.2000D-01
Y(42)	=	9.9495D-01
X(43)	=	7.4000D-01
Y(43)	=	9.9532D-01
X(44)	=	7.6000D-01
Y(44)	=	9.9568D-01
X(45)	=	7.8000D-01
Y(45)	=	9.9604D-01
X(46)	=	8.0000D-01


```

Y(46) = 9.9640D-01
X(47) = 8.2000D-01
Y(47) = 9.9676D-01
X(48) = 8.4000D-01
Y(48) = 9.9712D-01
X(49) = 8.6000D-01
Y(49) = 9.9748D-01
X(50) = 8.8000D-01
Y(50) = 9.9784D-01
X(51) = 9.0000D-01
Y(51) = 9.9820D-01
X(52) = 9.1000D-01
Y(52) = 9.9838D-01
X(53) = 9.2000D-01
Y(53) = 9.9856D-01
X(54) = 9.3000D-01
Y(54) = 9.9874D-01
X(55) = 9.4000D-01
Y(55) = 9.9892D-01
X(56) = 9.5000D-01
Y(56) = 9.9910D-01
X(57) = 9.6000D-01
Y(57) = 9.9928D-01
X(58) = 9.7000D-01
Y(58) = 9.9946D-01
X(59) = 9.8000D-01
Y(59) = 9.9964D-01
X(60) = 9.9000D-01
Y(60) = 9.9982D-01
X(61) = 1.0000D+00
Y(61) = 1.0000D+00

```

c

```

else if(VarName(1: 6) .eq. 'NDCRMG' .or.
&      VarName(1: 6) .eq. 'GDCRMG' .or.
&      VarName(1: 6) .eq. 'BDCRMG'      ) then
  NPairs = 61
  X(1) = 0.0000D+00
  Y(1) = 1.9500D-07
  X(2) = 1.0000D-02
  Y(2) = 1.5873D-05
  X(3) = 2.0000D-02
  Y(3) = 2.9741D-05
  X(4) = 3.0000D-02
  Y(4) = 3.1584D-05
  X(5) = 4.0000D-02
  Y(5) = 3.3246D-05
  X(6) = 5.0000D-02
  Y(6) = 3.4989D-05
  X(7) = 6.0000D-02

```

Y(7)	=	3.6811D-05
X(8)	=	7.0000D-02
Y(8)	=	3.8719D-05
X(9)	=	8.0000D-02
Y(9)	=	4.0715D-05
X(10)	=	9.0000D-02
Y(10)	=	4.2805D-05
X(11)	=	1.0000D-01
Y(11)	=	4.4990D-05
X(12)	=	1.2000D-01
Y(12)	=	4.9668D-05
X(13)	=	1.4000D-01
Y(13)	=	5.4786D-05
X(14)	=	1.6000D-01
Y(14)	=	6.0383D-05
X(15)	=	1.8000D-01
Y(15)	=	6.6503D-05
X(16)	=	2.0000D-01
Y(16)	=	7.3190D-05
X(17)	=	2.2000D-01
Y(17)	=	8.0494D-05
X(18)	=	2.4000D-01
Y(18)	=	8.8468D-05
X(19)	=	2.6000D-01
Y(19)	=	9.7181D-05
X(20)	=	2.8000D-01
Y(20)	=	1.0669D-04
X(21)	=	3.0000D-01
Y(21)	=	1.1706D-04
X(22)	=	3.2000D-01
Y(22)	=	1.2838D-04
X(23)	=	3.4000D-01
Y(23)	=	1.4073D-04
X(24)	=	3.6000D-01
Y(24)	=	1.5420D-04
X(25)	=	3.8000D-01
Y(25)	=	1.6889D-04
X(26)	=	4.0000D-01
Y(26)	=	1.8490D-04
X(27)	=	4.2000D-01
Y(27)	=	2.0237D-04
X(28)	=	4.4000D-01
Y(28)	=	2.6310D-04
X(29)	=	4.6000D-01
Y(29)	=	3.8010D-04
X(30)	=	4.8000D-01
Y(30)	=	5.1820D-04
X(31)	=	5.0000D-01
Y(31)	=	6.0469D-04

X(32) - 5.2000D-01
Y(32) - 7.0533D-04
X(33) - 5.4000D-01
Y(33) - 8.2252D-04
X(34) - 5.6000D-01
Y(34) - 9.5914D-04
X(35) - 5.8000D-01
Y(35) - 1.1188D-03
X(36) - 6.0000D-01
Y(36) - 1.3060D-03
X(37) - 6.2000D-01
Y(37) - 1.5260D-03
X(38) - 6.4000D-01
Y(38) - 1.7856D-03
X(39) - 6.6000D-01
Y(39) - 2.0931D-03
X(40) - 6.8000D-01
Y(40) - 2.4592D-03
X(41) - 7.0000D-01
Y(41) - 2.8973D-03
X(42) - 7.2000D-01
Y(42) - 3.4235D-03
X(43) - 7.4000D-01
Y(43) - 4.0584D-03
X(44) - 7.6000D-01
Y(44) - 4.8267D-03
X(45) - 7.8000D-01
Y(45) - 5.7609D-03
X(46) - 8.0000D-01
Y(46) - 6.9010D-03
X(47) - 8.2000D-01
Y(47) - 8.2993D-03
X(48) - 8.4000D-01
Y(48) - 1.0022D-02
X(49) - 8.6000D-01
Y(49) - 1.2155D-02
X(50) - 8.8000D-01
Y(50) - 1.4811D-02
X(51) - 9.0000D-01
Y(51) - 1.8136D-02
X(52) - 9.1000D-01
Y(52) - 2.0109D-02
X(53) - 9.2000D-01
Y(53) - 2.2326D-02
X(54) - 9.3000D-01
Y(54) - 2.4822D-02
X(55) - 9.4000D-01
Y(55) - 2.7640D-02
X(56) - 9.5000D-01

```
Y(56) = 3.0824D-02
X(57) = 9.6000D-01
Y(57) = 3.4428D-02
X(58) = 9.7000D-01
Y(58) = 3.8516D-02
X(59) = 9.8000D-01
Y(59) = 4.3166D-02
X(60) = 9.9000D-01
Y(60) = 4.8462D-02
X(61) = 1.0000D+00
Y(61) = 0.5084D+00
```

C

```
else if(VarName(1: 6) .eq. 'GDCRMB') then
  Err1 = 'ERROR. Lookup table for the CDF for coefficient ' //
  & 'term for pit growth law '
  Err2 = 'on the CRM not currently available. Program Stop'
  ErrFound = .true.
  RETURN
```

C

```
else if(VarName(1: 6) .eq. 'BDCRMB') then
  Err1 = 'ERROR. Lookup table for the CDF for coefficient ' //
  & 'term for pit growth law '
  Err2 = 'on the CRM not currently available. Program Stop'
  ErrFound = .true.
  RETURN
```

C

```
else if(VarName(1: 6) .eq. 'GDCRMn' .or.
& VarName(1: 6) .eq. 'BDCRMn') then
  NPairs = 61
  X(1) = 0.0000D+00
  Y(1) = 0.0000D+00
  X(2) = 1.0000D-02
  Y(2) = 9.0090D-02
  X(3) = 2.0000D-02
  Y(3) = 1.8018D-01
  X(4) = 3.0000D-02
  Y(4) = 2.5221D-01
  X(5) = 4.0000D-02
  Y(5) = 2.6429D-01
  X(6) = 5.0000D-02
  Y(6) = 2.7577D-01
  X(7) = 6.0000D-02
  Y(7) = 2.8642D-01
  X(8) = 7.0000D-02
  Y(8) = 2.9609D-01
  X(9) = 8.0000D-02
  Y(9) = 3.0266D-01
  X(10) = 9.0000D-02
  Y(10) = 3.0723D-01
```

X(11) = 1.0000D-01
Y(11) = 3.1165D-01
X(12) = 1.2000D-01
Y(12) = 3.2005D-01
X(13) = 1.4000D-01
Y(13) = 3.2770D-01
X(14) = 1.6000D-01
Y(14) = 3.3483D-01
X(15) = 1.8000D-01
Y(15) = 3.4157D-01
X(16) = 2.0000D-01
Y(16) = 3.4782D-01
X(17) = 2.2000D-01
Y(17) = 3.5369D-01
X(18) = 2.4000D-01
Y(18) = 3.5936D-01
X(19) = 2.6000D-01
Y(19) = 3.6463D-01
X(20) = 2.8000D-01
Y(20) = 3.6985D-01
X(21) = 3.0000D-01
Y(21) = 3.7474D-01
X(22) = 3.2000D-01
Y(22) = 3.7962D-01
X(23) = 3.4000D-01
Y(23) = 3.8430D-01
X(24) = 3.6000D-01
Y(24) = 3.8896D-01
X(25) = 3.8000D-01
Y(25) = 3.9353D-01
X(26) = 4.0000D-01
Y(26) = 3.9808D-01
X(27) = 4.2000D-01
Y(27) = 4.0263D-01
X(28) = 4.4000D-01
Y(28) = 4.0718D-01
X(29) = 4.6000D-01
Y(29) = 4.1178D-01
X(30) = 4.8000D-01
Y(30) = 4.1644D-01
X(31) = 5.0000D-01
Y(31) = 4.2115D-01
X(32) = 5.2000D-01
Y(32) = 4.2603D-01
X(33) = 5.4000D-01
Y(33) = 4.3097D-01
X(34) = 5.6000D-01
Y(34) = 4.3619D-01
X(35) = 5.8000D-01

Y(35) = 4.4153D-01
X(36) = 6.0000D-01
Y(36) = 4.4721D-01
X(37) = 6.2000D-01
Y(37) = 4.5317D-01
X(38) = 6.4000D-01
Y(38) = 4.5941D-01
X(39) = 6.6000D-01
Y(39) = 4.6626D-01
X(40) = 6.8000D-01
Y(40) = 4.7351D-01
X(41) = 7.0000D-01
Y(41) = 4.8127D-01
X(42) = 7.2000D-01
Y(42) = 4.8968D-01
X(43) = 7.4000D-01
Y(43) = 4.9879D-01
X(44) = 7.6000D-01
Y(44) = 5.1473D-01
X(45) = 7.8000D-01
Y(45) = 5.3389D-01
X(46) = 8.0000D-01
Y(46) = 5.5471D-01
X(47) = 8.2000D-01
Y(47) = 5.7625D-01
X(48) = 8.4000D-01
Y(48) = 5.9802D-01
X(49) = 8.6000D-01
Y(49) = 6.1985D-01
X(50) = 8.8000D-01
Y(50) = 6.4169D-01
X(51) = 9.0000D-01
Y(51) = 6.6353D-01
X(52) = 9.1000D-01
Y(52) = 6.7445D-01
X(53) = 9.2000D-01
Y(53) = 6.8537D-01
X(54) = 9.3000D-01
Y(54) = 6.9629D-01
X(55) = 9.4000D-01
Y(55) = 7.0849D-01
X(56) = 9.5000D-01
Y(56) = 7.2136D-01
X(57) = 9.6000D-01
Y(57) = 7.3423D-01
X(58) = 9.7000D-01
Y(58) = 7.4710D-01
X(59) = 9.8000D-01
Y(59) = 8.1982D-01

```
X(60) = 9.9000D-01
Y(60) = 9.0991D-01
X(61) = 1.0000D+00
Y(61) = 1.0000D+00
```

```
c
  else if(VarName(1: 6) .eq. 'GDCRMk') then
    Err1 = 'ERROR. Lookup table for the CDF for coefficient ' //
    &      'term for logarithmic pit '
    Err2 = 'growth law on the CRM not currently available.' //
    &      'Program Stop'
    ErrFound = .true.
    RETURN
c
  else if(VarName(1: 6) .eq. 'BDCRMk') then
    Err1 = 'ERROR. Lookup table for the CDF for coefficient ' //
    &      'term for logarithmic pit '
    Err2 = 'growth law on the CRM not currently available.' //
    &      'Program Stop'
    ErrFound = .true.
    RETURN
c
  else if(VarName(1: 7) .eq. 'GDCRMx0') then
    Err1 = 'ERROR. Lookup table for the CDF for constant ' //
    &      'term for logarithmic pit '
    Err2 = 'growth law on the CRM not currently available.' //
    &      'Program Stop'
    ErrFound = .true.
    RETURN
c
  else if(VarName(1: 7) .eq. 'BDCRMx0') then
    Err1 = 'ERROR. Lookup table for the CDF for constant ' //
    &      'term for logarithmic pit '
    Err2 = 'growth law on the CRM not currently available.' //
    &      'Program Stop'
    ErrFound = .true.
    RETURN
  end if
c
  j = CDFStackEnd
  if(NPairs + j .gt. MAXPAIRS) then
    Err1 = 'ERROR. Too many data pairs for all look-up ' //
    &      'tables combined.'
    Err2 = 'Program STOP'
    ErrFound = .true.
    RETURN
  end if
  do i = 1, NPairs
    CDFStack(i + j, 1) = X(i)
    CDFStack(i + j, 2) = Y(i)
```

```

      end do
      vard(1)      = CDFStackEnd + 1
      CDFStackEnd = CDFStackEnd + NPairs
      vard(2)      = CDFStackEnd
      RETURN
      END      !SUBROUTINE DefaultCDFs
C
C *****
C
C      SUBROUTINE UpCase(Text, L)
C
C      Change each lower case letter in Text to upper.  Text has length L.
C      Input : Text, L
C      Output: Text
C      Local : i, N
C
C      Arguments
C
C      implicit none
C      character Text*(*)
C      integer L
C
C      Local variables
C
C      integer i, N
C
C      do i = 1, L
C         N = ichar(Text(i: i))
C         if(N .ge. 97 .and. N .le. 122) then
C            Text(i: i) = char(N - 32)
C         end if
C      end do
C      RETURN
C      END      !SUBROUTINE UpCase
C
C *****
C
C      SUBROUTINE InitDPArray(X, N)
C
C      Set all N entries in double precision array X to 0.D0
C      Input : X, N
C      Output: X
C      Local : i
C
C      Arguments
C
C      IMPLICIT NONE
C      integer N
C      double precision X(N)

```



```

C
C Local variable
C
C     integer i
C
C     do i = 1, N
C       X(i) = 0.00
C     end do
C     RETURN
C     END      !SUBROUTINE InitDPArray
C
C *****
C
C     SUBROUTINE InsertTime(outfid, auxfid, binf, ErrFound, YDNowVal,
C       &                YDntim, YDntem, YDnr, Pack, TimeIn, NdxIn)
C
C     Insert time TimeIn into the history arrays so that it falls on a time
C     step boundary. Return the index for that time step in NdxIn.
C     Input : outfid, auxfid, binf, YDNowVal, YDntim, YDntem, YDnr, Pack,
C             TimeIn
C     Output: ErrFound, YDNowVal, YDntim, YDntem, YDnr, NdxIn
C     Local : Err1, Err2, intstr, NowTemp, i, j
C
C     Arguments
C
C     IMPLICIT NONE
C     include 'parm306.inc'
C     logical ErrFound
C     integer outfid, auxfid, YDNowVal, Pack, NdxIn
C     double precision binf, YDntim(*), YDntem(*), YDnr(*), TimeIn
C
C     Local variables
C
C     character Err1*74, Err2*74, intstring*4
C     logical AreEqual
C     integer NowTemp, i, j
C     double precision Rat
C
C     if(TimeIn .lt. binf) then
C       NowTemp = YDNowVal
C       do i = 1, NowTemp
C
C     If TimeIn is essentially on a boundary (within about 5 decimals),
C     then assume it is right on boundary.
C
C       if(AreEqual(YDntim(i), TimeIn, 1.D-5)) then
C         NdxIn = i
C         EXIT
C
C

```

```

c Otherwise, encompass TimeIn by adjacent time step boundaries and
c insert a new time step within.
c
      else if(YDntim(i) .gt. TimeIn) then
        YDNowVal = YDNowVal + 1
c
        if(YDNowVal .gt. MAXDAT1) then
          write(intstring, '(i4)') Pack
          Err1 = 'ERROR. For package ' // intstring //
            &      'inserting a time step for dripping onset '
          write(intstring, '(i4)') MAXDAT1
          Err2 = 'causes dimensions to be exceeded. Current ' //
            &      'dimension = ' // intstring
          ErrFound = .true.
          CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
          RETURN
c
        else
          do j = YDNowVal, i + 1, -1
            YDnr(j) = YDnr(j - 1)
            YDntem(j) = YDntem(j - 1)
            YDntim(j) = YDntim(j - 1)
          end do
          if(YDntim(i - 1) .gt. 0.D0) then
            Rat = log(TimeIn / YDntim(i - 1)) /
              &      log(YDntim(i) / YDntim(i - 1))
            else
              Rat = TimeIn / YDntim(i)
            end if
            YDntem(i) = YDntem(i-1) + (YDntem(i) - YDntem(i-1)) * Rat
            YDnr(i) = YDnr(i-1) + (YDnr(i) - YDnr(i-1)) * Rat
            YDntim(i) = TimeIn
            NdxIn = i
            EXIT
          end if
        end if
      end do
    end if
  RETURN
END !SUBROUTINE InsertTime
c
c .....
c
c SUBROUTINE AdjustTimeSteps(outfid, auxfid, binf, ErrFound, Pack,
  &      TpVal, HASstart, AquaTrans, N1, N2,
  &      NowVal, ntim, ntem, nr, GotTime)
c
c Adjust time steps according to environment thresholds for the package
c for the time segment from N1 to N2.

```

```

c 1. Corrosion begins after temp has dropped to TpVal. Insert a time
c step each time temperature curve crosses TpVal.
c 2. Humid-air corrosion begins after RH has increased to HASTart.
c Insert a time step each time RH curve crosses HASTart.
c 3. Aqueous corrosion begins after RH has increased to AquaTrans.
c Split the time step each time RH curve crosses AquaTrans.
c Note that the end point temperatures and RHs are averaged to get the
c representative value to use during the time step. So there should be
c no ambiguity whether (or what type of) corrosion should be modeled.
c Input : outfid, auxfid, binf, Pack, TpVal, HASTart, AquaTrans, N1,
c N2, NowVal
c Output: ErrFound, ntim, ntem, nr, dt, GotTime
c Local : (none)
c
c Arguments, and dimension sizes from the include file.
c
c IMPLICIT NONE
c INCLUDE 'parm306.inc'
c logical ErrFound, GotTime
c integer outfid, auxfid, Pack, N1, N2, NowVal
c double precision binf, TpVal, HASTart, AquaTrans
c double precision ntim(*), ntem(*), nr(*)
c
c GotTime = .true.
c ErrFound = .false.
c
c Insert time steps for temperatures that equal TpVal.
c
c CALL TreatTmpThresh(outfid, auxfid, ErrFound, TpVal, Pack, NowVal,
c & N1, N2, ntim, ntem, nr, GotTime)
c if(ErrFound .or. .not. GotTime) RETURN
c
c Insert time steps for RHs that equal HASTart.
c
c CALL TreatHATHresh(outfid, auxfid, ErrFound, HASTart, Pack,
c & NowVal, N1, N2, ntim, ntem, nr, GotTime)
c if(ErrFound .or. .not. GotTime) RETURN
c
c Insert time steps for RHs at each crossing of AquaTrans.
c
c CALL TreatRHTrans(outfid, auxfid, ErrFound, AquaTrans, Pack,
c & NowVal, N1, N2, ntim, ntem, nr)
c if(ErrFound) RETURN
c RETURN
c END !SUBROUTINE AdjustTimeSteps
c
c *****
c
c SUBROUTINE TreatTmpThresh(outfid, auxfid, ErrFound, TpVal, Pack,

```

```

&                                NowVal, N1, N2, Tm, Kel, RH, GotTime)
c
c Every time TpVal is crossed in the interval defined by N1, N2,
c insert a time step so that the crossing occurs on a step boundary.
c Input : outfId, auxfid, TpVal, Pack, NowVal, N1, N2, Tm, Kel, RH
c Output: ErrFound, NowVal, N2, Tm, Kel, RH, GotTime
c Local : Err1, Err2, intstring, PrvTmpLess, CurTmpLess, i, j, Rat
c
c Arguments
c
c     IMPLICIT NONE
c     include 'parm306.inc'
c     logical ErrFound, GotTime
c     integer outfId, auxfid, Pack, NowVal, N1, N2
c     double precision TpVal, Tm(*), Kel(*), RH(*)
c
c Local variables
c
c     character Err1*74, Err2*74, intstring*4
c     logical PrvTmpLess, CurTmpLess
c     integer i, j
c     double precision Rat
c
c If the temperature passes TpVal during a time step, then split that
c time step into two time steps, one where temperature is above and one
c where temperature is below. Add a step for each such occurrence,
c unless dimensions exceeded. If temperatures exceed TpVal throughout
c the time segment, return GotTime as false.
c
c     ErrFound = .false.
c     i = N1
c     PrvTmpLess = (Kel(i) .le. TpVal)
c     GotTime = PrvTmpLess
c
c     do while (i .lt. N2)
c         i = i + 1
c         CurTmpLess = (Kel(i) .le. TpVal)
c         if (PrvTmpLess .NEQV. CurTmpLess) then
c             NowVal = NowVal + 1
c             N2 = N2 + 1
c             GotTime = .true.
c
c         if (NowVal .gt. MAXDAT1) then
c             write(intstring, '(i4)') Pack
c             Err1 = 'ERROR. Package ' // intstring // ' requires too ' //
c             & 'many time step insertions for temp'
c             write(intstring, '(i4)') MAXDAT1
c             Err2 = 'crossing its threshold. Current dimension = ' //
c             & intstring // '. Program stop.'

```

```

ErrFound = .true.
CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
RETURN
C
else
  do j = NowVal, i + 1, -1
    RH(j) = RH(j - 1)
    Kel(j) = Kel(j - 1)
    Tm(j) = Tm(j - 1)
  end do
  Rat = (TpVal - Kel(i - 1)) / (Kel(i) - Kel(i - 1))
  if(Tm(i - 1) .gt. 0.D0) then
    Tm(i) = log(Tm(i - 1)) + log(Tm(i) / Tm(i - 1)) * Rat
    Tm(i) = exp(Tm(i))
  else
    Tm(i) = Tm(i) * Rat
  end if
  RH(i) = RH(i - 1) + (RH(i) - RH(i - 1)) * Rat
  Kel(i) = TpVal
end if
PrvTmpLess = CurTmpLess
end if
end do
RETURN
END      !SUBROUTINE TreatTmpThresh
C
C *****
C
C      SUBROUTINE TreatHAThresh(outfid, auxfid, ErrFound, HASTart, Pack,
C      &                          NowVal, N1, N2, Tm, Kel, RH, GotTime)
C
C      Every time HASTart is crossed in the interval defined by N1, N2,
C      insert a time step so that the crossing occurs on a step boundary.
C      Input : outfid, auxfid, HASTart, Pack, NowVal, N1, N2, Tm, Kel, RH
C      Output: ErrFound, NowVal, N2, Tm, Kel, RH
C      Local : Err1, Err2, intstring, PrvRHLess, CurRHLess, i, j, Rat
C
C      Arguments
C
C      IMPLICIT NONE
C      include 'parm306.inc'
C      logical ErrFound, GotTime
C      integer outfid, auxfid, Pack, NowVal, N1, N2
C      double precision HASTart, Tm(*), Kel(*), RH(*)
C
C      Local variables
C
C      character Err1*74, Err2*74, intstring*4
C      logical PrvRHLess, CurRHLess

```

```

      integer i, j
      double precision Rat
c
c If the RH passes HASTart during a time step then split that time step
c into two time steps, one where RH is above and one where RH is below.
c Add a step for each such occurrence, unless dimensions exceeded.
c unless dimensions exceeded. If RHs are less than HASTart throughout
c the time segment, return GotTime as false.
c
      ErrFound = .false.
      i = N1
      PrvRHLess = (RH(i) .lt. HASTart)
      GotTime = .not. PrvRHLess
c
      do while (i .lt. N2)
        i = i + 1
        CurRHLess = (RH(i) .lt. HASTart)
        if(PrvRHLess .NEQV. CurRHLess) then
          NowVal = NowVal + 1
          N2 = N2 + 1
          GotTime = .true.
c
          if(NowVal .gt. MAXDAT1) then
            write(intstring, '(i4)') Pack
            Err1 = 'ERROR. Package ' // intstring // ' requires too ' //
            & 'many time step insertions for RH'
            write(intstring, '(i4)') MAXDAT1
            Err2 = 'crossing humid-air threshold. Current ' //
            & 'dimension = ' // intstring // '. Program stop.'
            ErrFound = .true.
            CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
            RETURN
c
          else
            do j = NowVal, i + 1, -1
              RH(j) = RH(j - 1)
              Tm(j) = Tm(j - 1)
              Kel(j) = Kel(j - 1)
            end do
            Rat = (HASTart - RH(i - 1)) / (RH(i) - RH(i - 1))
            if(Tm(i - 1) .gt. 0.D0) then
              Tm(i) = log(Tm(i - 1)) + log(Tm(i) / Tm(i - 1)) * Rat
              Tm(i) = exp(Tm(i))
            else
              Tm(i) = Tm(i) * Rat
            end if
            Kel(i) = Kel(i - 1) + (Kel(i) - Kel(i - 1)) * Rat
            RH(i) = HASTart
          end if
        end if
      end do

```

```

        PrvRHLess = CurRHLess
      end if
    end do
  RETURN
END      !SUBROUTINE TreatHAThresh

C
C *****
C
      SUBROUTINE TreatRHTrans(outfid, auxfid, ErrFound, AquaTrans, Pack,
&                               NowVal, N1, N2, Tm, Kel, RH)
C
C   Every time AquaTrans is crossed in the interval defined by N1, N2,
C   insert a time step so that the crossing occurs on a step boundary.
C   Input : outfid, auxfid, AquaTrans, Pack, NowVal, N1, N2, Tm, Kel, RH
C   Output: ErrFound, NowVal, N1, Tm, Kel, RH
C   Local : Err1, Err2, intstring, PrvRHLess, CurRHLess, i, j, Rat
C
C   Arguments
C
      IMPLICIT NONE
      include 'parm306.inc'
      logical ErrFound
      integer outfid, auxfid, Pack, NowVal, N1, N2
      double precision AquaTrans, Tm(*), Kel(*), RH(*)
C
C   Local variables
C
      character Err1*74, Err2*74, intstring*4
      logical PrvRHLess, CurRHLess
      integer i, j
      double precision Rat
C
C   If the RH passes AquaTrans during a time step, then split that time
C   step into two time steps, one where humid-air corrosion applies and
C   one where aqueous corrosion applies. Add a step for each such
C   occurrence, unless dimensions exceeded.
C
      ErrFound = .false.
      i = N1
      PrvRHLess = (RH(i) .lt. AquaTrans)
C
      do while (i .lt. N2)
        i = i + 1
        CurRHLess = (RH(i) .lt. AquaTrans)
        if (PrvRHLess .NEQV. CurRHLess) then
          NowVal = NowVal + 1
          N2 = N2 + 1
C
          if (NowVal .gt. MAXDAT1) then

```

```

        write(intstring, '(i4)') Pack
        Err1 = 'ERROR. Package ' // intstring // ' requires too ' //
&        'many time step insertions for RH crossing'
        write(intstring, '(i4)') MAXDAT1
        Err2 = 'the humid-air/aqueous threshold. Current ' //
&        'dimension = ' // intstring
        ErrFound = .true.
        CALL WriteMessage(Err1, Err2, .true., outfid, auxfid, 0)
        RETURN
c
    else
        do j = NowVal, i + 1, -1
            RH(j) = RH(j - 1)
            Kel(j) = Kel(j - 1)
            Tm(j) = Tm(j - 1)
        end do
        Rat = (AquaTrans - RH(i - 1)) / (RH(i) - RH(i - 1))
        if(Tm(i - 1) .gt. 0.00) then
            Tm(i) = log(Tm(i - 1)) + log(Tm(i) / Tm(i - 1)) * Rat
            Tm(i) = exp(Tm(i))
        else
            Tm(i) = Tm(i) * Rat
        end if
        Kel(i) = Kel(i - 1) + (Kel(i) - Kel(i - 1)) * Rat
        RH(i) = AquaTrans
    end if
    PrvRHLess = CurRHLess
end if
end do
RETURN
END      !SUBROUTINE TreatRHTrans
c
c .....
c
c      SUBROUTINE InitIntArray(J, N)
c
c      Set all N entries in integer array J to 0.
c      Input : J, N
c      Output: J
c      Local : i
c
c      Arguments
c
c      IMPLICIT NONE
c      integer J(*), N
c
c      Local variable
c
c      integer i

```



```

c
c      do i = 1, N
c          J(i) = 0
c      end do
c      RETURN
c      END      !SUBROUTINE InitIntArray
c
c.....
c
c      DOUBLE PRECISION FUNCTION SampleValue(rv, Dist, Parm, CDFStack)
c
c      Return the value (function value) of a variable that has distribution
c      type Dist and parameters in Parm, where rv is taken from the
c      vertical axis of the CDF for the variable.
c      Input : rv, Dist, Parm, CDFStack
c      Output: (function value)
c      Local : I1, I2, i, x
c
c      Arguments, and include file containing distribution type values.
c
c      IMPLICIT NONE
c      include 'parm306.inc'
c      integer Dist
c      double precision rv, Parm(*), CDFStack(MAXPAIRS, 2)
c
c      Local variables
c
c      integer I1, I2, i
c      double precision x, InvNor
c
c      if(Dist .eq. DSTFXD) then
c          SampleValue = Parm(1)
c
c      else if(Dist .eq. DSTUNI) then
c          SampleValue = (Parm(2) - Parm(1)) * rv + Parm(1)
c
c      else if(Dist .eq. DSTNOR) then
c          x = InvNor(rv)
c          SampleValue = x * Parm(2) + Parm(1)
c
c      else if(Dist .eq. DSTBNOR) then
c
c      Bounded normal distributions require special treatment to account for
c      the valid area to be sampled. Parm(4) is assumed to be the area left
c      of the lower bound and Parm(3) is assumed to be the area between the
c      lower and upper bounds.
c
c          x = rv * Parm(3) + Parm(4)
c          x = InvNor(x)

```

```

      SampleValue = x * Parm(2) + Parm(1)
c
      else if(Dist .eq. DSTLOGU) then
        x = log(Parm(2) / Parm(1)) * rv + log(Parm(1))
        SampleValue = exp(x)
c
      else if(Dist .eq. DSTLOGN) then
        x = InvNor(rv)
        x = x * Parm(2) + Parm(1)
        SampleValue = exp(x)
c
      else if(Dist .eq. DSTCDF) then
        I1 = int(Parm(1) + 0.1D0)
        I2 = int(Parm(2) + 0.1D0)
        do i = I1, I2 - 1
          if(rv .ge. CDFStack(i, 2) .and. rv .lt. CDFStack(i+1, 2)) then
            x = (rv - CDFStack(i,2)) / (CDFStack(i+1,2) - CDFStack(i,2))
            x = CDFStack(i, 1) + (CDFStack(i+1, 1) - CDFStack(i, 1)) * x
            SampleValue = x
            exit
          endif
        end do
      end if
c
      RETURN
      END      !FUNCTION SampleValue
c
c*****
c
c      SUBROUTINE LHSSamp(Dist, Parm, N, CDFStack, X)
c
c      Generate a Latin Hypercube Sample of size N for distribution type
c      Dist having parameters in Parm. The sample is unmixed so that sampled
c      values are returned in increasing order, in the X array.
c      Input : Dist, Parm, N, CDFStack
c      Output: X
c      Local : i, R, ProbInc, Area
c
c      Arguments
c
c      IMPLICIT NONE
c      include 'parm306.inc'
c      integer Dist, N
c      double precision Parm(*), CDFStack(MAXPAIRS, 2), X(*)
c
c      Local variables (include FUNCTIONs SampleValue and ranmz)
c
c      integer i
c      double precision R, ProbInc, Area, SampleValue, ranmz

```

```

      ProbInc = 1.D0 / dble(N)
      Area    = 0.D0

      if(Dist .eq. DSTFXD) then
c
c Fixed distributions have a constant value.
c
        do i = 1, N
          X(i) = Parm(1)
        end do
c
      else
c
c All others are sampled according to LHS stratification.
c
        do i = 1, N
          R = ProbInc * ranrz() + Area
          X(i) = SampleValue(R, Dist, Parm, CDFStack)
          Area = Area + ProbInc
        end do
      end if
      RETURN
    END      !SUBROUTINE LHSSamp

c
c *****
c
c      SUBROUTINE DelayTime(CermTime, NowVal, ntim, ntem, nr, Ndx1, Ndx2,
c      &                      Ndx3, NoTime)
c
c Start corrosion time at CermTime by sliding the grid to there and
c interpolating temperature and RH.
c Input : CermTime, NowVal, ntim, ntem, nr, Ndx1, Ndx2, Ndx3
c Output: NowVal, ntim, ntem, nr, Ndx1, Ndx2, Ndx3, NoTime
c Local : Ndx, i, TimRat, Temp1, RH1
c
c Arguments
c
c      IMPLICIT NONE
c      logical NoTime
c      integer NowVal, Ndx1, Ndx2, Ndx3
c      double precision CermTime, ntim(*), ntem(*), nr(*)
c
c Local variables
c
c      integer Ndx, i
c      double precision TimRat, Temp1, RH1
c
c Bracket time step when CermTime reached, interval Ndx to Ndx + 1.

```

```

c
      NoTime = .false.
      Ndx    = NowVal
      do i = 1, NowVal
        if(ntim(i) .gt. CermTime) then
          Ndx = i - 1
          EXIT
        end if
      end do

c
c If time never reaches CermTime (Ndx = NowVal), then have insufficient
c history to model corrosion, so return with an error. If first time
c exceeds CermTime (Ndx = 0), then CermTime has no effect.
c
      if(Ndx .eq. NowVal) then
        NoTime = .true.
        RETURN
      else if(Ndx .eq. 0) then
        RETURN

c
c Interpolate temperature and RH to CermTime and slide time grid so
c that CermTime is the first time.
c
      else
        TimRat = log(CermTime/ ntim(Ndx)) / log(ntim(Ndx+1)/ ntim(Ndx))
        Temp1  = (ntem(Ndx + 1) - ntem(Ndx)) * TimRat + ntem(Ndx)
        RH1    = (nr(Ndx + 1) - nr(Ndx)) * TimRat + nr(Ndx)
        Ndx    = Ndx - 1
        do i = 2, NowVal - Ndx
          ntim(i) = ntim(Ndx + i)
          ntem(i) = ntem(Ndx + i)
          nr(i)   = nr(Ndx + i)
        end do
        NowVal = NowVal - Ndx
        Ndx1   = Ndx1 - Ndx
        Ndx2   = Ndx2 - Ndx
        Ndx3   = Ndx3 - Ndx
        ntim(1) = CermTime
        nr(1)   = RH1
        ntem(1) = Temp1
      end if
      RETURN
END      !SUBROUTINE DelayTime

c
c .....
c
c
c SUBROUTINE GetGPDepth(GPAmnt, FrcProtec, GPDepth)
c

```

```

c  Assigns GPDepth a value equal to GPAmnt with probability equal to
c  FrcProtec, otherwise GPDepth is set to zero.
c
c  Input : GPAmnt, FrcProtec
c  Output: GPDepth
c  Local : u, ranmz
c
c  Arguments
c
      IMPLICIT NONE
      double precision GPAmnt, FrcProtec, GPDepth
c
c  Local variables
c
      double precision u, ranmz
c
      if(FrcProtec .gt. 0.D0) then
        u = ranmz()
        if(u .le. FrcProtec) then
          GPDepth = GPAmnt
        else
          GPDepth = 0.D0
        end if
      else
        GPDepth = 0.D0
      end if
      RETURN
      END !SUBROUTINE GetGPDepth
c
c *****
c
      SUBROUTINE OuterPitMult(NumPit, Penet1, GPDepth, Start, SegStart,
&                               SegEnd, NowVal, ntim, CAMPit, GPNdx,
&                               GPTime, ActGPT, PatchD, SegNo, PDTrans,
&                               K4PIT, DTSave, CAM1st, Cnt1st, rcam, ctim,
&                               invpwr, FirstCAMSeg)
c
c  Apply pit multipliers to general corrosion depth to find pit depth.
c  Determine when outer barrier is penetrated for each pit.
c  Input : NumPit, Penet1, GPDepth, Start, SegStart, SegEnd, NowVal,
c          ntim, CAMPit, GPNdx, GPTime, ActGPT, PatchD, SegNo, PDTrans,
c          K4PIT, DTSave, CAM1st, Cnt1st, rcam, ctim, invpwr,
c          FirstCAMSeg
c  Output: K4PIT, DTSave, CAM1st, Cnt1st
c  Local : Pit, Count, N1, N2, M, thrdep, DTime
c
c  Arguments
c
      IMPLICIT NONE

```

```

    INCLUDE 'parm306.inc'
    integer NumPit, NowVal, SegStart(*), SegEnd(*), GPNdx(*), SegNo,
    &      Cntlst, FirstCAMSeg, K4Pit(*)
    double precision Penetl, GPDepth(*), Start, GPTime(*), ActGPT(*),
    &      PDTrans(*), CAM1st
    double precision ntim(*), CAMPit(MAXPITCAM, MAXSEG, 2), PatchD(*),
    &      DTSave(*)
    double precision rcam(*), ctim(*), invpwr(*)

c
c Local variables
c
    integer Pit, Count, N1, N2
    double precision M, thrdep, DTime

c
c For the first time segment change pit multipliers to critical pit
c depths. If are past the first time segment, modify pit multipliers
c to account for the depth already penetrated at transition time.
c
    if(SegNo .eq. FirstCAMSeg) then
        do Pit = 1, NumPit
            CAMPit(Pit, SegNo, 1) = Penetl / CAMPit(Pit, SegNo, 1)
        end do

c
    else

c
c Note that if we are here, no pit has penetrated the CAM. If the new
c multiplier is less than the previous 'net' multiplier, then either
c set the new critical depth or indicate that this pit cannot break
c thru during this segment. If the new multiplier exceeds the previous
c 'net' multiplier, then either the pit is instantly thru or set the
c new critical depth.
c
        do Pit = 1, NumPit
            M = CAMPit(Pit, SegNo, 1)
            if(M .lt. DTSave(Pit) / PDTrans(SegNo - 1)) then
                if(DTSave(Pit) / M .le. PDTrans(SegNo)) then
                    CAMPit(Pit, SegNo, 1) = Penetl / M
                else
                    K4Pit(Pit) = 0
                end if
            else
                M = Penetl / M
                if(M .le. PDTrans(SegNo - 1)) then
                    Cntlst = SegStart(SegNo) + 1
                    CAM1st = ntim(Cntlst - 1)
                    RETURN
                else
                    CAMPit(Pit, SegNo, 1) = M
                end if
            end if
        end do
    end if

```

```

        end if
      end do
    end if

c
c   For each pit, if K4Pit(Pit) > NowVal, then pit has not penetrated the
c   outer barrier and DTSave(Pit) contains the depth pit has penetrated
c   at the end of the previous time segment. If pit does not penetrate
c   the outer barrier within the time frame defined by indicies N1 to N2,
c   then return the pit depth at time index N2 in DTSave. CAM1st is the
c   earliest time corrosion can start in the inner barrier (same as the
c   earliest outer barrier penetration time if no galvanic protection).
c   Cnt1st is the time interval containing CAM1st.
c
      N1 = SegStart(SegNo)
      N2 = SegEnd(SegNo)
      do Pit = 1, NumPit
        if(K4Pit(Pit) .gt. NowVal) then
          thrdep = CAMPit(Pit, SegNo, 1)
          Count = N1 + 1
          do while (PatchD(Count) .lt. thrdep .and. Count .le. N2)
            Count = Count + 1
          end do
        c
        c   If Count le N2, then pit penetrated. It can start corroding the
        c   inner barrier either when galvanic protection ends or the point in
        c   time at which it penetrated. Note, if the patch does not corrode to
        c   galvanic protection depth in this time segment (or if galvanic
        c   protection not being treated), then GPDepth is zero.
        c
          if(Count .le. N2) then
            if(thrdep .le. GPDepth(SegNo)) then
              DTSave(Pit) = GPTime(SegNo)
              K4PIT(Pit) = GPNdx(SegNo)
              CAM1st = ActGPT(SegNo)
              Cnt1st = GPNdx(SegNo)
            c
            else
              Dtime = (thrdep / rcam(Count)) ** invpwr(Count)
              Dtime = ctim(Count) - Dtime
              DTSave(Pit) = Dtime
              K4PIT(Pit) = Count
              if(ntim(Count) - Dtime .lt. CAM1st) then
                CAM1st = ntim(Count) - Dtime
                Cnt1st = Count
              end if
            end if
          else
            c
            c
          end if
        c
      end do

```

```

c Pit didn't penetrate during this time segment so save the penetration
c depth for the pit at the end of the segment in DTSave.
c
      DTSave(Pit) = PatchD(N2) * Penet1 / thrdep
    end if
  else
    K4Pit(Pit) = NowVal + 1
  end if
end do
RETURN
END      !SUBROUTINE OuterPitMult

c
c .....
c
c      SUBROUTINE OuterPitLaw(NumPit, Penet1, NowVal, Start, ntim,
c      &                      SegStart, SegEnd, CAMPit, GPNdx, GPTIME,
c      &                      ActGPT, PatchD, SegNo, PDTrans, K4PIT,
c      &                      DTSave, CAM1st, Cnt1st)
c
c Apply pit growth law to general corrosion depth to find pit depth.
c Determine when outer barrier is penetrated for each pit.
c Input : NumPit, Penet1, NowVal, Start, ntim, SegStart, SegEnd,
c        CAMPit, GPNdx, GPTIME, ActGPT, PatchD, SegNo, PDTrans, K4PIT,
c        DTSave, CAM1st, Cnt1st
c Output: K4PIT, DTSave, CAM1st, Cnt1st
c Local : Pit, i, B, n, ta, tc, PitD, thrutime, PDPprev, DRat, TS
c
c Arguments
c
c      IMPLICIT NONE
c      INCLUDE 'parm306.inc'
c      integer NumPit, NowVal, SegStart(*), SegEnd(*), GPNdx(*), SegNo,
c      &          Cnt1st, K4Pit(*)
c      double precision Penet1, Start, ActGPT(*), GPTIME(*), PDTrans(*),
c      &          CAM1st
c      double precision ntim(*), CAMPit(MAXPITCAM, MAXSEG, 2), PatchD(*),
c      &          DTSave(*)
c
c Local variables
c
c      integer Pit, i
c      double precision B, n, ta, tc, PitD, thrutime, PDPprev, DRat
c      double precision TS(MAXPITCAM)
c
c If are past the first time segment, find the time adjustment for
c each pit that accounts for the time that would have been required to
c reach current incremental depth using the new pit law properties.
c
c      CALL InitDPArray(TS, NumPit)

```



```

      if(SegNo .gt. 1) then
        do Pit = 1, NumPit
          B = CAMPit(Pit, SegNo, 1)
          n = CAMPit(Pit, SegNo, 2)
          tc = ((DTSave(Pit) - PDTrans(SegNo - 1)) / B)**(1.D0 / n)
          TS(Pit) = ntim(SegStart(SegNo)) - tc
        end do
      end if

c
c   For each pit determine the time step at which the pit will be ready
c   to corrode the inner barrier K4Pit and the time remaining in that
c   time step DTSave. If the pit does not penetrate the outer barrier
c   within the time frame defined by indicies N1 to N2, then return
c   the pit depth at time index N2 in DTSave. CAM1st is the earliest
c   time corrosion can start in the inner barrier (same as earliest outer
c   barrier penetration time if no galvanic protection). Cnt1st is the
c   time interval containing CAM1st.
c
      do Pit = 1, NumPit
c
c   If K4Pit exceeds NowVal, then the pit has not yet penetrated the CAM.
c   Also, this means that TS contains the time adjustment to account for
c   pit penetration depth at the start of this segment.
c
        if(K4PIT(Pit) .gt. NowVal) then
          ta = TS(Pit)
          B = CAMPit(Pit, SegNo, 1)
          n = CAMPit(Pit, SegNo, 2)
          PDPprev = DTSave(Pit)

c
c   Loop over time for this segment, adjusting time in the pit growth
c   law portion.
c
          do i = SegStart(SegNo) + 1, SegEnd(SegNo)
            tc = ntim(i) - ta
            PitD = PatchD(i) + B * tc ** n
            if(PitD .ge. Penet1) then

c
c   Linearly interpolate depth vs log(time) to find penetration time.
c   Time that pit can start corroding inner barrier may be delayed if
c   galvanic protection in force.
c
              DRat = (Penet1 - PDPprev) / (PitD - PDPprev)
              thrutime = DRat * log(ntim(i)/ ntim(i-1)) + log(ntim(i-1))
              thrutime = exp(thrutime)
              thrutime = DRat * (ntim(i) - ntim(i - 1)) + ntim(i - 1)
              if(thrutime .le. ActGPT(SegNo)) then
                DTSave(Pit) = GPTtime(SegNo)
                K4PIT(Pit) = GPNdx(SegNo)
              end if
            end if
          end do
        end if
      end do

```

```

        thrutime      = ActGPT(SegNo)
      else
        DTSave(Pit) = ntim(i) - thrutime
        K4PIT(Pit) = i
      end if
      if(thrutime .lt. CAM1st) then
        CAM1st = thrutime
        Cnt1st = K4PIT(Pit)
      end if
      EXIT
    else
      PDPrev = PitD
    end if
  end do

c
c If i exceeds SegEnd(SegNo), then pit did not penetrate during this
c segment. Store latest penetration depth in DTSave.
c
      if(i .gt. SegEnd(SegNo)) then
        DTSave(Pit) = PitD
      end if
    end if
  end do
  RETURN
END      !SUBROUTINE OuterPitLaw

c
c *****
c
c      SUBROUTINE InnerGenHist(Penet2, NumSeg, SegStart, SegEnd, CRMGen,
c      &                      GenRateM, MxCRMRate, ntim, ntem, LostSeg,
c      &                      PatchD, PDTrans, PatGone, PatNdx)
c
c Develop the depth vs time history for general corrosion for this
c patch in the inner barrier.
c Input : Penet2, NumSeg, SegStart, SegEnd, CRMGen, GenRateM,
c         MxCRMRate, ntim, ntem, LostSeg
c Output: PatchD, PDTrans, PatGone, PatNdx
c Local : SegNo, N1, N2, Count, PrevD, PrevT, A, K, Rate, RM
c
c Arguments, and include file for dimensions
c
      IMPLICIT NONE
      include 'parm306.inc'
      integer NumSeg, LostSeg, PatNdx
      integer SegStart(*), SegEnd(*)
      double precision Penet2, PDTrans(*), PatGone
      double precision ntim(*), ntem(*), CRMGen(MAXSEG, 2),
      & GenRateM(MAXSEG, 2), PatchD(*), MxCRMRate(*)

```

c Local variables

c

integer SegNo, N1, N2, Count
double precision PrevD, PrevT, A, K, Rate, RM

c

CALL InitDPArray(PDTrans, NumSeg)

c

c Determine the general corrosion depth during each time step.

c

PrevD = 0.D0

PrevT = 0.D0

do SegNo = LostSeg + 1, NumSeg

 N1 = SegStart(SegNo)

 N2 = SegEnd(SegNo)

 A = CRMGen(SegNo, 1)

 K = CRMGen(SegNo, 2)

 RM = GenRateM(SegNo, 2)

c

 do Count = N1 + 1, N2

 Rate = RM * exp(A - K / ntem(Count))

 if(Rate .gt. MxCRMRate(SegNo)) then

 Rate = MxCRMRate(SegNo)

 end if

 PatchD(Count) = Rate * (ntim(Count) - PrevT) + PrevD

 if(PatchD(Count) .ge. Penet2) then

 PatGone = (Penet2 - PrevD) / (PatchD(Count) - PrevD)

 PatGone = PatGone * (ntim(Count) - PrevT) + PrevT

 PatNdx = Count

 RETURN

 end if

 PrevT = ntim(Count)

 PrevD = PatchD(Count)

 end do

 PDTrans(SegNo) = PatchD(N2)

end do

RETURN

END !SUBROUTINE InnerGenHist

c

c

c

```

SUBROUTINE InnerGenRate(Penet2, NumSeg, SegStart, SegEnd,
&                        GenRateM, MxCRMRate, CRMNoD, CRMGenR,
&                        CRMTemp, ntim, ntem, LostSeg, PatchD,
&                        PDTrans, PatGone, PatNdx)

```

c

c Develop the depth vs time history for general corrosion for this
c patch in the inner barrier.

c Input : Penet2, NumSeg, SegStart, SegEnd, GenRateM, MxCRMRate,
c CRMNoD, CRMGenR, CRMTemp, ntim, ntem, LostSeg

```
c Output: PatchD, PDTrans, PatGone, PatNdx
c Local : SegNo, N1, N2, NumCRMDists, Count, i, PrevD, PrevT, Rate,
c         RM, T1, T2, R1, R2, Slp
c
c Arguments, and include file for dimensions
c
  IMPLICIT NONE
  include 'parm306.inc'
  integer NumSeg, LostSeg, PatNdx
  integer SegStart(*), SegEnd(*), CRMNoD(*)
  double precision Penet2, PDTrans(*), PatGone
  double precision ntim(*), ntem(*),
  & GenRateM(MAXSEG, 2), PatchD(*), MxCRMRate(*)
  double precision CRMGenR(MAXSEG, MAXCRMDISTS),
  & CRMTemp(MAXSEG, MAXCRMDISTS)
c
c Local variables
c
  integer SegNo, N1, N2, NumCRMDists, Count, i
  double precision PrevD, PrevT, Rate, RM, T1, T2, R1, R2, Slp
c
  CALL InitDPArray(PDTrans, NumSeg)
c
c Determine the general corrosion depth during each time step.
c
  PrevD = 0.D0
  PrevT = 0.D0
  do SegNo = LostSeg + 1, NumSeg
    N1 = SegStart(SegNo)
    N2 = SegEnd(SegNo)
    RM = GenRateM(SegNo, 2)
    NumCRMDists = CRMNoD(SegNo)
c
    do Count = N1 + 1, N2
      if(ntem(Count) .lt. CRMTemp(SegNo, 1)) then
        T1 = CRMTemp(SegNo, 1)
        T2 = CRMTemp(SegNo, 2)
        R1 = CRMGenR(SegNo, 1)
        R2 = CRMGenR(SegNo, 2)
      else if(ntem(Count) .gt. CRMTemp(SegNo, NumCRMDists)) then
        T1 = CRMTemp(SegNo, NumCRMDists - 1)
        T2 = CRMTemp(SegNo, NumCRMDists)
        R1 = CRMGenR(SegNo, NumCRMDists - 1)
        R2 = CRMGenR(SegNo, NumCRMDists)
      else
        do i = 2, NumCRMDists
          if(ntem(Count) .lt. CRMTemp(SegNo, i)) then
            T1 = CRMTemp(SegNo, 1)
            T2 = CRMTemp(SegNo, 2)
```

```

        R1 = CRMGenR(SegNo, 1)
        R2 = CRMGenR(SegNo, 2)
        EXIT
    end if
end do
end if
S1p = log(R2 / R1) / (1.D0 / T2 - 1.D0 / T1)
Rate = (1.D0 / ntem(Count) - 1.D0 / T1) * S1p + log(R1)
Rate = RM * exp(Rate)
if(Rate .gt. MxCRMRate(SegNo)) then
    Rate = MxCRMRate(SegNo)
end if
PatchD(Count) = Rate * (ntim(Count) - PrevT) + PrevD
if(PatchD(Count) .ge. Penet2) then
    PatGone = (Penet2 - PrevD) / (PatchD(Count) - PrevD)
    PatGone = PatGone * (ntim(Count) - PrevT) + PrevT
    PatNdx = Count
    RETURN
end if
PrevT = ntim(Count)
PrevD = PatchD(Count)
end do
PDTrans(SegNo) = PatchD(N2)
end do
RETURN
END      !SUBROUTINE InnerGenRate

```

```

c *****
c
c      SUBROUTINE GetCRMRateParms(vg, LnA, K, Parm, p, c)
c
c      Get sampled values of CRM general corrosion rate parameters.
c      p = 1, 2 means package, patch.
c      c = 1, 2, 3 means condition - nodrip, neutdrip, chemdrip
c      Input : vg, Parm, p, c
c      Output: LnA, K
c
c      Arguments
c
c      integer p, c
c      double precision LnA, K, Parm(*), vg(2, 3, 4)
c
c      Double precision functions
c
c      double precision ranval, gran
c
c      ranval = gran()
c      LnA    = Parm(1) + vg(p, c, 1) * ranval + vg(p, c, 4) * gran()
c      K      = Parm(2) + vg(p, c, 2) * ranval + vg(p, c, 3) * gran()

```

```

      RETURN
      END      !SUBROUTINE GetCRMRateParms
c
c *****
c
      SUBROUTINE GetCRMPakRates(varshare, NumCRMDists, PakR)
c
c  Get a untransformed mean for general CRM corrosion rate at given
c  temperatures for the package.
c  Input : varshare, NumCRMDists
c  Output: PakR
c  Local : i, ranval
c
c  Arguments
c
c      IMPLICIT NONE
c      integer NumCRMDists
c      double precision varshare, PakR(*)
c
c  Local variables
c
c      integer i
c      double precision ranval, gran
c
c      ranval = gran() * sqrt(varshare)
c      do i = 1, NumCRMDists
c          PakR(i) = ranval
c      end do
c      RETURN
c      END      !SUBROUTINE GetCRMPakRates
c
c *****
c
      SUBROUTINE GetCRMRates(ranval, PakR, Dist, Parm, CDFStack, Rate)
c
c  Get a general CRM corrosion rate at each given temperature for patch.
c  Input : ranval, PakR, Dist, Parm, CDFStack
c  Output: Rate
c  Local : p
c
c  Arguments
c
c      IMPLICIT NONE
c      INCLUDE 'parm306.inc'
c      integer Dist
c      double precision ranval, PakR, Parm(*), CDFStack(MAXPAIRS, 2),
c      &
c          Rate
c
c  Local variables

```

```

c      double precision y, p, znorm, SampleValue
c
c      y = PakR + ranval
c      p = znorm(y)
c      Rate = SampleValue(p, Dist, Parm, CDFStack)
c      RETURN
c      END      !SUBROUTINE GetCRMRates
c
c      *****
c
c      SUBROUTINE InnerPitLaw(Penet2, NumPit, BinFact, BinTerm, Pit1st,
c      &                      BinStart, Bin, PatchType, SegStart, SegEnd,
c      &                      NowVal, ntim, CRMPit, PatchD, SegNo, K4Pit,
c      &                      PDTrans, CAM1st, TmInit, DTSave)
c
c      Add pit growth law to general corrosion depth to find pit depth.
c      Determine when inner barrier is penetrated for each pit.
c      Input : Penet2, NumPit, BinFact, BinTerm, Pit1st, BinStart, Bin,
c      PatchType, SegStart, SegEnd, NowVal, ntim, CRMPit, PatchD,
c      SegNo, K4Pit, PDTrans, CAM1st, TmInit, DTSave
c      Output: Pit1st, BinStart, Bin, K4PIT, DTSave
c      Local : Pit, i, CurBin, B, n, tc, thritime, PDPprev, DRat, PitD,
c      DPit0, DPit1, DGen0, DGen1
c
c      Arguments
c
c      IMPLICIT NONE
c      INCLUDE 'parm306.inc'
c      integer NumPit, Bin(MAXBIN, 3), PatchType, SegStart(*), SegEnd(*),
c      &          NowVal, SegNo, K4Pit(*)
c      double precision Penet2, BinFact, BinTerm, Pit1st(*), BinStart,
c      &          PDTrans(*), CAM1st, TmInit
c      double precision ntim(*), CRMPit(MAXPITCRM, MAXSEG, 3), PatchD(*),
c      &          DTSave(*)
c
c      Local variables
c
c      integer Pit, i, CurBin
c      double precision B, n, tc, thritime, PDPprev, DRat, PitD
c      double precision DPit0, DPit1, DGen0, DGen1
c
c      do Pit = 1, NumPit
c
c      If K4Pit = 0, then the pit has not yet penetrated the CRM.
c
c      if(K4PIT(Pit) .eq. 0) then
c          B      = CRMPit(Pit, SegNo, 1)
c          n      = CRMPit(Pit, SegNo, 2)

```

```

PDPprev = DTSave(Pit)
DGen0 = PatchD(SegStart(SegNo))
DPit0 = PDPprev - DGen0
c
c Loop over time for this segment, adjusting time in the pit growth
c law portion.
c
do i = SegStart(SegNo) + 1, SegEnd(SegNo)
  tc = ntim(i) - TmInit
  DPit1 = B * tc ** n
  DGen1 = PatchD(i)
  PitD = DGen1 + DPit1
  if(PitD .ge. Penet2) then
c
c Interpolate depth vs time to find penetration time.
c
    DRat = (Penet2 - PDPprev) / (PitD - PDPprev)
    if(DGen1 - DGen0 .ge. DPit1 - DPit0) then
      thritime = DRat * (ntim(i) - ntim(i - 1)) + ntim(i - 1)
      thritime = thritime + CAM1st
    else
      if(ntim(i - 1) .gt. 0.D0) then
        thritime = DRat*log(ntim(i)/ntim(i-1))+log(ntim(i-1))
        thritime = exp(thritime) + CAM1st
      else
        thritime = DRat * ntim(i) + CAM1st
      end if
    end if
    BinStart = MIN(BinStart, thritime)
    if(PatchType .eq. TOPTYPE) then
      Pit1st(4) = MIN(Pit1st(4), thritime)
    else if(PatchType .eq. BOTTYPE) then
      Pit1st(5) = MIN(Pit1st(5), thritime)
    end if
    CurBin = 1 + BinFact * log(thritime) - BinTerm
    Bin(CurBin, PatchType) = Bin(CurBin, PatchType) + 1
    K4Pit(Pit) = CurBin
    EXIT
  else
    PDPprev = PitD
    DPit0 = DPit1
    DGen0 = DGen1
  end if
end do
c
c If i exceeds SegEnd(SegNo), then pit did not penetrate during this
c segment. Store latest penetration depth in DTSave.
c
  if(i .gt. SegEnd(SegNo)) then

```



```

        DTSave(Pit) = PitD
      end if
    end if
  end do
  RETURN
END      !SUBROUTINE InnerPitLaw

C
C *****
C
      SUBROUTINE InnerLogLaw(Penet2, NumPit, BinFact, BinTerm, Pit1st,
&                               BinStart, Bin, PatchType, SegStart, SegEnd,
&                               NowVal, ntim, ntem, CRMPit, PatchD, SegNo,
&                               K4Pit, PDTrans, CAM1st, TmInit, DTSave)
C
C Add logarithmic pit growth law to general corrosion depth to find pit
C depth. Determine when inner barrier is penetrated for each pit.
C Input : Penet2, NumPit, BinFact, BinTerm, Pit1st, BinStart, Bin,
C         PatchType, SegStart, SegEnd, NowVal, ntim, ntem, CRMPit,
C         PatchD, SegNo, K4Pit, PDTrans, CAM1st, TmInit, DTSave
C Output: Pit1st, BinStart, Bin, K4PIT, DTSave
C Local : Pit, i, CurBin, PDTrans, k, x0, Q, tmp, tc, thritime, PDPprev,
C         DRat, PitD, DPit0, DPit1, DGen0, DGen1, delt, T0, K100I, HrYr
C
C Arguments
C
      IMPLICIT NONE
      INCLUDE 'parm306.inc'
      integer NumPit, Bin(MAXBIN, 3), PatchType, SegStart(*), SegEnd(*),
&           NowVal, SegNo, K4Pit(*)
      double precision Penet2, BinFact, BinTerm, Pit1st(*), BinStart,
&           PDTrans(*), CAM1st, TmInit
      double precision ntim(*), ntem(*), CRMPit(MAXPITCRM, MAXSEG, 3),
&           PatchD(*), DTSave(*)
C
C Local variables
C
      integer Pit, i, CurBin
      double precision k, x0, Q, tmp, tc, thritime, PDPprev, DRat, PitD
      double precision DPit0, DPit1, DGen0, DGen1, delt, T0, K100I, HrYr
C
      K100I = 1.00 / 373.1500
      HrYr = 8766.00
      do Pit = 1, NumPit
C
C If K4Pit = 0, then the pit has not yet penetrated the CRM.
C
        if(K4PIT(Pit) .eq. 0) then
          k = CRMPit(Pit, SegNo, 1)
          x0 = CRMPit(Pit, SegNo, 2)

```

```

Q      = CRMPit(Pit, SegNo, 3)
PDPprev = DTSave(Pit)
DGen0   = PatchD(SegStart(SegNo))
DPit0   = PDPprev - DGen0
T0      = TmInit

c
c Loop over time for this segment, adjusting time in the pit growth
c law portion to initial time.
c
      do i = SegStart(SegNo) + 1, SegEnd(SegNo)
        delt = ntim(i) - T0
        tmp = k * exp(Q * (K100I - 1.00 / ntem(i)))
        if(DPit0 .gt. x0) then
          if((DPit0 - x0) / tmp .gt. 60.00) then
            tmp = 0.00
            tc = 0.00
          else
            tc = 10.00 ** ((DPit0 - x0) / tmp) / HrYr
          end if
        else
          tc = 0.00
        end if
        if(tc + delt .gt. 0.00) then
          tc = (tc + delt) * HrYr
          DPit1 = tmp * log10(tc) + x0
          DGen1 = PatchD(i)
          PitD = DGen1 + DPit1
          if(PitD .ge. Penet2) then
c
c Interpolate depth vs time to find penetration time.
c
            DRat = (Penet2 - PDPprev) / (PitD - PDPprev)
            if(DGen1 - DGen0 .ge. DPit1 - DPit0) then
              thritime = DRat * (ntim(i) - ntim(i-1)) + ntim(i-1)
              thritime = thritime + CAM1st
            else
              if(ntim(i - 1) .gt. 0.00) then
                thritime = DRat * log(ntim(i)/ntim(i-1)) + log(ntim(i-1))
                thritime = exp(thritime) + CAM1st
              else
                thritime = DRat * ntim(i) + CAM1st
              end if
            end if
            BinStart = MIN(BinStart, thritime)
            if(PatchType .eq. TOPTYPE) then
              Pit1st(4) = MIN(Pit1st(4), thritime)
            else if(PatchType .eq. BOTTYPE) then
              Pit1st(5) = MIN(Pit1st(5), thritime)
            end if
          end if
        end if
      end do

```

```

CurBin = 1 + BinFact * log(thritime) - BinTerm
Bin(CurBin, PatchType) = Bin(CurBin, PatchType) + 1
K4Pit(Pit) = CurBin
EXIT
else
  PDPprev = PitD
  DPit0 = DPit1
  DGen0 = DGen1
  T0 = ntim(i)
end if
end if
end do

c
c If i exceeds SegEnd(SegNo), then pit did not penetrate during this
c segment. Store latest penetration depth in DTSave.
c
  if(i .gt. SegEnd(SegNo)) then
    DTSave(Pit) = PitD
  end if
end if
end do
RETURN
END      !SUBROUTINE InnerLogLaw

c
c *****
c
SUBROUTINE InnerArrLaw(Penet2, NumPit, BinFact, BinTerm, Pit1st,
  & BinStart, Bin, PatchType, SegStart, SegEnd,
  & NowVal, ntim, ntem, CRMPit, PatchD, SegNo,
  & K4Pit, PDTrans, CAM1st, TmInit, DTSave)
c
c .Add Arrhenius pit growth law to general corrosion depth to find pit
c depth. Determine when inner barrier is penetrated for each pit.
c Input : Penet2, NumPit, BinFact, BinTerm, Pit1st, BinStart, Bin,
c PatchType, SegStart, SegEnd, NowVal, ntim, ntem, CRMPit,
c PatchD, SegNo, K4Pit, PDTrans, CAM1st, TmInit, DTSave
c Output: Pit1st, BinStart, Bin, K4PIT, DTSave
c Local : Pit, i, CurBin, A, C, n, PitR, thritime, PDPprev, DRat,
c PitD, DPit0, DPit1, DGen0, DGen1, PrevPwr, CurrPwr
c
c Arguments, and dimensioning parameters from include file
c
  IMPLICIT NONE
  INCLUDE 'parm306.inc'
  integer NumPit, Bin(MAXBIN, 3), PatchType, SegStart(*), SegEnd(*),
  & NowVal, SegNo, K4Pit(*)
  double precision Penet2, BinFact, BinTerm, Pit1st(*), BinStart,
  & PDTrans(*), CAM1st, TmInit
  double precision ntim(*), ntem(*), CRMPit(MAXPITCRM, MAXSEG, 3),

```

```

&          PatchD(*), DTSave(*)
c
c Local variables
c
c      integer Pit, i, CurBin
c      double precision A, C, n, PitR, thritime, PDPprev, DRat, PitD
c      double precision DPit0, DPit1, DGen0, DGen1, PrevPwr, CurrPwr
c
c      do Pit = 1, NumPit
c
c      If K4Pit = 0, then the pit has not yet penetrated the CRM.
c
c      if(K4PIT(Pit) .eq. 0) then
c          A      = CRMPit(Pit, SegNo, 1)
c          C      = CRMPit(Pit, SegNo, 2)
c          n      = CRMPit(Pit, SegNo, 3)
c          PDPprev = DTSave(Pit)
c          DGen0  = PatchD(SegStart(SegNo))
c          DPit0  = PDPprev - DGen0
c          PrevPwr = (ntim(SegStart(SegNo)) - TmInit) ** n
c
c      Loop over time for this segment, change incremental pit depth to mm.
c
c      do i = SegStart(SegNo) + 1, SegEnd(SegNo)
c          CurrPwr = (ntim(i) - TmInit) ** n
c          PitR    = exp(A - C / ntim(i)) * (CurrPwr - PrevPwr)
c          DPit1   = 0.001D0 * PitR + DPit0
c          DGen1   = PatchD(i)
c          PitD    = DGen1 + DPit1
c          if(PitD .ge. Penet2) then
c
c      Interpolate depth vs time to find penetration time.
c
c          DRat = (Penet2 - PDPprev) / (PitD - PDPprev)
c          if(DGen1 - DGen0 .ge. DPit1 - DPit0) then
c              thritime = DRat * (ntim(i) - ntim(i - 1)) + ntim(i - 1)
c              thritime = thritime + CAM1st
c          else
c              if(ntim(i - 1) .gt. 0.D0) then
c                  thritime = DRat*log(ntim(i)/ntim(i-1))+log(ntim(i-1))
c                  thritime = exp(thritime) + CAM1st
c              else
c                  thritime = DRat * ntim(i) + CAM1st
c              end if
c          end if
c          BinStart = MIN(BinStart, thritime)
c          if(PatchType .eq. TOPTYPE) then
c              Pit1st(4) = MIN(Pit1st(4), thritime)
c          else if(PatchType .eq. BOTTYPE) then

```

```

        Pit1st(5) = MIN(Pit1st(5), thritime)
    end if
    CurBin = 1 + BinFact * log(thritime) - BinTerm
    Bin(CurBin, PatchType) = Bin(CurBin, PatchType) + 1.
    K4Pit(Pit) = CurBin
    EXIT
else
    PDPrev = PitD
    DPit0 = DPit1
    DGen0 = DGen1
    PrevPwr = CurrPwr
end if
end do

c
c If i exceeds SegEnd(SegNo), then pit did not penetrate during this
c segment. Store latest penetration depth in DTSave.
c
    if(i .gt. SegEnd(SegNo)) then
        DTSave(Pit) = PitD
    end if
end if
end do
RETURN
END      !SUBROUTINE InnerArrLaw

c
c *****
c
SUBROUTINE GetCRMThick(Penet2, binf, CRMThk, NowVal, ntim, PatchD,
&                      TmInit, PatGone)
c
c Update array that stores CRM thickness using results of current patch
c Input : Penet2, binf, CRMThk, NowVal, ntim, PatchD, TmInit, PatGone
c Output: CRMThk
c Local : j, jst, k, delt, tst, t1, t2, p1, p2, rat, p, all
c
c Arguments and size parameters
c
    implicit none
    include 'parm306.inc'
    integer NowVal
    double precision Penet2, binf, TmInit, patGone
    double precision CRMThk(*), ntim(*), PatchD(*)
c
c Local variables
c
    integer j, jst, k
    double precision delt, tst, t1, t2, p1, p2, rat, p, all
c
    jst = int(MAXCRMT * TmInit / binf) + 1

```

```

delt = binf / dble(MAXCRMT)
tst = dble(jst) * delt
all = PatGone + TmInit
do j = 2, NowVal
  t1 = ntim(j - 1) + TmInit
  t2 = ntim(j) + TmInit
  p1 = PatchD(j - 1)
  p2 = PatchD(j)
  do while(tst .le. t2 .and. jst .le. MAXCRMT)
    rat = log(tst / t1) / log(t2 / t1)
    p = (p2 - p1) * rat + p1
    CRMThk(jst) = CRMThk(jst) + p
    jst = jst + 1
    tst = tst + delt
    if(tst .ge. all). then
      do k = jst, MAXCRMT
        CRMThk(k) = CRMThk(k) + Penet2
      end do
      RETURN
    end if
  end do
end do
RETURN
END      !SUBROUTINE GetCRMThick

```

```

c
c *****
c

```

```

c      SUBROUTINE LinTimeIB(Penet2, NumPit, BinFact, BinTerm, BinStart,
c      &                      Bin, PatchType, Start, NowVal, ntim, dt,
c      &                      CRMPit, Cntlst, K4PIT, c10Tdt, c10T, DTSave,
c      &                      SumC0)

```

```

c
c User requested linear time be used to simulate pitting corrosion
c for the inner barrier. Find and bin penetration times.
c Input : Penet2, NumPit, BinFact, BinTerm, BinStart, Bin, PatchType,
c         NowVal, Start, ntim, dt, C0Pak, Cntlst, K4PIT, c10Tdt, c10T,
c         DTSave, SumC0
c Output: BinStart, Bin
c Local : Pit, j, Count, CurBin, Dtime, PitCoef, PitRate, PitD,
c         thritime, RemRat, sum, sumprev
c

```

```

c Arguments
c

```

```

c      IMPLICIT NONE
c      INCLUDE 'parm306.inc'
c      integer NumPit, Bin(MAXBIN, 3), PatchType, NowVal, Cntlst,
c      &      K4PIT(*)
c      double precision Penet2, BinFact, BinTerm, BinStart, Start
c      double precision ntim(*), dt(*), c10Tdt(*), c10T(*), DTSave(*),

```

```

&          SumC0(*)
double precision CRMPit(MAXPITCRM, MAXSEG, 3)
c
c Local variables
c
integer Pit, j, Count, CurBin
double precision Dtime, PitCoef, PitRate, PitD, thritime
double precision RemRat, sum, sumprev, ranmz
c
c Define SumC0. SumC0 will be used for an initial cut. If
c C0Pak(Pit) < SumC0(Count), where Pit starts corroding inner barrier
c at time step Count, then Pit cannot possibly penetrate the inner.
c c10Tdt helps speed the search for time step in which Pit penetrated.
c
CALL InitDPArray(SumC0, NowVal)
do Count = Cnt1st, NowVal
  do j = Count, NowVal
    SumC0(Count) = SumC0(Count) + c10Tdt(j)
  end do
  SumC0(Count) = Penet2 / SumC0(Count)
end do
c
c Loop over pits. Retrieve K4Pit(Pit), which indicates whether (when)
c the pit is ready for corrosion. If K4Pit .le. NowVal, then K4Pit is
c the time step during which corrosion can begin and DTSave(Pit) is the
c time remaining in that step for corrosion.
c
do Pit = 1, NumPit
  Count = K4PIT(Pit)
  if(Count .le. NowVal) then
    PitCoef = CRMPit(1 + int(NumPit * ranmz()), 1, 1)
    Dtime = DTSave(Pit)
c
c The next check asks if PitCoef is large enough to enable complete
c penetration of the inner barrier (remaining portion), if the pit
c is at depth PitD at the beginning of time step Count.
c
    if(PitCoef .gt. SumC0(Count)) then
      PitRate = PitCoef * c10T(Count)
      PitD = PitRate * Dtime
      if(PitD .ge. Penet2) then
c
c Penetration time is the time pit penetrated outer barrier plus the
c time required to penetrate the inner. CurBin should be
c (1 + BinFact * log(thritime / bin0)). To save a division, BinTerm
c was set to BinFact * log(bin0) in the main program, which yields the
c expression for CurBin used in two places below.
c
        thritime = Penet2 / PitRate

```

```

        thritime = (Start + ntim(Count) - Dtime) + thritime
        BinStart = MIN(BinStart, thritime)
        CurBin = 1 + BinFact * log(thritime) - BinTerm
        Bin(CurBin, PatchType) = Bin(CurBin, PatchType) + 1
        K4Pit(Pit) = NowVal + 1
    else
c
c      Remaining thickness is now Penet2-PitD. Bracket when this thickness
c      is reached and linearly interpolate to find penetration time.
c
        Count = Count + 1
        RemRat = (Penet2 - PitD) / PitCoef
        sum = 0.D0
        do while (sum .lt. RemRat .and. Count .le. NowVal)
            sum = sum + c10Tdt(Count)
            Count = Count + 1
        end do
c
c      If time not exhausted, then pit penetrated the inner barrier..
c      Calculate when it penetrated, update first pit penetration time,
c      and bin the penetration time.
c
        if(Count .le. NowVal) then
            Count = Count - 1
            sumprev = sum - c10Tdt(Count)
            RemRat = (RemRat - sumprev) / c10Tdt(Count)
            thritime = Start + ntim(Count-1) + RemRat * dt(Count)
            BinStart = MIN(BinStart, thritime)
            CurBin = 1 + BinFact * log(thritime) - BinTerm
            Bin(CurBin, PatchType) = Bin(CurBin, PatchType) + 1
        end if
    end if
end if
end do !Pit
RETURN
END      !SUBROUTINE LinTimeIB
c
c *****
c
c      SUBROUTINE NonLinIB(Penet2, NumPit, BinFact, BinTerm, BinStart,
c      &                      Bin, PwrIB, PatchType, Start, NowVal, ntim,
c      &                      dt, CRMPit, Cntlst, K4PIT, c10T, DTSave,
c      &                      SumC0)
c
c      User requested nonlinear time be used to simulate pitting corrosion
c      for the inner barrier. Find and bin penetration times.
c      Input : Penet2, NumPit, BinFact, BinTerm, BinStart, Bin, PwrIB,
c      PatchType, Start, NowVal, ntim, dt, CRMPit, Cntlst, K4PIT,

```



```

c      c10T, DTSave, SumC0
c  Output: BinStart, Bin
c  Local : Pit, Count, CurBin, j, k, Dtime, PitCoef, PitRate, PitD,
c          thritime, PwrIBinv, CurrPwr, PrevPwr
c
c  Arguments
c
c      IMPLICIT NONE
c      INCLUDE 'parm306.inc'
c      integer NumPit, Bin(MAXBIN, 3), PatchType, NowVal, Cntlst,
c      &      K4PIT(*)
c      double precision Penet2, BinFact, BinTerm, BinStart, PwrIB, Start
c      double precision ntim(*), dt(*), c10T(*), DTSave(*), SumC0(*)
c      double precision CRMPit(MAXPITCRM, MAXSEG, 3)
c
c  Local variables
c
c      integer Pit, Count, CurBin, j, k
c      double precision Dtime, PitCoef, PitRate, PitD, thritime
c      double precision PwrIBinv, CurrPwr, PrevPwr, ranmz
c
c  Define SumC0. It will be used for an initial cut. If COPak(Pit)
c  is less than SumC0(Count), where Pit starts corroding inner barrier
c  at time step Count, then Pit cannot possibly penetrate the inner.
c
c      CALL InitDPArray(SumC0, NowVal)
c      do Count = Cntlst, NowVal
c          PrevPwr = 0.D0
c          do j = Count, NowVal
c              CurrPwr = (ntim(j) - ntim(Count - 1)) ** PwrIB
c              SumC0(Count) = SumC0(Count) + c10T(j) * (CurrPwr - PrevPwr)
c              PrevPwr = CurrPwr
c          end do
c          SumC0(Count) = Penet2 / SumC0(Count)
c      end do
c
c  Loop over pits. Retrieve K4Pit(Pit), which indicates whether (when)
c  the pit is ready for corrosion. If K4Pit .le. NowVal, then K4Pit is
c  the time step during which corrosion can begin and DTSave(Pit) is the
c  time remaining in that step for corrosion.
c
c      PwrIBinv = 1.D0 / PwrIB
c      do Pit = 1, NumPit
c          Count = K4PIT(Pit)
c          if(Count .le. NowVal) then
c              PitCoef = CRMPit(1 + int(NumPit * ranmz()), 1, 1)
c              Dtime = DTSave(Pit)
c          end if
c      end do
c
c  The next check asks if PitCoef is large enough to enable complete

```

```

c penetration of the inner barrier (remaining portion), if the pit
c is at depth PitD at the beginning of time step Count.
c

```

```

      if(PitCoef .gt. SumC0(Count)) then
        PitRate = PitCoef * c10T(Count)
        CurrPwr = Dtime ** PwrIB
        PitD    = PitRate * CurrPwr
        if (PitD .ge. Penet2) then

```

```

c
c Penetration time is time the pit penetrated outer barrier plus the
c time required to penetrate the inner.
c

```

```

          thritime = (Penet2 / PitRate) ** PwrIBinv
          thritime = (Start + ntim(Count) - Dtime) + thritime
          BinStart = MIN(BinStart, thritime)
          CurBin   = 1 + BinFact * log(thritime) - BinTerm
          Bin(CurBin, PatchType) = Bin(CurBin, PatchType) + 1
        else

```

```

c
c Starting at the next time step, accumulate corrosion depth (PitD) and
c elapsed time (Dtime) until depth exceeds Penet2. When detected, find
c penetration time.
c

```

```

      do k = Count + 1, NowVal
        PrevPwr = CurrPwr
        Dtime   = Dtime + dt(k)
        CurrPwr = Dtime ** PwrIB
        PitRate = PitCoef * c10T(k)
        PitD    = PitD + PitRate * (CurrPwr - PrevPwr)

```

```

c
      if(PitD .ge. Penet2) then
        PrevPwr = (PitD - Penet2) / PitRate
        thritime = (CurrPwr - PrevPwr) ** PwrIBinv
        thritime = Start + ntim(k) - (Dtime - thritime)
        BinStart = MIN(BinStart, thritime)
        CurBin   = 1 + BinFact * log(thritime) - BinTerm
        Bin(CurBin, PatchType) = Bin(CurBin, PatchType) + 1
        EXIT
      end if
    end do
  end if
end if
end if
end do !Pit
RETURN
END      !SUBROUTINE NonLinIB

```

```

c
c *****
c

```

```

      SUBROUTINE RandMix(X, N)
c
c Randomly arrange the entries in array X, having N entries.
c Input : X, N
c Output: X
c Local : J, K, XS, ranmz
c
c Arguments
c
c      integer N
c      double precision X(*)
c
c Local variables
c
c      integer J, K
c      double precision XS, ranmz
c
c      do J = N, 2, -1
c         K = int(J * ranmz()) + 1
c         XS = X(J)
c         X(J) = X(K)
c         X(K) = XS
c      end do
c      RETURN
c      END      !SUBROUTINE RandMix
c
c*****
c
c      double precision FUNCTION ranmz()
c
c Returns a uniform random deviate between 0.0 and 1.0 (exclusive
c of endpoints). Long period (>2e28) random number generator of
c George Marsaglia.
c
c Input : intseed (when entry is to ranmzset)
c Output: ranmzset, ranmz
c Local : i, j, k, n, mzran
c
c Local variables
c
c      IMPLICIT NONE
c      integer intseed(*)
c
c      integer i, j, k, n, mzran, ranmzset
c      save    i, j, k, n
c      data    i, j, k, n / 521288629, 362436069, 16163801, 1131199299/
c
c      mzran = i - k
c      if (mzran .lt. 0) mzran = mzran + 2147483579

```

```

i      = j
j      = k
k      = mzran
n      = 69069 * n + 1013904243
mzran  = mzran + n
ranmz  = 0.5D0 + 0.2328306D-9 * dble(mzran)
RETURN

c
entry ranmzset(intseed)
i = 1 + iabs(intseed(1))
j = 1 + iabs(intseed(2))
k = 1 + iabs(intseed(3))
n = intseed(4)
ranmzset = n
RETURN
END      !FUNCTION ranmz

c
c*****
c
c      integer FUNCTION iran0()
c
c      Linear Congruential generator due to Park and Miller
c
c      Input : is (for seeding @ entry iran0set)
c      Output: iran0set, iran0
c      Local : A, M, Q, R, k, izeed
c
c      Local variables
c
c      IMPLICIT NONE
c      integer is, iran0set
c      integer A, M, Q, R
c      PARAMETER (A = 48271, M = 2147483647, Q = 44488, R = 3399)
c      integer k, izeed
c      save izeed
c      data izeed /431256789/
c
c      Compute izeed = mod(A*izeed,M) by Schrage's method
c
c      k      = izeed / Q
c      izeed = A * (izeed - k * Q) - R * k
c      if (izeed .lt. 0) izeed = izeed + M
c      iran0 = izeed
c      RETURN
c
c      Set the initial seed
c
c      entry iran0set(is)
c      izeed      = 1 + iabs(is)

```

```

iran0set = iseed
RETURN
END !FUNCTION iran0

c
c *****
c
c      double precision FUNCTION gran()
c
c      Returns a normally distributed deviate with zero mean and unit
c      variance, using ranmz() as a source of uniform deviates.
c      Based on the Box-Mueller Algorithm.
c      Input : (none)
c      Output: function value of gran
c      Local : getfac, fac, gran2, rsq, v1, v2, ranmz
c
c      IMPLICIT NONE
c      logical getfac, granset
c      double precision fac, gran2, rsq, v1, v2, ranmz
c
c      SAVE    getfac, gran2
c      DATA   getfac /.true./
c
c      IF (getfac) then
c         rsq = 2.D0
c         do while (rsq .ge. 1.D0 .OR. rsq .eq. 0.D0)
c            v1 = 2.D0 * ranmz() - 1.D0
c            v2 = 2.D0 * ranmz() - 1.D0
c            rsq = v1 * v1 + v2 * v2
c         end do
c         fac  = sqrt(-2.D0 * log(rsq) / rsq)
c         gran = v1 * fac
c         gran2 = v2 * fac
c         getfac = .false.
c      else
c         gran = gran2
c         getfac = .true.
c      end if
c      RETURN
c
c      Set the initial seed
c
c      entry granset()
c      getfac = .true.
c      granset = getfac
c      RETURN
c      END !FUNCTION gran
c
c *****
c

```

```

      double precision FUNCTION InvNor(X)
c
c   Given area X, 0 < X < 1, find the deviate from the standard normal
c   using the numerical approximation to the complementary error function
c   found in FUNCTION InVERFC. The result is bounded by +/- 8 standard
c   deviations.
c   Input : X
c   Output: function value of InvNor
c   Local : sqrt2
c
c   Argument
c
c       IMPLICIT NONE
c       double precision X
c
c   Local variable
c
c       double precision sqrt2, InVERFC
c       data sqrt2 / 1.4142135623731 /
c
c       if (X .lt. 1.0D-15) then
c           InvNor = -8.0D0
c       else if (X .lt. 0.5D0) then
c           InvNor = -sqrt2 * InVERFC(2.D0 * X)
c       else if (X .gt. 0.9999999999999999D0) then
c           InvNor = 8.0D0
c       else if (X .gt. 0.5D0) then
c           InvNor = sqrt2 * InVERFC(2.D0 * (1.D0 - X))
c       else
c           InvNor = 0.D0
c       end if
c       RETURN
c       END      !FUNCTION InvNor
c
c*****
c
c       double precision FUNCTION InVERFC(Y)
c
c   Approximate the inverse complementary error function at Y. Y must be
c   greater than 6.63967719958073D-36 and less than 1.D0.
c   Input : Y
c   Output: function value of InVERFC
c   Local : I, J, K, D, TD, W, VN, VNP1, TEMP, A, A1, A2, A3, C1, C2
c
c   Argument
c
c       IMPLICIT NONE
c       double precision Y
c

```

c Local variables

c

```
integer I, J, K
double precision D, TD, W, VN, VNP1, TEMP
double precision A(22, 3), A1(22), A2(22), A3(22), C1, C2
EQUIVALENCE (A(1, 1), A1(1))
EQUIVALENCE (A(1, 2), A2(1))
EQUIVALENCE (A(1, 3), A3(1))
```

c

```
DATA(A1(I), I = 1, 22) / 9.18725611735013D-01, 0.,
& 1.68792878000327D-02, 0., 6.60337139058300D-04, 0.,
& 3.20203849839380D-05, 0., 1.72060607522481D-06, 0.,
& 9.81965971588191D-08, 0., 5.83049613537653D-09, 0.,
& 3.56019351836136D-10, 0., 2.21968915783128D-11, 0.,
& 1.40639693109741D-12, 0., 9.02597345404862D-14, 0./
```

c

```
DATA(A2(I), I=1,22) / 1.54701109458613D+00,-3.31460331083896D-01,
& 4.33001124090060D-02,-1.06564004165532D-02, 2.90613542304156D-03,
&-8.61872838022491D-04, 2.67933751795053D-04,-8.60838893942933D-05,
& 2.83232058814598D-05,-9.48870819734494D-06, 3.22422655069385D-06,
&-1.10815778472076D-06, 3.84464770797987D-07,-1.34439275565208D-07,
& 4.73255976052393D-08,-1.67556011100019D-08, 5.96199003969093D-09,
&-2.13070503291886D-09, 7.64427040920545D-10,-2.75198005584737D-10,
& 9.93792246090789D-11,-3.59877382902119D-11/
```

c

```
DATA(A3(I), I=1,22) / 1.10642888011036D+01, 4.34299147561447D+00,
&-2.33781774969295D-02, 4.23345215362947D-03, 8.68757084192089D-06,
&-5.98261113270881D-04, 4.50490139240298D-04,-2.54858131942102D-04,
& 1.27824189261340D-04,-5.97873878043957D-05, 2.66474012012582D-05,
&-1.14381836209267D-05, 4.75393030377615D-06,-1.91759589929610D-06,
& 7.50806465594834D-07,-2.84791180387123D-07, 1.04187791696225D-07,
&-3.64567243689145D-08, 1.20129296139030D-08,-3.61030126779729D-09,
& 9.12356140081759D-10,-1.36851363400914D-10/
```

c

```
DATA C1, C2 / 2.35777520630369D-01, 1.35777520630369D+00/
```

c

```
if (Y .ge. 0.5D0) then
  J = 1
  D = 1.D0 - Y
  D = D + D
else if (Y .ge. 0.1D0) then
  J = 2
  D = 5.D0 * Y - 1.5D0
else
  J = 3
  W = SQRT(-log(Y))
  D = C1 * W - C2
endif
```

```

TD   = D + D
VNP1 = 0.D0
VN   = 0.D0
do K = 22, 2, -1
  TEMP = VN
  VN   = TD * VN - VNP1 + A(K, J)
  VNP1 = TEMP
end do
VN = D * VN - VNP1 + .5D0 * A(1, J)
if (J .eq. 1) VN = D * VN
InVERFC = VN
RETURN
END      !FUNCTION InVERFC

```

```

c
c*****
c
c      double precision FUNCTION znorm(x)
c
c  This routine computes the probability that a unit normal random
c  variable is less than x, P(Z<x).
c  From: Introduction to Probability and Statistics for Engineers
c  and Scientists by Sheldon M. Ross, 1987 p.460.
c
c  Input : x
c  Output: function value of znorm
c  Local : j, u, y, sum
c
c  Arguments
c
c      IMPLICIT NONE
c      double precision x
c
c  Local variables
c
c      integer j
c      double precision u, y, sum
c
c      u   = abs(x)
c      y   = u * u
c      sum = u
c      do j = 1, 100
c        u   = -u * y * dble(2 * j - 1) / dble(2 * j * (2 * j + 1))
c        sum = sum + u
c      end do
c      sum = sum / sqrt(2.D0 * 3.14159265358979D0)
c      if (x .lt. 0.D0) then
c        znorm = 0.5D0 - sum
c      else
c        znorm = 0.5D0 + sum

```



```

      end if
      RETURN
      END !FUNCTION znorm
c
c*****
c
c      logical FUNCTION AreEqual(a, b, eps)
c
c      If the relative difference between nonzero a and b is less than eps,
c      then return AreEqual = true.  If one of a or b is zero and if the
c      other is less than or equal eps, return AreEqual = true.
c      Input : a, b, eps
c      Output: function value of AreEqual
c      Local.: absa, absb, maxab
c
c      Arguments
c
c      IMPLICIT NONE
c      double precision a, b, eps
c
c      Local variable
c
c      double precision absa, absb, maxab
c
c      absa = abs(a)
c      absb = abs(b)
c      maxab = max(absa, absb)
c      if(maxab .le. 0.D0) then                                !Both are 0
c          AreEqual = .true.
c      else if(min(absa, absb) .gt. 0.D0) then                  !Both are nonzero
c          if(abs(a - b) / maxab .lt. eps) then
c              AreEqual = .true.
c          else
c              AreEqual = .false.
c          end if
c      else                                                     !One of each
c          if(abs(a - b) .le. eps) then
c              AreEqual = .true.
c          else
c              AreEqual = .false.
c          end if
c      endif
c      RETURN
c      END !FUNCTION AreEqual
c
c*****

```


[illegible]

Corrosion	Corrosion	Corrosion
Const., but causes aqueous	Const., but causes aqueous	Const., but causes aqueous
Const., but causes humid air	Const., but causes humid air	Const., but causes humid air
Const. General corrosion only	Const. General corrosion only	Const. General corrosion only

[illegible]

```

ICM argument always required
ICM correction model for no drifts
Distribution type for ICM
Distribution parameter(s)
Distribution parameter(s)
Distribution parameter(s)
Convergence and error values
Your general CPU correction rate
Your general CPU correction rate
Shift for thermal protection temperature
Distribution parameter(s)
Dist type for hand-off initiation
Dist type for hand-off termination
Dist type for hand-off/segment transition
Dist type for hand-off/segment transition
Spreading depth as a % of thickness
Multiple for CPU, CPU correction rates

```

Const1.hst causes aqueous	CM corrosion
Const2.hst causes humid air	CM corrosion
CM General corrosion only	

[illegible][illegible]

Const1.hot cranes aqueous CHW corrosion
Const2.hot cranes humid air CHW corrosion
Const General corrosion only (JS mode) with 3-8 in the neutral drips case)

START OF PARAMETERS	END OF PARAMETERS	DESCRIPTION
1.0	2.0	Number of alternate histories
3.0	4.0	History file
5.0	6.0	History file
7.0	8.0	History file
9.0	10.0	History file
11.0	12.0	History file
13.0	14.0	History file
15.0	16.0	History file
17.0	18.0	History file
19.0	20.0	History file
21.0	22.0	History file
23.0	24.0	History file
25.0	26.0	History file
27.0	28.0	History file
29.0	30.0	History file
31.0	32.0	History file
33.0	34.0	History file
35.0	36.0	History file
37.0	38.0	History file
39.0	40.0	History file
41.0	42.0	History file
43.0	44.0	History file
45.0	46.0	History file
47.0	48.0	History file
49.0	50.0	History file
51.0	52.0	History file
53.0	54.0	History file
55.0	56.0	History file
57.0	58.0	History file
59.0	60.0	History file
61.0	62.0	History file
63.0	64.0	History file
65.0	66.0	History file
67.0	68.0	History file
69.0	70.0	History file
71.0	72.0	History file
73.0	74.0	History file
75.0	76.0	History file
77.0	78.0	History file
79.0	80.0	History file
81.0	82.0	History file
83.0	84.0	History file
85.0	86.0	History file
87.0	88.0	History file
89.0	90.0	History file
91.0	92.0	History file
93.0	94.0	History file
95.0	96.0	History file
97.0	98.0	History file
99.0	100.0	History file

[illegible]

Const2.out

[illegible]

[illegible]

Summary:

Canot1.hat causes agaveus CMV infection Canot2.hat causes hand at CMV infection CMV Canot1 infection only (2nd model with B-0 in the chemical drugs row)

STAFF OF PARLIAMENT

[illegible]

[illegible]

Const3.out

START OF PARAMETERS	Version number of code
2.94	
Com1.hat causes aqueous CM corrosion	
Com2.hat causes humid air CM corrosion	
CM General corrosion only 1/8 model with 8-0 in the chemical drips case)	

[illegible]

Covariance and error variances	
Non-general CM correction rate	
Distribution type for σ^2	
Distribution parameter(s)	
Distribution for σ^2	
Value of σ^2	
Distribution for q	
Value of q	
Isolated if very general device to	
Blat for thermal protection to	
Distribution parameter(s)	
Blat for temperature to fault	
Distribution parameter(s)	
Blat type for humid-air initiation	
Distribution parameter(s)	
Blat type for humid-air/temperature	
Distribution parameter(s)	
Isolated project depth q , p and	
Spelling depth as a % of thickness	
Multiple for CM, CM correction	

Const4.out

Consol.hat causes spurious	CMM correction
Consol2.hat causes biased ols	CMM correction
CMM DS model Kreuter] ditto	

```
Version number of mode
Number of alternate histories
Number of times the program has been run
Change time and temp and IM std deviations
Program file name
IM std deviation
Thickness of water/inner barrier(s)
IM std deviation
IM std deviation
It is for about half, wetness above peaks
is for about half, wetness below peaks
Start and end times, and number of bins
Time output (0) location (yes)
Time output (1) location (yes)
Temperature change allowed in T (K), IM (V)
Sample defining top, 1 type cooling dipe
Sample defining bottom, 1 bottom in pools
Sample defining both, 1 bottom in pools
Distribution type
Fixed value
Distribution type
JOLIE drive control, change server
Distribution type
Distribution type
A segment always required
Type of variation model
Distribution type
Fixed value
Distribution type
A segment always required
Distribution type
Distribution type
Fixed value
Distribution type
Distribution type
Coefficient error term for variance
Has general CMH correlation rate
Distribution type
Distribution type
Fixed value
Distribution type
Fixed value
Distribution type
Distribution type
Is ballistic protect dump, batches protected
```

	trial values
3.3689E-01	is galvanic protect depth, thickness protected
0.00	spalling depth (mm)
0.00	
0.00	

Const5.inp

STATE OF PARAMETERS

(number number of code
 (number number of alternate hitlines
 (number file
 (number group and Temp a MM old deviation
 (number group and Temp a MM old deviation
 (number of outer, inner hitline (mm)
 (thickness of outer, inner hitline (mm)
 (thick to fall CMC, fall width to pads
 (thick to fall CMC, fall width to pads
 (this start time a and time (yr), and of time
 (thick time (yr) for cum. pit penetration
 (used for random number generator
 (used for random number generator (C, 1987)
 (length defining top (cm), a scaling dirps
 (length defining bottom (cm), a scaling dirps
 (distinction for dripping start time
 (distinction for dripping stop time

[illegible]

Const5.out

```

          C0001.hat causes aqueous      CHM corrosion
          C0002.het causes humid air    CHM corrosion
          CHM B3 model Chemical drip
          START OF PARAMETERS
          1.05
          !Version

```

Balling depth (cm)

[illegible]

Const16.inp

Const1.hat causes aqueous CAN corrosion
 Const2.hat causes humid a/s CAN corrosion
 CAN_23 model (D = Ben + DC'n) Neutral delay

START OF PARAMETERS

[illegible]

[illegible]**Const6.out**[illegible]

[illegible]

Const7.out

[illegible]

[illegible]

Const8.inp

Contact, but causes myofascial pain
 Contact, but causes muscle aches
 Contact with the TBSA 95 model, CPT95model95Data
 Model95

TABLE OF PARAMETERS

2.481	Number of observations	12000
2.482	Number of alternative likelihoods	1
2.483	History of parameter estimates	History of parameter and mean $\pm$ SD and deviations
2.484	Model	History of parameter and mean $\pm$ SD and deviations
2.485	Model	History of parameter and mean $\pm$ SD and deviations
2.486	Model	History of parameter and mean $\pm$ SD and deviations
2.487	Model	History of parameter and mean $\pm$ SD and deviations
2.488	Model	History of parameter and mean $\pm$ SD and deviations
2.489	Model	History of parameter and mean $\pm$ SD and deviations
2.490	Model	History of parameter and mean $\pm$ SD and deviations
2.491	Model	History of parameter and mean $\pm$ SD and deviations
2.492	Model	History of parameter and mean $\pm$ SD and deviations
2.493	Model	History of parameter and mean $\pm$ SD and deviations
2.494	Model	History of parameter and mean $\pm$ SD and deviations
2.495	Model	History of parameter and mean $\pm$ SD and deviations
2.496	Model	History of parameter and mean $\pm$ SD and deviations
2.497	Model	History of parameter and mean $\pm$ SD and deviations
2.498	Model	History of parameter and mean $\pm$ SD and deviations
2.499	Model	History of parameter and mean $\pm$ SD and deviations
2.500	Model	History of parameter and mean $\pm$ SD and deviations

Const8.out

Ca²⁺, but causes aqueous Ca²⁺ excretion
Ca²⁺, but causes bound a/c Ca²⁺ excretion
Ca²⁺ with the TGF- β system, chondrogenesis
Modip

STATE OF MAINE
COUNTY OF _____
_____, ss.: I, _____,
Notary Public in and for said County of _____,
do hereby certify that _____
is duly qualified as a Justice of the Peace in and for said County of _____,
and has taken the oaths of office and qualification required by law.
In testimony whereof, I have hereunto set my hand and the seal of said Office,
this _____ day of _____, A.D. 19_____.
Notary Public in and for the State of Maine.
My Comm. Expires _____, 19_____.[illegible]

[illegible]

Caustic, hot causes apertosis Can corrosion
Caustic, hot causes burnid air Can corrosion
Can be model Chemical drip

START OF PARAMETERS	END OF PARAMETERS
1, 0.0	1, 0.0
2, 0.0	2, 0.0
3, 0.0	3, 0.0
4, 0.0	4, 0.0
5, 0.0	5, 0.0
6, 0.0	6, 0.0
7, 0.0	7, 0.0
8, 0.0	8, 0.0
9, 0.0	9, 0.0
10, 0.0	10, 0.0
11, 0.0	11, 0.0
12, 0.0	12, 0.0
13, 0.0	13, 0.0
14, 0.0	14, 0.0
15, 0.0	15, 0.0
16, 0.0	16, 0.0
17, 0.0	17, 0.0
18, 0.0	18, 0.0
19, 0.0	19, 0.0
20, 0.0	20, 0.0
21, 0.0	21, 0.0
22, 0.0	22, 0.0
23, 0.0	23, 0.0
24, 0.0	24, 0.0
25, 0.0	25, 0.0
26, 0.0	26, 0.0
27, 0.0	27, 0.0
28, 0.0	28, 0.0
29, 0.0	29, 0.0
30, 0.0	30, 0.0
31, 0.0	31, 0.0
32, 0.0	32, 0.0
33, 0.0	33, 0.0
34, 0.0	34, 0.0
35, 0.0	35, 0.0
36, 0.0	36, 0.0
37, 0.0	37, 0.0
38, 0.0	38, 0.0
39, 0.0	39, 0.0
40, 0.0	40, 0.0
41, 0.0	41, 0.0
42, 0.0	42, 0.0
43, 0.0	43, 0.0
44, 0.0	44, 0.0
45, 0.0	45, 0.0
46, 0.0	46, 0.0
47, 0.0	47, 0.0
48, 0.0	48, 0.0
49, 0.0	49, 0.0
50, 0.0	50, 0.0
51, 0.0	51, 0.0
52, 0.0	52, 0.0
53, 0.0	53, 0.0
54, 0.0	54, 0.0
55, 0.0	55, 0.0
56, 0.0	56, 0.0
57, 0.0	57, 0.0
58, 0.0	58, 0.0
59, 0.0	59, 0.0
60, 0.0	60, 0.0
61, 0.0	61, 0.0
62, 0.0	62, 0.0
63, 0.0	63, 0.0
64, 0.0	64, 0.0
65, 0.0	65, 0.0
66, 0.0	66, 0.0
67, 0.0	67, 0.0
68, 0.0	68, 0.0
69, 0.0	69, 0.0
70, 0.0	70, 0.0
71, 0.0	71, 0.0
72, 0.0	72, 0.0
73, 0.0	73, 0.0
74, 0.0	74, 0.0
75, 0.0	75, 0.0
76, 0.0	76, 0.0
77, 0.0	77, 0.0
78, 0.0	78, 0.0
79, 0.0	79, 0.0
80, 0.0	80, 0.0
81, 0.0	81, 0.0
82, 0.0	82, 0.0
83, 0.0	83, 0.0
84, 0.0	84, 0.0
85, 0.0	85, 0.0
86, 0.0	86, 0.0
87, 0.0	87, 0.0
88, 0.0	88, 0.0
89, 0.0	89, 0.0
90, 0.0	90, 0.0
91, 0.0	91, 0.0
92, 0.0	92, 0.0
93, 0.0	93, 0.0
94, 0.0	94, 0.0
95, 0.0	95, 0.0
96, 0.0	96, 0.0
97, 0.0	97, 0.0
98, 0.0	98, 0.0
99, 0.0	99, 0.0
100, 0.0	100, 0.0

Const9.out

[illegible]

i191.inp

Control. hot gases aqueous CO₂ corrosion
Control. hot gases humid air CO₂ corrosion

Chem 91 notes: Chemical drip

STATE OF MASSACHUSETTS

[illegible]

[illegible]

Const10.inp

Const1.but causes aqueous CMH corrosion
Const2.but causes humid air CMH corrosion
CMH Archiving model Chemical Drip

STATE OF FLORIDA

[illegible]

Const10.out

[illegible]

[illegible]

stl.inp

Cell hot source against CMH correlation
Coast2 hot source should also CMH correlation
CMH Attribution model w/ CMH gen rate CMHs, Chemical Bids

```

START OF PARAMETERS
3.06
2
const1.dat
3, 0., 0.
const2.dat
3, 0., 0.
const3.dat
3, 0., 0.
10.0, 2.0,
75., 0.3
0, 120., 1000, 10
1.0, 1.05, 1200
1.04, 3.04, 1.05, 1.06
51210
0.0, 0.0
120., 30.
120., 30.
Fixed
Fixed
0
Fixed
1000000.
F, F
Fixed
0.0
[No Drip Model, CAN]
CANGeneral-PitMultiple
Fixed
3.3
[No Drip Model, CHN]
CHNGeneral-PitMultiple
Fixed
3, 1.04
30.
File
10.041
100.
File
100.041
[No Drip Features]
Fixed
100.0
Fixed
10.0
Fixed
00.0
0.0, 0.0
0.0
1.0, 1.0
[Chemical Drip Model, CAN]
CANGeneral-PitMultiple
Fixed
1.0
[Chemical Drip Model, CHN]
CHNGeneral-PitMultiple
Fixed
3, 1.04
30.
File
10.041
100.
File
100.041
Fixed
20.105
Fixed
5550.7
Fixed
0.0
[No Drip Features]
Fixed

```

```

!Version number of code
!Number of alternate histories
!History file
!# patches/group and Temp & RH std deviations
!History file
!# patches/group and Temp & RH std deviations
!Thickness of series, inner barrier (cm)
!Is thick to fall CHN, free variance to patch
!Number of patches, patches/patch, pits/patch
!Site start time & end time (yr), and 0 of bins
!Output times (yr) for simul. pit penetrations
!Seed for random number generator
!How long, RH change over a time step (C, 100)
!Angle defining topography, 1 sealing drips
!Angle defining bottomography, 1 sealing drips
!Distribution for dripping start time
!Distribution parameter(s)
!Distribution for dripping stop time
!Distribution parameter(s)
!Neutral (T/F) water initially, non water (T/F)
!Water for time range for ceramic protection
!Distribution parameter(s)
!This segment always required
!CAN corrosion model for no drips
!Distribution type for m
!Distribution parameter(s)
!This segment always required
!CHN corrosion model for drips
!Number of disto (temp), non CHN rate
!Temp appropriate for dist #1
!Distribution type for #1
!Distribution parameter (a)
!Temp appropriate for dist #2
!Distribution type for #2
!Distribution parameter (a)
!This segment always required
!Dist for thermal protection temperature
!Distribution parameter(s)
!Dist type for humid-air initiation
!Distribution parameter(s)
!Dist type for humid-air/aqueous transition
!Distribution parameter(s)
!Salvage protect depth 0, 1 patches protected
!Spelling depth as a % of thickness
!Multiples for CHN, CHN corrosion rates
!Required if any non-neutral drips can be seen
!CAN corrosion model for no drips
!Distribution type for m
!Distribution parameter(s)
!Required if any non-neutral drips can be seen
!CHN corrosion model for drips
!Number of disto (temp), non CHN rate
!Temp appropriate for dist #1
!Distribution type for #1
!Distribution parameter (a)
!Temp appropriate for dist #2
!Distribution type for #2
!Distribution parameter (a)
!Distribution type for A (RH)
!Distribution parameter(s)
!Distribution type for B (RH)
!Distribution parameter(s)
!Distribution type for a
!Distribution parameter(s)
!Required if any non-neutral drips can be seen
!Dist for thermal protection temperature
!Distribution parameter(s)
!Dist for temperature to salt CHN pitting
!Distribution parameter(s)
!Dist type for humid-air initiation
!Distribution parameter(s)
!Dist type for humid-air/aqueous transition

```

```

distribution parameters:
    iculvalve protect depth %: 1 patches protected
    0.0, 0.0
    jspelling depth as % of thickness
    0.0
    multiple for CML CML correction rate
    1.0, 1.0

```

Const11.out

[illegible]

6.3.1 Discussion of Test Data

From the listing of the test data input and output files, it is seen that version 3.06 of the WAPDEG code is used for these simulations and that the two history files const1.hst and const2.hst, presented in Section 4.0 are used to simulate three waste package degradation profiles. The simulation covers a time period of 1,000,000 years, the limiting temperature value is 100°C for every package, no galvanic protection or spalling is considered, aqueous corrosion begins at an RH greater than 90% for each package and humid-air corrosion begins at an RH greater than 80% but less than 90%. The history files are not modified by WAPDEG during the simulations.

While one can easily see that the input file has not been copied line by line into the output file, the reader should be assured that this was the intention of the program developers and will be rectified in future releases. However, the salient information is presented.

The constant history tests are accomplished by modifying the code in the main line:

```
CALL SetModelParameters (Mean, v5, StanDev, varshare)
c (KM)
v5=0.D0
StanDev=0.D0
```

to force the outer barrier corrosion model parameters to assume their mean values and by fixing the appropriate distributions and parameters (RH and T thresholds, general corrosion coefficients, etc.) in the input files used.

The hand calculations were performed in MathCad™ 7.0 Professional and are presented below:

This file is used in verifying and validating WAPDEG.

These are the outer (CAM) barrier thickness, D1, and the inner (CRM) barrier thickness, D2.

$D1 := 10^5 \text{ } \mu\text{m}$ $D2 := 20 \text{ mm}$

These are the CAM corrosion coefficients:

$c1 := 1.7184748910^1$	$c5 := 111.529542$
$c2 := -6.2346152510^2$	$c6 := 0.531987099$
$c3 := -9.7445821710^2$	$c7 := -23291.1746$
$c4 := 6.2270206110^{-1}$	$c8 := -0.000319184722$

T := 273.15 + 95 tterm := 0 n := 0.5 gp% := 0 Start := 1 fp := 1.5

$D_{th} := \frac{Dl}{fp}$ Star := max((tterm Start)) Star = 1 t := 1

$D_{gp} := \frac{gp\%}{100} \cdot Dl$ Dgp = 0 $D_{th} := \max((D_{th} \ D_{gp}))$

$Dh(RH, t) := fp \cdot \exp\left(c1 + \frac{c2}{T} + \frac{c3}{RH}\right) \cdot t^{c4}$ $Da(T, t) := fp \cdot \exp\left(c5 + \frac{c7}{T} + c8 \cdot T^2\right) \cdot t^{c6}$

CAMGeneral+PitMultiples, humid air

CAMGeneral+PitMultiples, aqueous

$tlh(RH, t) := \text{root}\left(\frac{Dh(85, t)}{1000} - 100, t\right) + \text{Star}$

$tl_a(T) := \left(\text{root}\left(\frac{Da(T, t)}{1000} - 100, t\right)\right) + \text{Star}$

tlh(85, t) = 86724.9

tl_a(T) = 96158.5

CAMGeneral+PitGrowthLaw

$dCAMa(B, t) := \frac{1}{1000} \cdot \left(\frac{Da(T, t)}{fp} + B \cdot t^n\right)$

$(\text{root}(dCAMa(300, t) - 100, t)) + \text{Star} = 38628.1$

B is in $\mu\text{m}/\text{yr}^n$

CRMGeneralOnly

$dCRMg(A, Q, T, t) := A \cdot \exp\left(-\frac{Q}{T}\right) \cdot t \quad \text{mm}$

$dCRMgen(t) := dCRMg(\exp(6.3), 4990, T, t)$

$\text{root}(dCRMgen(t) - 20, t) = 28282.3$

$dCRMgen(28282.3) = 20$

CRM penetration from general corrosion in 28,282 years.

LNLL General Corrosion Model

$dCRMgenLLNL(t) := dCRMg(\exp(8.2907), 5558.7, T, t)$

$\text{root}(dCRMgenLLNL(t) - 20, t) = 18106.8$

CRMGeneralRateOnly

$$i := 1..2 \quad T_s := \begin{bmatrix} 100 \\ 50 \end{bmatrix} \quad T_t := \frac{1}{(T_s + 273.15)}$$

$$cdf50 := \begin{bmatrix} 5 \cdot 10^{-5} & 0 \\ 5 \cdot 10^{-5} & 1 \end{bmatrix} \quad cdf100 := \begin{bmatrix} 5 \cdot 10^{-4} & 0 \\ 5 \cdot 10^{-4} & 1 \end{bmatrix} \quad rs := \begin{bmatrix} cdf100_{i,1} \\ cdf50_{i,1} \end{bmatrix} \quad r_i := \ln(rs_i)$$

$$RCRM := \text{linterp}\left(T_t, r_i, \frac{1}{T}\right) \quad RCRM = -7.8$$

CRMGeneral+PitGrowthLaw

$$d2(B, t) := dCRMgen(t) + B \cdot t^n \quad \text{mm} \quad dCRMjs(t) := d2(0.1, t)$$

$$\text{root}(dCRMjs(t) - 20, t) = 12482.8 \quad dCRMjs(12482.8) = 20$$

CRM pit penetration in 12,482.8 years

CRMGeneral+LogPitGrowth

$$dds(k100, x_0, Q, T, t) := k100 \cdot \exp\left[Q \cdot \left(\frac{1}{373.15} - \frac{1}{T}\right)\right] \cdot \log(t \cdot 8766) + x_0 + dCRMgen(t)$$

$$dCRMds(t) := dds(0.75, 1.2, 6615, T, t) \quad \text{root}(dCRMds(t) - 20, t) = 19718$$

CRM pit penetration in 19718 years

CRMExponentialPitRate

$$i2(T) := \left(\frac{D2}{\exp\left(50.373 + \frac{-19655.85}{T}\right)} \right)^{\frac{1}{n}}$$

$$i2(T) = 167244.3$$

CRM pit penetration in 167,244.3 years

CRMGeneralLLNL+ArrheniusPit

$$dda(b_0, b_1, T, t) := dCRMgenLLNL(t) + \exp\left(b_0 - \frac{b_1}{T}\right) \cdot 10^{-3} \cdot t^n$$

$$dCRMap(t) := dda(20.105, 5558.7, T, t)$$

$$\text{root}(dCRMap(t) - 20, t) = 6888.2$$

To get total failure times:

Humid CAM + CRMGeneralOnly

$$tgen := \text{root}(dCRMgen(t) - 20, t)$$

$$timeh(T, RH) := t1h(T, RH) + tgen$$

$$t1h(T, 85) = 86724.9$$

$$timeh(T, 85) = 115007.2$$

Aqueous CAM + CRMGeneralOnly

$$tgen = 28282.3$$

$$timea(T) := t1a(T) + tgen$$

$$t1a(T) = 96158.5$$

$$timea(T) = 124440.8$$

Humid CAM + CRMExponentialPitRate

$$timeh(T, RH) := t1h(T, RH) + t2(T)$$

$$t1h(T, 85) = 86724.9$$

$$timeh(T, 85) = 253969.3$$

Aqueous CAM + CRMExponentialPitRate

$$timea(T) := t1a(T) + t2(T)$$

$$t1a(T) = 96158.5$$

$$timea(T) = 263402.8$$

Aqueous CAM + CRMGeneral+PitGrowthLaw

$$tgenpitJS(T) := \text{root}(dCRMjs(t) - 20, t)$$

$$timejs(T) := t1a(T) + tgenpitJS(T)$$

$$t1a(T) = 96158.5$$

$$tgenpitJS(T) = 12482.8$$

$$timejs(T) = 108641.3$$

Aqueous CAM + CRMGeneral+LogPitGrowth

$$tgenpitDS(T) := \text{root}(dCRMds(t) - 20, t)$$

$$timeds(T) := t1a(T) + tgenpitDS(T)$$

$$t1a(T) = 96158.5$$

$$tgenpitDS(T) = 19718$$

$$timeds(T) = 115876.5$$

CAMGeneral+PitGrowthLaw + CRMGeneral+LogPitGrowth

$$tCAMPit(T) := (\text{root}(dCAMa(300, t) - 100, t)) + \text{Star}$$

$$timeds2(T) := tCAMPit(T) + tgenpitDS(T)$$

$$tCAMPit(T) = 38628.1$$

$$timeds2(T) = 58346.1$$

$$tCAMPit(T) + tgen = 66910.3$$

Aqueous CAM + CRMGeneral+ArrheniusPit

$$tgenpitAR(T) := \text{root}(dCRMp(t) - 20, t)$$

$$tgenpitAR(T) + t1a(T) = 103046.7$$

$$tgenLLNL := \text{root}(dCRMgenLLNL(t) - 20, t)$$

$$t1a(T) + tgenLLNL = 114265.3$$

$$tgenpitAR(T) = 6888.2$$

CRMGenRate+ArrheniusPit

$$\text{CRM}(b_0, b_1, t) := \exp(\text{RCRM}) \cdot t + \exp\left(b_0 - \frac{b_1}{T}\right) \cdot 10^{-3} \cdot t^n$$

$$t_{\text{CRM}} := \text{root}(\text{CRM}(20.105, 5558.7, t) - 20, t) \quad t_{\text{CRM}} = 10863.9$$

$$t_{\text{la}}(T) + t_{\text{CRM}} = 107022.4$$

$$\text{root}(\exp(\text{RCRM}) \cdot t - 20, t) = 48959.5$$

$$t_{\text{la}}(T) + \text{root}(\exp(\text{RCRM}) \cdot t - 20, t) = 145118$$

$$t_{\text{lh}}(T, 85) + \text{root}(\exp(\text{RCRM}) \cdot t - 20, t) = 135684.4$$

Before presenting the results of the constant history simulations, a brief discussion of the accuracy of WAPDEG output is warranted. While the equations that characterize the general corrosion depths can be solved in closed form, those resulting from coupling a general corrosion model to a localized corrosion model can not. As mentioned in Section 4.3 (Comment 4.3.3), WAPDEG uses a relatively fast time interpolation/solving algorithm and the accuracy of the results it produces is affected by the number of time history points used in the simulation, i.e. the accuracy of the results is a user-controlled parameter through the time, temperature, relative humidity history used as input. As the histories used in the constant history simulations is sparse (see Section 4.0), the accuracy of localized corrosion (pit) penetration times is not expected to be high. As will be seen, errors of up to 9% are obtained and considered acceptable for the purposes of these simulations.

Consideration of the input/output files presented previously shows that WAPDEG predicts dripped-upon patches in the outer barrier (CAM) will be penetrated in 96,158.5 years as will CAM patches that have undergone aqueous corrosion (i.e. were modeled using const1.hst) if the CAMGeneral+PitMultiples model is used. For the same CAM model, WAPDEG predicts humid-air CAM penetration in 86724.9 years. These values agree perfectly with the hand calculations presented above.

Consideration of const1.out shows that WAPDEG predicts that the CAMGeneral+PitMultiples model coupled with the CRMGeneralOnly model results in CRM penetration in 124,440.8 years using const1.hst (aqueous CAM corrosion) and in 115,007.2 years using const2.hst (humid-air CAM corrosion). These values are also in agreement with the hand calculations. Note that 'Inner Barrier - Drips' columns reflect the total simulation time as no patches are dripped upon.

Const1cerm.out shows the result of using a 50 year ceramic delay time on the corrosion process. All values reported are 49 years later than in const1.out as ceramic delay time is measured from year 0 whereas the corrosion simulation begins with the first time point in the history file which is year 1 in this case.

Consideration of const2.out shows that WAPDEG predicts that the CAMGeneral+PitMultiples model coupled with the CRMGeneral+PitGrowthLaw with $B = 0$ in the [Neutral Drip, . . .] input

segment also results in CRM penetration in 124,440.8 years using const1.hst (aqueous CAM corrosion) and in 115,007.2 years using const2.hst (humid-air CAM corrosion). These values are also in agreement with the hand calculations. Note that the 'Inner Barrier - Drips' columns report penetration times consistent with the aqueous CAMGeneral+PitMultiples model coupled with the CRMGeneralOnly model.

Consideration of const3.out shows that WAPDEG predicts that the CAMGeneral+PitMultiples model coupled with the CRMGeneral+PitGrowthLaw with $B = 0$ in the [Chemical Drip, . . .] input segment also results in CRM penetration in 124,440.8 years using const1.hst (aqueous CAM corrosion) and in 115,007.2 years using const2.hst (humid-air CAM corrosion). These values are also in agreement with the hand calculations.

Consideration of const4.out shows that WAPDEG predicts that the CAMGeneral+PitMultiples model coupled with the CRMGeneral+LogPitGrowth in the [Neutral Drip, . . .] input segment results in CRM penetration in 116,432.6 years. This value is also in agreement with the hand calculations which predict 115,876.5 years. The current version of WAPDEG uses a relatively unsophisticated interpolation algorithm to determine the pit penetration time which, in this case, results in a 0.5% error.

Consideration of const5.out shows that WAPDEG predicts that the CAMGeneral+PitMultiples model coupled with the CRMGeneral+LogPitGrowth in the [Chemical Drip, . . .] input segment also results in CRM penetration in 116,432.6 years. This value is also in agreement with the hand calculations which predict 115,876.5 years. Again there is a 0.5% error in determining the penetration time.

Consideration of const6.out shows that WAPDEG predicts that the CAMGeneral+PitMultiples model coupled with the CRMGeneral+PitGrowthLaw in the [Neutral Drip, . . .] input segment results in CRM penetration in 113,755.3 years. This value is also in agreement with the hand calculations which predict 108,641.3 years. There is a 4.5% error in determining the penetration time.

Consideration of const7.out shows that WAPDEG predicts that the CAMGeneral+PitMultiples model coupled with the CRMGeneral+PitGrowthLaw in the [Chemical Drip, . . .] input segment also results in CRM penetration in 113,755.3 years. This value is also in agreement with the hand calculations which predict 108,641.3 years. Again, there is a 4.5% error in determining the penetration time.

Consideration of const8.out shows that WAPDEG predicts that the CAMGeneral+PitMultiples model coupled with the CRMExponentialPitRate model in the [No Drip, . . .] input segment results in no CRM penetration in 1,000,000 years of simulation time. This value is not in agreement with the hand calculations which predict 253,969.3 years for humid-air CAM corrosion and 263,402.8 years for aqueous CAM corrosion. The CRMExponentialPitRate model, which was used in the TSPA-1995 analysis, will not be considered for the TSPA-VA analysis. Previous versions of the WAPDEG code that include this model have been baselined.

Nonetheless, future versions of the code will properly implement the CRMExponentialPitRate model.

Consideration of const9.out shows that WAPDEG predicts that the CAMGeneral+PitGrowthLaw model coupled with the CRMLogPitGrowth model in the [Chemical Drip, . . .] input segment results in CAM penetration in 49847.0 years and CRM penetration in 73865.9 years for dripped-on patches. These values are not in agreement with the hand calculations which predict CAM penetration in 38628.1 years and CRM penetration in 58346.1 years for dripped on patches. The CAMGeneral+PitGrowthLaw model will be considered for the TSPA-VA analysis. To determine if this disagreement was due to the numerical interpolation algorithm, two lines were added to the two history files used, const1.hst and const2.hst, with time steps of 30,000 years and 50,000 years and identical temperatures and RH to the rest of the file's entries (in essence this is manual refinement of the time grid). The const9.inp file was then copied unchanged (except for use of the const91.hst and const92.hst files) to const91.inp and the resultant output was presented above (const91.out). Consideration of const91.out, shows that use of the 'refined' history files causes WAPDEG to report CAM penetration in 39242.4 years and CRM penetration in 59066.1 years for dripped-on patches. WAPDEG also reports that dripped-on patches fail in 67524.7 years. These values compare much better to the hand calculated values of 38628.1, 58346.1, and 66910.3 years respectively.

The above exercise indicates the importance of the use of history files with a fine enough time grid to yield the desired level of accuracy from WAPDEG simulations. The reader should be assured that the TSPA-VA history files possess a relatively fine time grid and the program users are well aware of WAPDEG's requirements and limitations.

Consideration of const10.out shows that WAPDEG predicts that the CAMGeneral+PitGrowthLaw model coupled with the CRMArrehiusPit model in the [Chemical Drip, . . .] input segment results in CAM penetration in 96158.5 years for the aqueous case and in 86724.9 years in the humid-air case. CRM penetration is in 124440.8 and 115007.2 years respectively for patches that are and are not dripped on. For dripped-on patches, CRM pit penetration is predicted in 105194.1 years which compares well with the predicted value of 103046.7 years, and CRM patch penetration is predicted in 114265.3 years which compares well with the calculated value of 114265.3 years.

Consideration of const11.out shows that WAPDEG correctly predicts CAM penetrations as before, and predicts dripped-on patches of the CRM will be penetrated in 145118.0 years. This value agrees well with the hand calculated value of 145118.0 years. CRM patches that are not dripped upon are predicted to fail in 135684.4 years, again comparing well with the hand calculated value of 135684.4 years. CRM pit penetration is predicted at 116507.0, which also compares well with the hand calculated value of 107022.4 years (9% error). Likely a finer history time grid would improve the agreement between these times.

Appendix A. Humid-Air CAM Corrosion Model

In this appendix, the corrosion data used in the formulation of the CAM humid-air corrosion model is presented. A detailed discussion of the data sources and numeric manipulations necessary to prepare the data for use in the model is included.

A1. Humid-Air Corrosion Data Compilation

A literature survey was conducted to collect humid-air corrosion data for a suite of cast iron and carbon steel alloys known to have corrosion behavior similar to the candidate carbon steel. The atmospheric corrosion data include the effects of pollutants such as SO₂ and other chemicals in the atmosphere that interact with test specimens. The data also embed any effects of salts that may form on the surface of the corroding specimen due to cyclic wetting and drying. Data from marine sites were excluded because these environments are much more corrosive (mostly due to the presence of chloride salts) than the potential repository near-field environment considered in TSPA-VA.

Important testing parameters that were considered in the data compilation included test duration, average exposure temperature and relative humidity, and average sulfur dioxide content in the test atmosphere. From the general corrosion data collected, a total of 186 data points (Table A1) were included in the model abstraction. The test environments included rural, urban, and industrial sites. A list of the data sources used in the model development is included at the end of this Appendix.

Table A1. Humid-Air Corrosion Data

	T _{avg} (°C)	RH _{avg} (%)	f ₇₀	[SO ₂] (μg/m ³)	time (years)	Depth (μm)	Rate (μm/yr)	Remarks
1	8	78	0.772	71.14	0.25	17.75	71.01	Knotkova, et. al., 1981
2	8	78	0.772	71.14	0.50	23.67	47.34	carbon steel; urban
3	8	78	0.772	71.14	1.00	35.51	35.51	area;
4	8	78	0.772	71.14	2.00	71.01	35.51	Letnany
5	8	78	0.772	71.14	5.00	118.35	23.67	
6	8	78	0.772	71.14	10.00	150.90	15.09	
7	8	81	0.820	89.27	0.50	67.14	134.29	Knotkova-Cermakova, et. al., 1982.
8	8	81	0.820	89.27	1.00	85.71	85.71	carbon steel; urban
9	8	81	0.820	89.27	2.00	107.14	53.57	area;
10	8	81	0.820	89.27	3.00	131.43	43.81	Letnany
11	8	81	0.820	89.27	5.00	150.00	30.00	
12	8	81	0.820	89.27	10.00	190.00	19.00	
13	10	76	0.732	36.47	0.50	21.43	42.86	Knotkova-Cermakova, et. al., 1982.
14	10	76	0.732	36.47	1.00	42.14	42.14	carbon steel; rural
15	10	76	0.732	36.47	2.00	58.57	29.29	area;
16	10	76	0.732	36.47	3.00	71.43	23.81	Hurbanovo

	T _{avg} (°C)	RH _{avg} (%)	f ₇₀	[SO ₂] (μg/m ³)	time (years)	Depth (μm)	Rate (μm/yr)	Remarks
17	10	76	0.732	36.47	5.00	90.00	18.00	
18	10	76	0.732	36.47	10.00	111.43	11.14	
19	18	84	0.849	2.06	1.00	19.18	19.18	Tri, et. al., 1993
20	18	84	0.849	2.06	2.00	26.66	13.33	carbon steel; rural
21	18	84	0.849	2.06	3.00	32.32	10.77	area.
22	18	84	0.849	2.06	4.00	37.06	9.26	Dalat
23	18	84	0.849	2.06	5.00	41.20	8.24	
24	23	83	0.830	2.06	1.00	20.58	20.58	Tri, et. al., 1993
25	23	83	0.830	2.06	2.00	37.33	18.66	carbon steel; rural
26	23	83	0.830	2.06	3.00	52.88	17.63	area.
27	23	83	0.830	2.06	4.00	67.70	16.92	Vinhphu
28	23	83	0.830	2.06	5.00	82.00	16.40	
29	24	82	0.815	2.06	1.00	32.31	32.31	Tri, et. al., 1993
30	24	82	0.815	2.06	2.00	56.37	28.19	carbon steel; urban
31	24	82	0.815	2.06	3.00	78.07	26.02	area;
32	24	82	0.815	2.06	4.00	98.36	24.59	Hanoi
33	24	82	0.815	2.06	5.00	117.66	23.53	
34	27	83	0.825	3.61	1.00	38.44	38.44	Tri, et. al., 1993
35	27	83	0.825	3.61	2.00	61.12	30.56	carbon steel; urban
36	27	83	0.825	3.61	3.00	80.17	26.72	area;
37	27	83	0.825	3.61	4.00	97.18	24.30	HoChiMinh
38	27	83	0.825	3.61	5.00	112.83	22.57	
39	22	81	0.803	57.25	1.00	61.41	61.41	Wei, 1991
40	22	81	0.803	57.25	2.00	92.24	46.12	carbon steel; rural-
41	22	81	0.803	57.25	3.00	117.02	39.01	urban area;
42	22	81	0.803	57.25	4.00	138.55	34.64	Tsing-Hua
43	22	81	0.803	57.25	5.00	157.94	31.59	
44	22	81	0.803	57.25	8.00	208.12	26.02	
45	25	77	0.732	42.44	1.00	42.83	42.83	Wei, 1991
46	25	77	0.732	42.44	2.00	69.05	34.52	carbon steel; rural-
47	25	77	0.732	42.44	3.00	91.30	30.43	urban area;
48	25	77	0.732	42.44	4.00	111.32	27.83	Sun Yat-Sen
49	25	77	0.732	42.44	5.00	129.82	25.96	
50	25	77	0.732	42.44	8.00	179.46	22.43	
51	25	77	0.732	42.44	1.00	42.78	42.78	Wei, 1991
52	25	77	0.732	42.44	2.00	63.73	31.87	carbon steel; rural-
53	25	77	0.732	42.44	3.00	80.46	26.82	urban area;
54	25	77	0.732	42.44	4.00	94.94	23.73	Sun Yat-Sen
55	25	77	0.732	42.44	5.00	107.93	21.59	
56	25	77	0.732	42.44	8.00	141.43	17.68	
57	27	83	0.825	58.7	1.00	45.00	45.00	Southwell and Bultman, 1982
58	27	83	0.825	58.7	2.00	81.00	40.50	wrought iron;
59	27	83	0.825	58.7	4.00	121.00	30.25	tropical area.
60	27	83	0.825	58.7	8.00	175.00	21.88	Panama
61	27	83	0.825	58.7	16.00	310.00	19.38	
62	27	83	0.825	58.7	1.00	42.00	42.00	Southwell and Bultman, 1982
63	27	83	0.825	58.7	2.00	80.00	40.00	wrought iron;

	T _{avg} (°C)	RH _{avg} (%)	f ₇₀	[SO ₂] (μg/m ³)	time (years)	Depth (μm)	Rate (μm/yr)	Remarks
64	27	83	0.825	58.7	4.00	118.00	29.50	tropical area.
65	27	83	0.825	58.7	8.00	182.00	22.70	Panama
66	27	83	0.825	58.7	16.00	290.00	19.00	
67	27	83	0.825	58.7	1.00	35.00	35.00	Southwell and Bultman, 1982
68	27	83	0.825	58.7	2.00	65.00	32.50	carbon steel;
69	27	83	0.825	58.7	4.00	106.00	26.50	tropical area.
70	27	83	0.825	58.7	8.00	160.00	20.00	Panama
71	27	83	0.825	58.7	16.00	290.00	18.13	
72	27	83	0.825	58.7	1.00	38.00	38.00	Southwell and Bultman, 1982
73	27	83	0.825	58.7	2.00	71.00	35.50	carbon steel;
74	27	83	0.825	58.7	4.00	105.00	26.25	tropical area.
75	27	83	0.825	58.7	8.00	163.00	20.38	Panama
76	27	83	0.825	58.7	16.00	304.00	19.00	
77	27	83	0.825	58.7	1.00	35.00	35.00	Southwell and Bultman, 1982
78	27	83	0.825	58.7	2.00	57.00	28.50	carbon steel;
79	27	83	0.825	58.7	4.00	86.00	21.50	tropical area.
80	27	83	0.825	58.7	8.00	128.00	16.00	Panama
81	27	83	0.825	58.7	16.00	218.00	13.63	
82	27	83	0.825	58.7	1.00	31.00	31.00	Southwell and Bultman, 1982
83	27	83	0.825	58.7	2.00	51.00	25.50	cast iron; tropical
84	27	83	0.825	58.7	4.00	79.00	19.75	area.
85	27	83	0.825	58.7	8.00	113.00	14.13	Panama
86	27	83	0.825	58.7	16.00	191.00	11.94	
87	27	83	0.825	58.7	1.00	25.00	25.00	Southwell and Bultman, 1982
88	27	83	0.825	58.7	2.00	42.00	21.00	gray cast iron;
89	27	83	0.825	58.7	4.00	69.00	17.25	tropical area.
90	27	83	0.825	58.7	8.00	99.00	12.38	Panama
91	27	83	0.825	58.7	16.00	151.00	9.44	
92	10	68	0.554	20.0	1.00	31.82	31.82	Townsend and Zoccola, 1982
93	10	68	0.554	20.0	2.00	52.28	26.14	carbon steel; rural
94	10	68	0.554	20.0	4.00	86.92	21.73	area.
95	10	68	0.554	20.0	8.00	129.52	16.19	Saylorsburg, PA
96	13	66	0.502	245.0	1.00	50.00	50.00	Townsend and Zoccola, 1982
97	13	66	0.502	245.0	2.00	65.90	32.95	carbon steel; urban
98	13	66	0.502	245.0	4.00	81.24	20.31	area.
99	13	66	0.502	245.0	8.00	101.68	12.71	Newark, NJ
100	10	68	0.554	20.00	1.50	44.00	29.33	Komp, 1987
101	10	68	0.554	20.00	3.50	73.00	20.86	carbon steel; semi-
102	10	68	0.554	20.00	7.50	117.00	15.60	rural area.
103	10	68	0.554	20.00	15.50	179.00	11.55	S. Bend, PA
104	10	71	0.625	300.0	1.50	56.00	37.33	Komp, 1987
105	10	71	0.625	300.0	3.50	93.00	26.57	carbon steel;
106	10	71	0.625	300.0	7.50	130.00	17.33	semi-industrial area
107	10	71	0.625	300.0	15.50	193.39	12.48	Monroeville, PA
108	13	66	0.502	245.0	0.50	35.00	70.00	Komp, 1987
109	13	66	0.502	245.0	1.50	60.00	40.00	carbon steel;
110	13	66	0.502	245.0	3.50	84.00	24.00	industrial area.

	T _{avg} (°C)	RH _{avg} (%)	f ₇₀	[SO ₂] (µg/m ³)	time (years)	Depth (µm)	Rate (µm/yr)	Remarks
111	13	66	0.502	245.0	7.50	104.00	13.87	Newark, NJ
112	13	66	0.502	245.0	15.50	134.00	8.65	
113	10	65	0.480	406	0.33	19.22	58.24	Haynie and Upham, 1971
114	10	65	0.480	406	0.67	30.48	45.72	urban area;
115	10	65	0.480	406	1.33	52.8	39.60	Chicago
116	10	65	0.480	406	0.33	18.59	56.33	
117	10	65	0.480	406	0.67	30.42	45.63	
118	10	65	0.480	406	1.33	45.64	34.23	
119	12	69	0.576	79	0.33	14.64	44.36	Haynie and Upham, 1971
120	12	69	0.576	79	0.67	18.94	28.41	urban area;
121	12	69	0.576	79	1.33	23.18	17.39	Cincinnati
122	12	69	0.576	79	2.67	28.82	10.81	
123	12	69	0.576	79	5.33	37.98	7.12	
124	12	69	0.576	79	0.33	14.05	42.58	
125	12	69	0.576	79	0.67	18.13	27.19	
126	12	69	0.576	79	1.33	22.51	16.88	
127	12	69	0.576	79	2.67	28.1	10.54	
128	12	69	0.576	79	5.33	37.34	7.00	
129	5	67	0.534	118	0.33	12.24	37.09	Haynie and Upham, 1971
130	5	67	0.534	118	0.67	25.12	37.68	urban area;
131	5	67	0.534	118	1.33	34.19	25.64	Detroit
132	5	67	0.534	118	2.67	49.95	18.73	
133	5	67	0.534	118	5.33	74.13	13.90	
134	5	67	0.534	118	0.33	16.12	48.85	
135	5	67	0.534	118	0.67	24.39	36.58	
136	5	67	0.534	118	1.33	33.64	25.23	
137	5	67	0.534	118	2.67	50.15	18.81	
138	5	67	0.534	118	5.33	74.07	13.89	
139	16	70	0.595	39	0.33	9.72	29.45	Haynie and Upham, 1971
140	16	70	0.595	39	0.67	11.63	17.44	urban area;
141	16	70	0.595	39	1.33	15.97	11.98	Los Angeles
142	16	70	0.595	39	2.67	19.57	7.34	
143	16	70	0.595	39	5.33	25.81	4.84	
144	16	70	0.595	39	0.33	8.98	27.21	
145	16	70	0.595	39	0.67	10.42	15.63	
146	16	70	0.595	39	1.33	14.82	11.12	
147	16	70	0.595	39	2.67	18.71	7.02	
148	16	70	0.595	39	5.33	24.83	4.66	
149	19	77	0.739	24	0.33	10.12	30.67	Haynie and Upham, 1971
150	19	77	0.739	24	0.67	16.51	24.76	urban area;
151	19	77	0.739	24	1.33	24.52	18.39	New Orleans
152	19	77	0.739	24	2.67	35.38	13.27	
153	19	77	0.739	24	0.33	8.56	25.94	
154	19	77	0.739	24	0.67	14.82	22.23	
155	19	77	0.739	24	1.33	23.48	17.61	
156	19	77	0.739	24	2.67	34.85	13.07	
157	12	66	0.503	218	0.33	19.5	59.09	Haynie and Upham, 1971

	T _{avg} (°C)	RH _{avg} (%)	f ₇₀	[SO ₂] (μg/m ³)	time (years)	Depth (μm)	Rate (μm/yr)	Remarks
158	12	66	0.503	218	0.67	24.16	36.24	urban area;
159	12	66	0.503	218	1.33	32.09	24.07	Philadelphia
160	12	66	0.503	218	2.67	41.37	15.51	
161	12	66	0.503	218	5.33	51.5	9.66	
162	12	66	0.503	218	0.33	18.09	54.82	
163	12	66	0.503	218	0.67	23.16	34.74	
164	12	66	0.503	218	1.33	31.54	23.66	
165	12	66	0.503	218	2.67	40.41	15.15	
166	12	66	0.503	218	5.33	50.95	9.55	
167	13	73	0.667	34	0.33	10.8	32.73	Haynie and Upham, 1971
168	13	73	0.667	34	0.67	17.74	26.61	urban area;
169	13	73	0.667	34	1.33	28.61	21.46	San Francisco
170	13	73	0.667	34	2.67	41.44	15.54	
171	13	73	0.667	34	5.33	57.36	10.76	
172	13	73	0.667	34	0.33	8.57	25.97	
173	13	73	0.667	34	0.67	16.34	24.51	
174	13	73	0.667	34	1.33	27.95	20.96	
175	13	73	0.667	34	2.67	41.18	15.44	
176	13	73	0.667	34	5.33	57.94	10.86	
177	13	63	0.429	126	0.33	14.58	44.18	Haynie and Upham, 1971
178	13	63	0.429	126	0.67	20.5	30.75	urban area;
179	13	63	0.429	126	1.33	25.87	19.40	Washington, DC
180	13	63	0.429	126	2.67	33.94	12.73	
181	13	63	0.429	126	5.33	43.04	8.07	
182	13	63	0.429	126	0.33	13.14	39.82	
183	13	63	0.429	126	0.67	19.34	29.01	
184	13	63	0.429	126	1.33	25.16	18.87	
185	13	63	0.429	126	2.67	33.35	12.51	
186	13	63	0.429	126	5.33	42.59	7.99	

The longest-term data were from the 16-year corrosion test program conducted by the Naval Research Laboratory in a tropical environment in Panama (Southwell, et al., 1976; Southwell and Bultman, 1982). The exposure conditions for the entire set of data range from 5 to 27°C average temperature, 63 to 84 % average relative humidity (RH), and an average SO₂ level of 2 to 406 μg SO₂/m³.

The first six data lines shown in Table A1 were derived from Knotkova, Holler, and Vickova (1981). The average T and RH were obtained from Table 1 of this reference. f₇₀, the fraction of exposure time during which RH ≥ 70%, was determined using an equation to be discussed in the next section of this appendix. The weight loss in g/m² at each exposure time was determined by graphical analysis of Figure 1 in the above cited reference. This value is divided by the density of mild steel, 7.86x10⁶ g/m³, and multiplied by 10⁶ μm/m to obtain the penetration depth. Division of this depth by the exposure time gives an average penetration rate over the time period considered.

The concentration of SO₂ is given as 71.6 mg SO₂/m²day. Multiplication by 1.25 (= Molecular Weight of SO₃ / Molecular Weight of SO₂) gives 89.5 mg SO₃/m²day which is then divided by 100 to change units to mg SO₃/dm²day. Using the following equation from Guttman (1967):

$$\text{ppm SO}_2 = 0.023 x + 0.0082 x^2$$

and inserting our value of 0.895 mg SO₃/dm²day as 'x', we can obtain the ppm SO₂ in the exposure atmosphere which can readily be converted to µg/m³.

The next two data sets (lines 7 – 18) were derived from Knotkova-Cermakova, Vickova, and Honzak (1982). The average T and RH are given in Table 1 of this publication as well as average values for mg SO₂/m²day relevant to the exposure period. Transformation of SO₂ data to units of µg/m³ was accomplished as discussed above. Corrosion depths in µm at a given exposure time were read directly from Figure 5 of this reference.

The next four data sets (lines 19-38) were derived from Tri, Huy, Cuong, and San (1993). The average T and RH values were found in Table 1 of this work. The authors fit their penetration data to an equation of the form:

$$\text{weight loss (g/m}^2\text{)} = A t^B$$

and presented A and B values for each city as shown below:

Table A2. Taken from Table 2 in Tri, Huy, Cuong, and San (1993).

City	A (g/m ²)	B
Dalat	150.772	0.475
Vinhphu	161.748	0.859
Hanoi	253.964	0.803
Ho Chi Minh City	302.158	0.669

The obtained weight loss values were then divided by the density of mild steel 7.86x10⁶ g/m³ and multiplied by 10⁶ µm/m to obtain the penetration depth.

A value of 2.7 mg SO₂/m²day for Hanoi was obtained from Binh, Huy, and Strecalov (1993). This value was used for Dalat and Vinhphu as well, while a value of 4.7 mg SO₂/m²day was obtained from this reference for Ho Chi Minh City.

The next three data sets (lines 39-56) were derived from Wei (1991). The average T, RH and mg SO₂/m²day values were found in Table 2 of this work. The authors fit their penetration data to an equation of the form:

$$\text{weight loss (g/m}^2\text{)} = A t^B$$

and presented A and B values for each city as shown below:

Table A3. Taken from Table 3 of Wei (1991).

City	A (g/m ²)	B
Tsing-Hua	14.28	0.587
Sun Yat-Sen	7.73	0.689
Sun Yat-Sen	10.25	0.575

The obtained weight loss values were then divided by the density of mild steel 7.86×10^6 g/m³ and multiplied by 10^6 $\mu\text{m}/\text{m}$ to obtain the penetration depth.

The next seven data sets (lines 57 - 91) were derived from Southwell and Bultman (1982). The average T and RH were determined from averaging values read from the 'Pacific Side' plots appearing in Figure 64.1 of this publication. The concentration of SO₄ was quoted in Table 64.4 as 0.88 mg/10m³ SO₄, dividing this by 1.5 (= Molecular Weight of SO₄ / Molecular Weight of SO₂) and application of a few simple unit conversions yields 58.7 $\mu\text{g}/\text{m}^3$ SO₂. Average corrosion depths were read directly from Table 64.5 in units of μm .

The next two data sets (lines 92 - 99) were derived from Townsend and Zoccola (1982). The average T, RH, and $\mu\text{g}/\text{m}^3$ SO₂ were determined from atmospheric data given in Table 6 of Haynie, Spence, et. al. (1990). The environmental conditions corresponding to State College, PA were used for those of Saylorsburg, PA. Average corrosion depths in μm were read directly from Figures 2 and 4 of Townsend, and Zoccola (1982).

The next three data sets (lines 100 - 112) were derived from Komp (1987). The average T, RH, and $\mu\text{g}/\text{m}^3$ SO₂ were determined from atmospheric data given in Table 6 of Haynie, Spence, et. al. (1990). The environmental conditions corresponding to State College, PA were used for those of South Bend, PA. The environmental conditions corresponding to Pittsburgh, PA were used for those of Monroeville, PA. Average corrosion depths in μm were read directly from Table 2 in Komp (1987) and the last data point (15.5 yr) for Monroeville, PA was calculated using coefficients found in Table 3.

The last eight data sets (lines 113 - 186) were derived from Haynie and Upham (1971). The average T was determined from atmospheric data given in Table 3 of Haynie and Upham (1974). The average RH was determined from Table 3 of Haynie and Upham (1970) and was chosen to be that corresponding to the maximum nominal exposure time, i.e. the last row of data corresponding to each exposure location. Similarly, the $\mu\text{g}/\text{m}^3$ SO₂ was determined from Table 3 in Haynie and Upham (1971). The average corrosion depth in μm was obtained directly from Table 2 of Haynie and Upham (1971).

A2. Exposure Parameter Transformations

Atmospheric exposure conditions relevant to the humid-air corrosion data presented in Table A1 can fluctuate over periods as short as a day. Therefore, it is necessary to transform the corrosion data (time of exposure, corrosion rate, etc.) to reflect the fact that humid-air corrosion does not actively occur during the entire exposure time.

In the data transformation, three generalized relationships were needed to transform the exposure conditions and corrosion data:

- (1) a relationship to estimate the fraction of the exposure time during which humid-air corrosion was active;
- (2) a relationship to estimate the average relative humidity during that fraction of the time;
- (3) a relationship to estimate the average temperature during that fraction of the time.

As detailed exposure conditions were not available for the corrosion data reported in Table A1, an effort was made to develop generalized relationships. To this end, hourly weather data for a 10-year period from 9 geographical areas were received from the Western Climatic Center in Reno, Nevada (Prowell, 1994). The areas were selected to cover a wide range of weather conditions from hot and dry (Las Vegas, NV, Albuquerque, NM, and Tucson, AZ) to warm and humid (Honolulu, HI, and Arcata, CA) to cold and humid (Great Falls, MT, Denver, CO, Cheyenne, WY, and Anchorage, AK). These data have been provided in electronic format (MI: 30048-MO4-001).

A2.1 Time Fraction for Relative Humidity Greater than 70 %

Numerous experimental results of iron specimens exposed to controlled humidity conditions have shown that at around 70% RH their surfaces become covered with a 'thin' water film and 'active' corrosion initiates (Vernon, 1933; Phipps and Rice, 1979). Accordingly, 70% RH was chosen as the threshold level at which 'active' humid-air corrosion initiates. Using the weather data, a relationship for estimating the fraction of time during which the relative humidity is greater than or equal to 70% was developed as a function of the average relative humidity and average temperature (averages were taken over 720 hour (30 day) 'months'). There were a total of 121 such months for each of the 9 cities considered. These data were fit to a sigmoidal function expressed as:

$$f_{70} = \frac{1}{1 + \exp(0.92781 + \frac{28.01246(56.44258 - RH_{avg})}{(T_{avg} + 273.15)})} \quad (A1)$$

where f_{70} is the fraction of time during which $RH \geq 70\%$, RH_{avg} is the monthly average relative humidity (%), and T_{avg} is the monthly average temperature ($^{\circ}C$). The geographical weather data

and calculations used to obtain the coefficients in Equation (A1) have also been provided in electronic form (MI: 30048-MO4-001). The fourth column of Table A1 contains the value of f_{70} calculated from the average T and RH using Equation (A1).

A2.2 Reduction of Relative Humidity and Temperature

A 'new' exposure time for each corrosion data point was estimated by multiplying the actual exposure time with the time fraction during which $RH \geq 70\%$ (f_{70}), calculated with Equation (A1). This is meant to represent the time fraction during which corrosion was active. Accordingly, it was necessary to estimate a 'new' average relative humidity and 'new' average temperature for each corrosion data point to more accurately reflect the exposure conditions during that period of time during which humid-air corrosion was active ($RH \geq 70\%$). The weather data were utilized to develop relationships for estimating the 'new' average relative humidity and 'new' average temperature as follows: for each geographical location, the average monthly (30 day) relative humidity and temperature were calculated for the periods with $RH \geq 70\%$, and fit to a function of the monthly average relative humidity and average temperature. The relationship developed for the new average relative humidity is:

$$RH_{new} = 83.24 - 0.1422 RH_{avg} - 0.1054 T_{avg} + 0.002377 RH_{avg}^2 \quad (A2)$$

where RH_{new} is the new average relative humidity (%). Similarly, the new average temperature was fit to the following equation:

$$T_{new} = -13.33 + 0.1512 RH_{avg} + 1.017 T_{avg} \quad (A3)$$

where T_{new} is the new average temperature ($^{\circ}C$). The calculations used to obtain the coefficients in Equations (A2) and (A3) have also been provided in electronic form (MI: 30048-MO4-001).. The new average relative humidity and temperature are assumed to be the representative exposure conditions for formulation of the humid-air corrosion models used in WAPDEG.

The net result of these data transformations can be considered conservative in that the 'new' humid-air corrosion rate would be higher than the one reported in the literature used as the 'new' exposure time is shorter than the reported one.

A3. Humid-Air Corrosion Data References

Binh, D.T; Huy, V.D.; and Strecalov, P.V.; 1993. "Atmospheric Corrosion of Metals and Alloys After Five Years of Tests in Vietnam," *Progress in the Understanding and Prevention of Corrosion*, 1, 126-130. J.M. Costa and A.D. Mercer (eds.).

Guttman, H.; 1967. "Effects of Atmospheric Factors on the Corrosion of Rolled Zinc," *Metal Corrosion in the Atmosphere*, 223, Philadelphia, PA: ASTM Special Publication No. 435, American Society for Testing and Materials.

Haynie, F.H.; Spence, J.W.; Lipfert, F.W.; Cramer, S.D.; and McDonald, L.G.; 1990. "Evaluation of an Atmospheric Damage Function for Galvanized Steel," *Corrosion Testing and Evaluation: Silver Anniversary Volume*, 225-240. R. Baboian and S.W. Dean, eds., ASTM STP 1000.

Haynie, F.H.; and Upham, J.B.; 1974. "Correlation Between Corrosion Behavior of Steel and Atmospheric Pollution Data," *Corrosion in Natural Environments*, 33-51. ASTM STP 558.

Haynie, F.H.; and Upham, J.B.; 1971. "Effects of Atmospheric Pollutants on Corrosion Behavior of Steels," *Materials Production and Performance*, 10:11, 18-40.

Haynie, F.H.; and Upham, J.B.; 1970. "Effects of Atmospheric Sulfur Dioxide on the Corrosion Behavior of Zinc," *Materials Production and Performance*, 9:8, 35-40.

Knotkova, D.; Holler, P.; and Vickova, J.; 1981. "The Evaluation of the Microclimate Effect on the Characteristics of Environment and Corrosion of Metals," *Proceedings of the 8th International Congress on Metallic Corrosion*, 1, 859-864. Mainz, FRG.

Knotkova-Cermakova, D.; Vickova, J.; and Honzak, J.; 1982. "Atmospheric Corrosion of Weathering Steels," *Atmospheric Corrosion of Metals*, 7-44. ASTM STP 767, S.W. Dean, Jr., and E.C. Rhea (eds.), American Society for Testing and Materials.

Komp, M.E.; 1987. "Atmospheric Corrosion Ratings of Weathering Steels-Calculation and Significance," *Materials Performance*, 42-44.

Phipps, P.B. and Rice, D.W. 1979. "The Role of Water in Atmospheric Corrosion." *Corrosion Chemistry*, 235-261, G. R. Brubaker, and P. B. Phipps (eds.), ACS Symposium Series 89, American Chemical Society. 4764.

Prowell, G. 1994. Personal Communication, December 28.

Southwell, C.R. and Bultman, J.D. 1982. "Atmospheric Corrosion Testing in the Tropics." *Atmospheric Corrosion*, 943-967. W.H. Ailor (ed.). Wiley-Interscience. NNA.19890919.0304.

Townsend, H.E.; and Zoccola, J.C.; 1982. "Eight-Year Atmospheric Corrosion Performance of Weathering Steel in Industrial, Rural, and Marine Environments," *Atmospheric Corrosion of Metals*, 45-59. ASTM STP 767, S.W. Dean, Jr., and E.C. Rhea (eds.), American Society for Testing and Materials.

Tri, N.Q.; Huy, V.D.; Cuong, L.V.; San, P.Th.; 1993. "Atmospheric Corrosion Testing Vietnam," *Progress in the Understanding and Prevention of Corrosion*, 1, 105-114. J.M. Costa and A.D. Mercer (eds.).

Vernon, W.H. 1933. "The Role of the Corrosion Product in the Atmospheric Corrosion of Iron." *Transactions of the Electrochemical Society*, 64, 31-41.

Wei, F.-I.; 1991. "Atmospheric Corrosion of Carbon Steels and Weathering Steels in Taiwan," *British Corrosion Journal*, 26:3, 209-214.

Appendix B. Aqueous CAM Corrosion Model

B1. Aqueous Corrosion Data Compilation

Aqueous corrosion data (Table B1 – 70 observations) was gathered from a number of sources. Long-term corrosion data (up to 16 years) in polluted river water (Coburn, 1978) and in tropical lake water (Southwell and Alexander, 1970) were used to estimate the time dependence of aqueous corrosion. This data was also used as a baseline since it includes the potential effects of microbial activity and various chemical species dissolved in the waters. A set of short-term (100 days) corrosion data of mild steel in distilled water at temperatures from 5 to 90°C (Brasher and Mercer, 1968; Mercer et al, 1968) was used to develop the temperature dependence of aqueous corrosion. This dependence was incorporated into the aqueous corrosion models, but it was scaled to be consistent with the lake and river water since distilled water provides a much less corrosive environment.

Table B1. Aqueous Corrosion Data

	time (years)	Temperature (°C)	Depth (μm)	Comments
1	1	27.78	195.58	Southwell & Alexander, 1970
2	2	27.78	304.80	Table 2 – Class A
3	4	27.78	431.80	carbon steel; lake water
4	8	27.78	558.80	Panama
5	16	27.78	711.20	
6	1	27.78	190.50	Southwell & Alexander, 1970
7	2	27.78	304.80	Table 2 – Class B
8	4	27.78	406.40	carbon steel; lake water
9	8	27.78	508.00	Panama
10	16	27.78	635.00	
11	1	27.78	160.02	Southwell & Alexander, 1970
12	2	27.78	241.30	Table 2 – Class C
13	4	27.78	355.60	carbon steel; lake water
14	8	27.78	482.60	Panama
15	16	27.78	635.00	
16	1	27.78	200.66	
17	2	27.78	304.80	Southwell & Alexander, 1970
18	4	27.78	457.20	Table 2 – Class D
19	8	27.78	584.20	carbon steel; lake water
20	16	27.78	736.60	Panama
21	1	27.78	208.28	Southwell & Alexander, 1970
22	2	27.78	304.80	Table 2 – Class M.
23	4	27.78	355.60	cast steel; lake water
24	8	27.78	482.60	Panama
25	16	27.78	660.40	
26	1	27.78	177.80	Southwell & Alexander, 1970

	time (years)	Temperature (°C)	Depth (µm)	Comments
27	2	27.78	304.80	Table 2 - Class O
28	4	27.78	381.00	cast iron; lake water
29	8	27.78	584.20	Panama
30	16	27.78	838.20	
31	1	27.78	193.04	Southwell & Alexander, 1970
32	2	27.78	304.80	Table 2 - Class N
33	4	27.78	406.40	cast steel; lake water
34	1	27.78	200.66	Southwell & Alexander, 1970
35	2	27.78	330.20	Table 2 - Class P
36	4	27.78	431.80	cast iron; lake water
37	1	15.00	182.90	Coburn - 1978
38	1	15.00	161.80	Figure 4
39	1	15.00	135.50	carbon steel; river water
40	2	15.00	306.60	Charleroi, PA
41	2	15.00	236.80	
42	2	15.00	229.00	
43	2	15.00	223.60	
44	2	15.00	210.60	
45	4	15.00	431.60	
46	4	15.00	410.40	
47	4	15.00	389.60	
48	4	15.00	294.80	
49	8	15.00	863.20	
50	8	15.00	631.20	
51	8	15.00	579.20	
52	8	15.00	442.40	
53	8	15.00	431.20	
54	1	15.00	106.60	Coburn - 1978
55	1	15.00	103.90	Figure 4
56	1	15.00	100.00	carbon steel; river water
57	2	15.00	189.40	Kittanning, PA
58	2	15.00	176.20	
59	4	15.00	278.80	
60	4	15.00	273.60	
61	4	15.00	260.40	
62	8	15.00	484.00	
63	8	15.00	479.20	
64	0.27	5.00	1.76	Brasher & Mercer, 1968
65	0.27	25.00	9.60	Mercer, et. al., 1968
66	0.27	40.00	20.15	mild steel; distilled water;
67	0.27	60.00	42.73	
68	0.27	70.00	42.26	

	time (years)	Temperature (°C)	Depth (µm)	Comments
69	0.27	80.00	18.35	
70	0.27	90.00	13.53	

A list of the data sources used in the model development is included at the end of this Appendix. The corrosion depths shown in the first 36 data lines of Table B1 were derived from Table 2 of Southwell and Alexander (1970). In this study, samples of cast iron and carbon steel were submerged in fresh water (Gatun Lake) in Panama. The average temperature of the lake water was reported to be 82°F (27.8°C).

Data rows 37-63 in Table B1 were derived from plots of average corrosion rate versus exposure time appearing in Figure 4 of Coburn (1978). In this study, carbon steel samples were exposed to river waters (Monongahela River, Charleroi, PA; Allegheny River, Kittanning, PA) for periods up to 8 years. While the average temperature of these river waters was not reported, a value of 15°C was assumed.

The data in rows 64-70 of Table B1 were derived from Brasher and Mercer (1968) and Mercer, et. al. (1968). In this study two duplicate mild steel coupons were submerged in distilled water for 100 days at various exposure temperatures. The corrosion depth values reported in column 4 of Table B1 are derived from the average weight loss of both specimens.

The regression fit to the aqueous CAM corrosion model is provided in electronic format (MI: 30048-MO4-001).

B2. Aqueous Corrosion Data References

Brasher, D.M. and Mercer, A.D. 1968. "Comparative Study of Factors Influencing the Action of Corrosion Inhibitors for Mild Steel in Neutral Solution. I. Sodium Benzoate." *British Corrosion Journal*, 3, 121-129.

Coburn, S.K. 1978. "Corrosion in Fresh Water: Properties and Selection: Irons and Steels." *Metals Handbook*, Ninth Edition, 1, 733-738. Metal Parks, Ohio: American Society for Metals. 223362.

Mercer, A.D.; Jenkins, I.R.; and Rhoades-Brown, J.E. 1968. "Comparative Study of Factors Influencing the Action of Corrosion Inhibitors for Mild Steel in Neutral Solution. III. Sodium Nitrite." *British Corrosion Journal*, 3, 136-144.

Southwell, C.R. and Alexander, A.L. 1970. "Corrosion of Metals in Tropical Waters. Structural Ferrous Metals." *Materials Protection*, 14-23. 223416.

Appendix C. Batchwpdg – WAPDEG Unix Script

C1. Purpose

The purpose of the Unix Script is to facilitate the execution of multiple WAPDEG runs through the use of a single command. This script is in no way necessary for the execution of WAPDEG.

C2. Listing

```
#!/bin/sh
# batchwpdg:
# set -x
# WAPDEG executable name here is wap306

if [ $# != 1 ]
then
    echo "You must supply exactly one argument.";exit 1
fi

if [ ! -f $1 ]
then
    echo "file \"$1\" does not exist.";exit 1
fi

date;
echo "RUNNING WAPDEG"

date > $1.log;
echo "using file: \"$1 >> $1.log"

for i in `cat $1`
do
    echo $i > $1.temp
    wap306 < $1.temp >> $1.log 2>#1
done

rm $1.temp
date >> $1.log
exit 1
```

C3. Description

Batchwpdg is a short Unix script file designed to run in the bash shell. To execute multiple WAPDEG simulations type at the Unix prompt:

```
% batchwpdg filename
```

where filename is the name of an ASCII text file that contains one WAPDEG input (*.inp) filename per line. An easy way to create this file is to type at the Unix prompt:

```
% ls *.inp > filename
```

Batchwpdg will also create a filename.log file, within the directory from which it is executed, into which are echoed the date and time that batchwpdg execution began and then the screen output from the various WAPDEG simulations that are executed. The filename.log file ends with the date and time that execution of batchwpdg ended.

Appendix D. Post306 – The WAPDEG Post Processor

D1. Summary

Post306 is a postprocessor for WAPDEG (version 3.06): It reads the *.out, *.bin, and *.pat files from a WAPDEG (version 3.06) simulation and produces data files suitable for graphing as well as a RIP (Integrated Probabilistic Simulator for Environmental Systems, version 5.x) (RIP, 1997) input table.

D2. Description and Testing

Post306 is installed on several personal computers (CRWMS-M&O tags 112371, 112517, 111033) and a variety of Hewlett-Packard HP-UX 20 workstations (CRWMS-M&O tags 102877, 112515, 108319, 107436, 107437, 108335, 111031).

D2.1 Description of Post306

Post306 is a postprocessor for WAPDEG (version 3.06). It reads the *.out, *.bin, and *.pat files from a WAPDEG (version 3.06) simulation and produces data files suitable for graphing (*.dat and *.asc) as well as a RIP (Integrated Probabilistic Simulator for Environmental Systems, version 5.x) (RIP, 1997) input table. The *.dat file is used to produce graphs of the first penetration times of the waste package outer barrier (CAM) and inner barrier (CRM) as well as the diffusion start and advection start times (defined later). The *.asc file is used to produce graphs of the average number pits and patches per failed waste packages versus time.

If a non-dripping WAPDEG simulation is being post processed, then the diffusion start times are defined to be the times at which waste package failure occurs. If a dripping WAPDEG simulation is being post processed, then the advection start time curve is defined to be the waste package failure curve.

The RIP table contains the same information as the *.asc file, however contains fewer points (≤ 83 , obtained by averaging) and is formatted as a three dimensional RIP lookup table (RIP, 1997).

D3. Description of the Post306 Code

Post306 is a FORTRAN program 649 lines in extent. It conforms to the FORTRAN 77 standard and is thus highly portable. WAPDEG has been compiled with the FORTRAN 77 compilers on various HP UX 20.0 – PA 2.0 UNIX systems, and in Digital FORTRAN 5.0 in the Windows/PC environments. Post306 is designed to run independently of any other software application.

Post306 prompts the user for a filename prefix (hereafter represented by a "*"). After verifying that the *.out file exists, post306 opens the *.out file to determine how many waste packages where simulated (stored in variable cpckg), how many waste packages were written to the *.bin file (stored in the variable npckg), and how many waste packages were written to the *.pat file (stored in the variable mpckg). Also, the user-supplied comments (appearing before the "START OF PARAMETERS") in the header of the *.out file are copied to the RIP table header along with the WAPDEG filename prefix and WAPDEG code version number.

The *.out file contains the input file (*.inp) as a header and then columns of data relevant to the degradation of each package simulated. The table below elucidates the contents of each column in the *.out file:

Column	Contents
1	Package number simulated
2	Advection start time, i.e., the maximum of the earliest patch failure time amongst the top patches which are dripped upon and amongst bottom patches that are dripped upon or not. The advection time corresponds to the earliest time at which water can enter the waste package (from the top), interact with the waste form, and then leave the waste package (from the bottom) to enter the near field environment.
3	Diffusion start time, i.e., the earliest penetration (of any kind) time in the bottom of the waste package.
4	Structural failure time. Interpolated time when the CRM average general corrosion depth (over all patches in a package) surpasses the user supplied structural failure percentage of the total CRM thickness.
5	First penetration (of any type) time for the CAM outer barrier.
6	First penetration (of any type) time for the CRM inner barrier.
7	First pit penetration time (years) of the top of the CAM.
8	First pit penetration time (years) of the side of the CAM.
9	First pit penetration time (years) of the bottom of the CAM.
10	First not-dripped-on patch penetration time (years) of the top of the CRM.
11	First not-dripped-on patch penetration time (years) of the side of the CRM.
12	First not-dripped-on patch penetration time (years) of the bottom of the CRM.
13	First dripped-on pit penetration time (years) of the top of the CRM.
14	First dripped-on pit penetration time (years) of the bottom of the CRM.
15	First dripped-on patch penetration time (years) of the top of the CRM.
16	First dripped-on patch penetration time (years) of the bottom of the CRM.
17	Cumulative pit penetrations of the CRM (at any location) at user supplied time 1.
18	Cumulative pit penetrations of the CRM (at any location) at user supplied time 2.
19	Cumulative pit penetrations of the CRM (at any location) at user supplied time 3.
20	Cumulative pit penetrations of the CRM (at any location) at user supplied time 4.
21	Cumulative patch penetrations of the CRM (at any location) at user supplied time 1.
22	Cumulative patch penetrations of the CRM (at any location) at user supplied time 2.
23	Cumulative patch penetrations of the CRM (at any location) at user supplied time 3.
24	Cumulative patch penetrations of the CRM (at any location) at user supplied time 4.

Post306 does not read the entire *.out file; only columns 2, 3, 5, 6 and 17 though 24. Post306 sorts each column in ascending order (i.e. independently) and then writes them to the *.dat post processed file. The *.dat file has 14 columns and the same number of rows as the *.out file (this

is equal to the number of packages simulated). The first column is the cumulative fraction of packages (i.e.

$$\frac{\text{Row \#}}{\text{Total Number of Rows}}$$

and the second column contains $1 -$ (the cumulative fraction of packages). The third through sixth columns contain the sorted first penetration times of the CAM, diffusion start times, advection start times, and first penetration times of the CRM, respectively. These correspond to columns 5, 3, 2, and 6 of the *.out file. Columns 7 through 10 of the *.dat file contain the sorted cumulative number of pit penetrations. These correspond to columns 17 through 20 of the *.out file. Columns 11 through 14 of the *.dat file contain the sorted cumulative number of patch penetrations. These correspond to columns 21 through 24 of the *.out file.

The *.dat file is used to produce graphs for visual presentation of WAPDEG results and is not used as RIP input.

After writing the *.dat file, Post306 reads the *.bin and *.pat files. Before discussing the *.pat and *.bin files, it is instructional to consider how WAPDEG 'bins' temporal data. WAPDEG does not report precise penetration times (beyond those already discussed in the out file); rather logarithmically distributed time bins are used for this purpose. The user specifies the start and end times for binning (bin0 and binf, respectively) and the total number of bins (nbin). To determine the bin in which to place a pit or patch that penetrated the package at time t, one should use the following equation:

$$\text{bin number} = \text{trunc} \left[n\text{bin} * \left(\frac{\ln(t) - \ln(\text{bin0})}{\ln(\text{binf}) - \ln(\text{bin0})} \right) + 1 \right]$$

where trunc represents the truncation operator and assures 'bin number' is an integer that ranges from 1 to (nbin + 1). Using this methodology, a time bin array is created for each waste package whose array elements contain the cumulative number of pit (in the bin file) or patch (in the patch file) penetrations. Penetrations added to a given bin have penetration times greater than or equal to the time associated with that bin. After a waste package has been simulated, the pit penetration array is appended to the bin (*.bin) file and the patch penetration array is appended to the patch (*.pat) file. A portion of a patch file is reproduced below:

1	1019	1201	124440.76
0	0	400	401
0	0	400	401
0	0	400	401
.	.	.	.
.	.	.	.
.	.	.	.

The first line of the patch file consists of four header values; the package number, the array index of the first time bin that contains a nonzero value, the array index of the last time bin, and the first patch penetration time. The subsequent lines of output are composed of five columns:

Column	Contents
1	Cumulative patch penetrations for dripped-on top patches.
2	Cumulative patch penetrations for dripped-on bottom patches.
3	Cumulative patch penetrations for not-dripped-on top patches.
4	Cumulative patch penetrations for not-dripped-on side patches.
5	Cumulative patch penetrations for not-dripped-on bottom patches.

These 'subsequent lines of output' are followed immediately by the three header values for the second package and so on until the final package's results are reported.

The bin file (*.bin) is quite similar to the patch file in header format, however, since the current conceptual model does not allow for pit growth in the absence of drips and waste package sides are never dripped upon, the bin file's 'subsequent lines of output' contain only two columns reporting cumulative pit penetrations of the top and bottom surfaces, i.e.:

```

1      1019      1201  124440.76
      0          0
      0          0
      0          0
      0          0
      0          0
      0          0
      0          0
      .          .
      .          .
      .          .

```

Post306 calculates cumulative sums over all the waste packages for each time bin for each column of the *.bin (stored in the sum1pit and sum2pit arrays) and *.pat (stored in the sum1pat, sum2pat, sum3pat, sum4pat, and sum5pat arrays) files.

If a non-dripping WAPDEG simulation is being post processed (defined by the fifth and sixth characters in the filename prefix being "nd"), then the diffusion start times are considered to be the waste package failure times. If a dripping WAPDEG simulation is being post processed (defined by the fifth and sixth characters in the filename prefix being anything other than "nd"), then the advection start times are considered to be the waste package failure times.

Post306 interpolates the fraction of waste packages failed (defined by first breach, not diffusion or advection as above) to the times represented by the time bin array. This representation of the waste package failure curve is stored in the array wpbc. The average number of pits at each of the times represented by the time bin array is calculated by division of the cumulative sum of pits over all the waste packages through the bottom of the waste packages (sum2pit) by the waste

package failure curve. The average number of patches at each of the times represented by the time bin array is calculated by division of the cumulative sum of patches over all the waste packages through the bottom of the waste packages (sum2pat + sum5pat) by the waste package failure curve.

The waste package failure curve (array wpbc) is then divided (element by element) by the total number of waste packages simulated, and thus becomes the fraction of waste packages failed at each of the times represented by the time bin array.

The *.asc file is then written. The *.asc file consists of (nbin+1) rows by four columns. The row number corresponds to the time bin array index. The first column is the time represented by the time bin array index. The second column is the fraction of waste packages failed at the time represented by the time bin array index. The third column is the average number of pits per failed waste package at the time represented by the time bin array index. The fourth column is the average number of patches per failed waste package at the time represented by the time bin array index.

Post306 then produces the RIP input table. The total fraction of waste packages that fail is computed. All the nonzero average pit or patch time points are equally grouped into 80 groups and the averages of each of those groups is used to represent the average number of pits or patches.

D4. Description of Test Cases

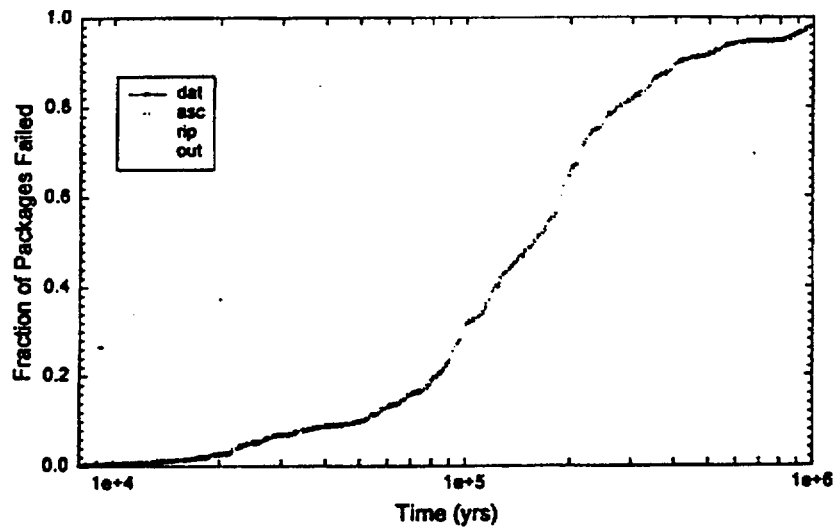
The program results were checked by taking a WAPDEG simulation result and directly comparing the *.out file with the *.dat and *.asc files produced by the postprocessor in an EXCEL spreadsheet. The columns in the *.dat file were directly compared with their corresponding columns in the *.out file. The failure curve in the *.asc file and the RIP table were compared with the failure curve from the *.dat file by a graphical comparison. The average number of pit and patch per package curves from the *.asc file and rip table were compared graphically.

D4.1 Description of Test Results

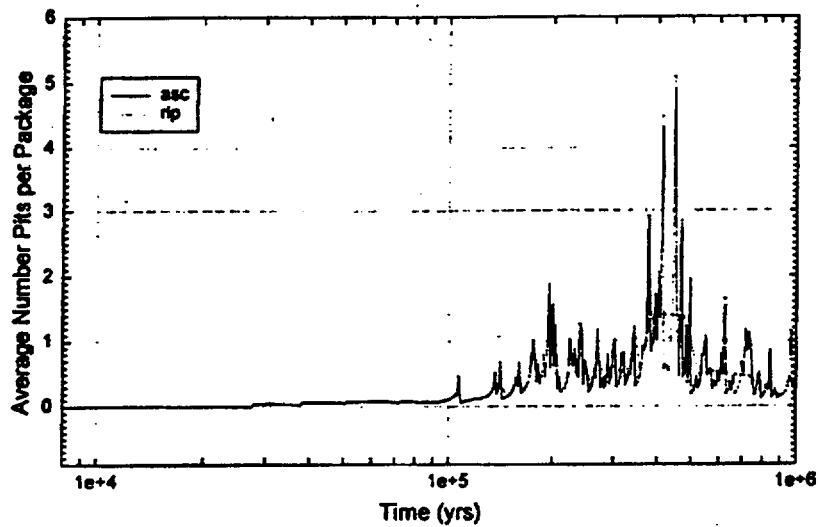
In an EXCEL workbook the *.dat, *.asc, *.out, and RIP files were each imported to a spreadsheet. Then the out and dat files were checked against each other by sorting the out file column and subtracting the corresponding dat column and then checking that the sum was zero. A spreadsheet was made to graph the following results.

There are three graphs for the dripping simulation using Nesfad100mh. One showing all four failure curves, one for the average number of pits from the *.asc and rip table, and one for the average number of patches from the *.asc and rip table.

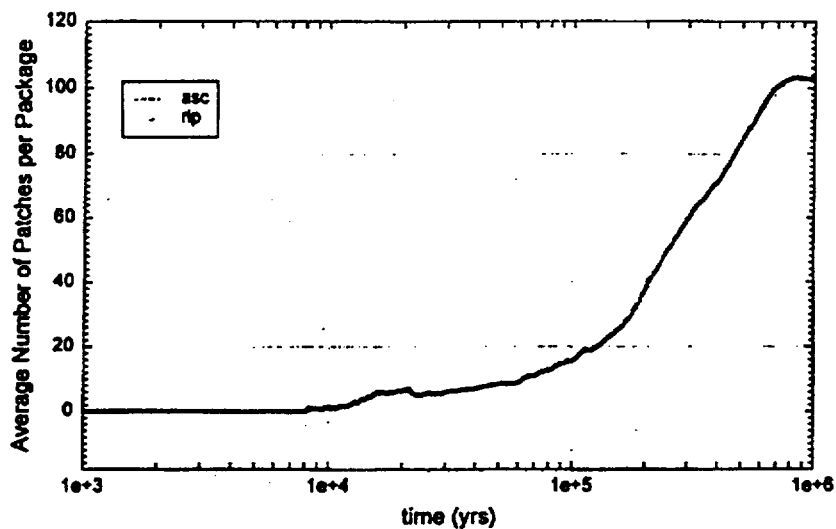
Fraction of Packages Failed vs. time (yrs)
NEsfad100mh



Average Number Pits vs. time (yrs)
NEsfad100mh



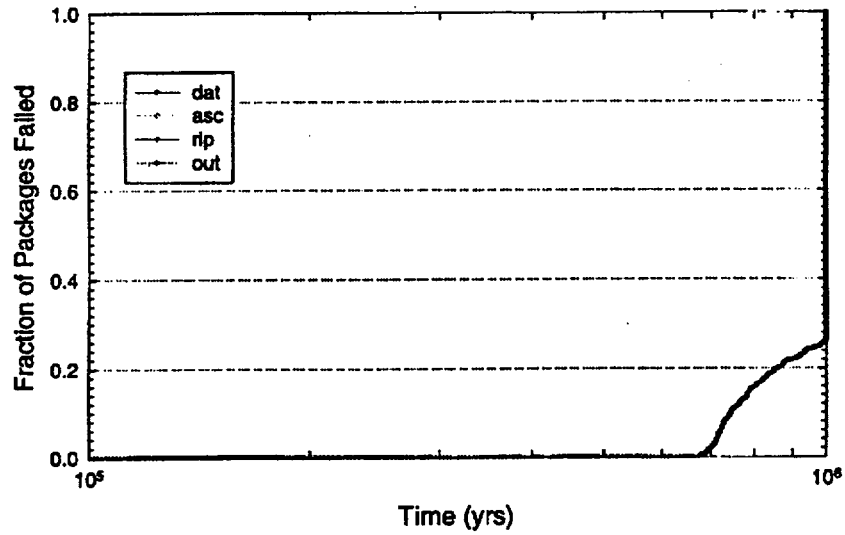
Average Number of Patches vs. time (yrs)
NEsfad100mh



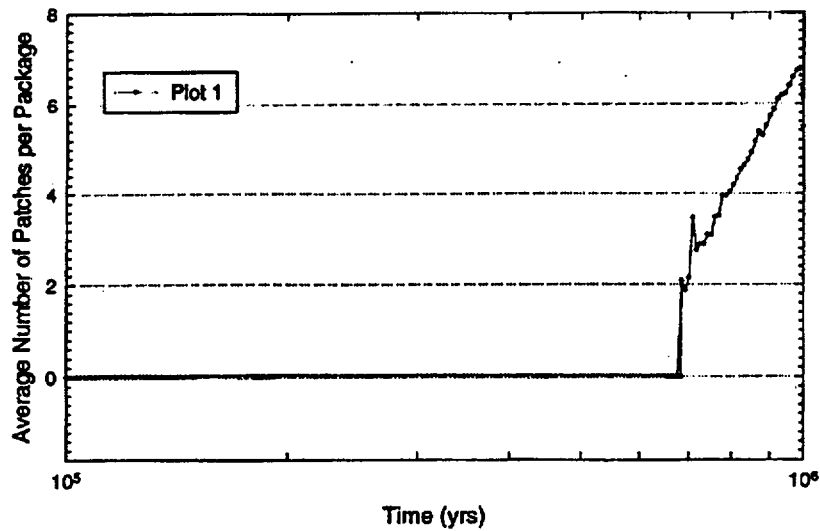
In the 'Fraction of Packages Failed vs. time' curve above, one may notice that the *.asc and RIP table output differs slightly from the *.dat and *.out file output. This is related to the fact that waste package failure was defined by the first breach data rather than the advection or diffusion data in creating these curves.

There are two graphs for the non-dripping simulation using NEsfnd100mh. One showing all four failure curves, and one for the average number of patches from the *.asc and rip table.

Fraction of Packages Failed vs. time (yrs)
NEsfnd100mh



Average Number of Patches vs. time (yrs)
NEsfnd100mh



D4.2 Range of Input Parameter Values Over which Results were Verified

Post306 has been used on a variety of WAPDEG output files in addition to those presented above. The output from each use is exhaustively checked, visually as well as through independent evaluation of RIP outputs.

D4.3 Identification of Limitations on Software Routine or Validity

The only limitations on Post306 arise from the inability of FORTRAN to dynamically assign array bounds, thus the maximum size of the arrays used in Post306 must be assigned beforehand. The maximum size of the sum arrays (sum1pit, sum2pit, sum1pat, sum2pat, etc.) are limited to 1201. Similarly Post306 can be used to post process WAPDEG simulations consisting of 400 waste packages or less and only accepts four pit/patch output times from the *.out file. Header lengths are limited to 24 lines and file names are limited to 72 characters. All of these parameters are easily changeable PARAMETER values in the program.

D5. Reference List of All Documentation Relevant to the Qualification

RIP Theory Manual and User's Guide, 1997. Section 7.5.3, pp. 7-22, June, Golder Associates, Inc.

D6. Listing

PROGRAM post306

```

c
c This program reads the out, bin, and pat files from a WAPDEG run
c and produces data files suitable for graphing and RIP input.
c
  IMPLICIT NONE
  INTEGER NAR0, NAR1, NAR2, NL, HL
  PARAMETER (NAR0 = 1201)
  PARAMETER (NAR1 = 400, NAR2 = 4)
  PARAMETER (NL = 72, HL = 24)
  INTEGER nchar, nbin, cpckg, npckg, mpckg
  INTEGER i, ripk, hcnt
  INTEGER cntpit(NAR1,NAR2), cntpat(NAR1,NAR2)
  INTEGER sumlpit(NAR0), sum2pit(NAR0)
  INTEGER sumlpat(NAR0), sum2pat(NAR0), sum3pat(NAR0)
  INTEGER sum4pat(NAR0), sum5pat(NAR0)
  DOUBLE PRECISION bin0, binf, frac(NAR1)
  DOUBLE PRECISION BinFact, BinTerm
  DOUBLE PRECISION advect(NAR1), diffus(NAR1)
  DOUBLE PRECISION outlst(NAR1), innlst(NAR1)
  DOUBLE PRECISION time(NAR0), wpbc(NAR0)
  DOUBLE PRECISION avgpit(NAR0), avgpat(NAR0)
  DOUBLE PRECISION ript(NAR0), ripf(NAR0)
  DOUBLE PRECISION rippit(NAR0), rippat(NAR0)
  character    flprefix*30, ver*5
  character*34 flout, flbin, flpat, flasc, fldat
  character*1 header(HL,NL)
c
c Initialize arrays
c
  sumlpit = 0
  sum2pit = 0
  sumlpat = 0
  sum2pat = 0
  sum3pat = 0
  sum4pat = 0
  sum5pat = 0
c
c Get the prefix name of the wapdeg file to post process the out,
c bin, and pat files. Create the asc and dat files for graphing data
c
  CALL getflprefix(nchar, flprefix)
  flout = flprefix(1:nchar)//'.out'
  flbin = flprefix(1:nchar)//'.bin'
  flpat = flprefix(1:nchar)//'.pat'
  flasc = flprefix(1:nchar)//'.asc'
  fldat = flprefix(1:nchar)//'.dat'
  OPEN (30, FILE = flasc)
  OPEN (31, FILE = fldat)
c

```

```
c Read the out file and determine how many packages where run (cpckg),
c how many packages were written to the bin file (npckg), and how many
c packages were written to the pat file (mpckg). Also copy the header
c comment information to insert in the RIP table along with the
c wapdeg file name and version number.
c
      CALL readout(NAR1, NL, HL, flout, ver, hdcnt, header,
&                cpckg, npckg, mpckg, bin0, binf, nbin,
&                advect, diffus, outlst, innlst, cntpit, cntpat)
c
c Error check array size
c
      if ( (nbin+1) .gt. NARO) then
        write(*,*) 'Error: array NARO dimension is too small'
        write(*,*) 'Increase and recompile'
        STOP
      end if
c
c Sort each column from the out file.
c
      CALL sortout(NAR1, cpckg,
&                advect, diffus, outlst, innlst, cntpit, cntpat)
c
c Create dat file (cpckg rows by 14 columns)
c Column order: frac, 1-frac, outlst, diffus, advect, innlst,
c pittimes(4), pattimes(4)
c
      do i = 1, cpckg
        frac(i) = dble(i)/dble(cpckg)
        write(31,22) frac(i), 1-frac(i),
& outlst(i), diffus(i), advect(i), innlst(i),
& cntpit(i,1), cntpit(i,2), cntpit(i,3), cntpit(i,4),
& cntpat(i,1), cntpat(i,2), cntpat(i,3), cntpat(i,4)
      end do
c
c Read the bin and pat files and calculate cumulative sums.
c
      CALL readbin(flbin, cpckg, sumlpit, sum2pit)
      CALL readpat(flpat, cpckg,
&                sumlpat, sum2pat, sum3pat, sum4pat, sum5pat)
c
c Interpolate breach number to bin times
c
      if (flprefix(5:6) .eq. 'nd') then
        write(*,*) 'No dripping case'
        CALL interptime(npckg, diffus, bin0, binf, nbin, wpbc)
      else
        write(*,*) 'Dripping case'
        CALL interptime(npckg, advect, bin0, binf, nbin, wpbc)
      end if
c
c Create asc file ((nbin+1) rows by 4 columns)
c Column order: time, wpbc, avgpit, avgpat
c
```

```

BinFact = nbin/log(binf/bin0)
BinTerm = BinFact*log(bin0)
do i = 1, (nbin+1)
  time(i) = exp((dblc(i-1) + BinTerm)/BinFact)
  if (wpbc(i) .eq. 0.D0) then
    avgpit(i) = 0.D0
    avgpat(i) = 0.D0
  else
    avgpit(i) = dblc(sum2pit(i))/wpbc(i)
    avgpat(i) = dblc(sum2pat(i) + sum5pat(i))/wpbc(i)
    wpbc(i) = wpbc(i)/dblc(cpckg)
  end if
  write(30,20) time(i), wpbc(i), avgpit(i), avgpat(i)
end do
c
c Create RIP tables which is an abbreviated version of the asc file.
c RIP tables are required to have 84 or less table entries.
c
  call avgc3(nbin, time, wpbc, avgpit, avgpat,
&          ripk, ript, ripf, rippit, rippat)
  call table3(NL, HL, nchar, flprefix, ver, hdcnt, header,
&          ripk, ript, ripf, rippit, rippat)
  CLOSE(30)
  CLOSE(31)
20  format(1x, 4(f15.5, 1x))
22  format(1x, 2(f12.4, 1x), 4(f12.2, 1x), 8(i10, 1x))
END !PROGRAM post306
c
c*****
c
  SUBROUTINE getflprefix(nchar, flprefix)
c
c  Quiries user for wapdeg out file name.
c  If no file name is specified, the default name is chosen.
c
  IMPLICIT NONE
  INTEGER nchar
  CHARACTER*30 flprefix

  INTEGER count
  LOGICAL outloop, testex
  CHARACTER*1 chl
  CHARACTER*34 outflname

  write(*,*) 'Enter name of out file? '
  read (*, '(A34)') outflname
  if (outflname(1:1) .EQ. '') then
    nchar = 6
    flprefix = 'wap306'
    write (*,*) 'Default file selected: ', flprefix
  else
    outloop = .TRUE.
    count = 0
    do while (outloop)

```

```

count = count + 1
chl = outflname(count:count)
if ((chl .EQ. ' ') .OR. (chl .EQ. '.')) then
  outloop = .FALSE.
else if (count .EQ. 31) then
  write(*,*) 'Input file names longer than 30 characters'
  write(*,*) 'are truncated.'
  outloop = .FALSE.
end if
end do
nchar = count-1
flprefix = outflname(1:nchar)
end if
outflname = flprefix(1:nchar)//'.out'
INQUIRE(FILE = outflname, EXIST = testex)
if (.NOT. testex) then
  write (*,*) 'ERROR - Out file does not exist:',outflname
  write (*,*) '          Try again '
  STOP
else
  write (*,*) 'processing - out file: ',outflname
end if
RETURN
END !SUBROUTINE getflprefix

```

```

c
c*****
c

```

```

SUBROUTINE readout(NAR1, NL, HL, flout, ver, hdcnt, header,
&                  cpckg, npckg, mpckg, bin0, binf, nbin,
&                  advect, diffus, outlst, innlst, cntpit, cntpat)

```

```

c
c Reads the out file to get waste pckg failure times and
c determine how many waste pckgs failed
c

```

```

IMPLICIT NONE
INTEGER NAR1, NL, HL, hdcnt, cpckg, npckg, mpckg, nbin
INTEGER cntpit(NAR1,*), cntpat(NAR1,*)
DOUBLE PRECISION bin0, binf
DOUBLE PRECISION advect(*), diffus(*), outlst(*), innlst(*)
character*34 flout

```

```

INTEGER outfid, i, j
INTEGER idummy
DOUBLE PRECISION ddummy, pat(5)
character dum*80, ver*5
character*1 header(HL,NL)

```

```

outfid = 15
OPEN (outfid, FILE = flout, STATUS = 'OLD')
hdcnt = 1
read (outfid,8000) dum
do j = 1, NL
  header(hdcnt,j) = dum(j:j)
end do

```

```

do while (dum(1:4) .ne. 'STAR')
  hdcnt = hdcnt + 1
  read (outfid,8000) dum
  if (hdcnt .le. HL) then
    do j = 1, NL
      header(hdcnt,j) = dum(j:j)
    end do
  else
    hdcnt = HL
    write(*,*) 'Header cut off at line: ',HL
  end if
end do
read (outfid,8002) ver
write(*,*) 'version: '//ver
c
c Read the number of history files, then read past them
c to the number of packages
c
  read(outfid,*) j
  do i = 1, (2*j + 2)
    read(outfid,8000) dum
  end do
c
c Read the number of packages simulated and the binning parameters
c
  read(outfid,*) cpckg
  read(outfid,*) bin0, binf, nbin
c
c Error check array size
c
  if ( cpckg .gt. NAR1) then
    write(*,*) 'Error: array NAR1 dimension is too small'
    write(*,*) 'Increase and recompile'
    STOP
  end if
c
c Read the out file columns
c
  do while (dum(1:4) .ne. '----')
    read (outfid,8000) dum
  end do
  npckg = 0
  mpckg = 0
  do j = 1, cpckg
    read(outfid, *) idummy, advect(j), diffus(j), ddummy,
&      outlst(j), innlst(j), ddummy, ddummy, ddummy, pat(1),
&      pat(2), pat(3), ddummy, ddummy, pat(4), pat(5),
&      (cntpit(j,i), i = 1, 4), (cntpat(j,i), i = 1, 4)
    if ( innlst(j) .lt. binf ) then
      npckg = npckg + 1
      if ( pat(1) .lt. binf ) then
        mpckg = mpckg + 1
      else if ( pat(2) .lt. binf ) then
        mpckg = mpckg + 1
      end if
    end if
  end do

```

```

        else if ( pat(3) .lt. binf ) then
            mpckg = mpckg + 1
        else if ( pat(4) .lt. binf ) then
            mpckg = mpckg + 1
        else if ( pat(5) .lt. binf ) then
            mpckg = mpckg + 1
        else
            mpckg = mpckg
        end if
    end if
end do
8000 format(A80)
8002 format(A5)
CLOSE(outfid)
RETURN
END !SUBROUTINE readout

```

C

C*****

C

```

SUBROUTINE sortout(NAR1, cpckg,
&          advect, diffus, outlst, innlst, cntpit, cntpat)

```

```

IMPLICIT NONE

```

```

INTEGER NAR1, cpckg

```

```

INTEGER cntpit(NAR1,*), cntpat(NAR1,*)

```

```

DOUBLE PRECISION advect(*), diffus(*), outlst(*), innlst(*)

```

```

INTEGER i

```

```

INTEGER pit1(NAR1), pit2(NAR1), pit3(NAR1), pit4(NAR1)

```

```

INTEGER pat1(NAR1), pat2(NAR1), pat3(NAR1), pat4(NAR1)

```

```

CALL dshellsort1(cpckg, advect)

```

```

CALL dshellsort1(cpckg, diffus)

```

```

CALL dshellsort1(cpckg, outlst)

```

```

CALL dshellsort1(cpckg, innlst)

```

```

do i=1, cpckg

```

```

    pit1(i) = cntpit(i,1)

```

```

    pit2(i) = cntpit(i,2)

```

```

    pit3(i) = cntpit(i,3)

```

```

    pit4(i) = cntpit(i,4)

```

```

    pat1(i) = cntpat(i,1)

```

```

    pat2(i) = cntpat(i,2)

```

```

    pat3(i) = cntpat(i,3)

```

```

    pat4(i) = cntpat(i,4)

```

```

end do

```

```

CALL ishellsort1(cpckg,pit1)

```

```

CALL ishellsort1(cpckg,pit2)

```

```

CALL ishellsort1(cpckg,pit3)

```

```

CALL ishellsort1(cpckg,pit4)

```

```

CALL ishellsort1(cpckg,pat1)

```

```

CALL ishellsort1(cpckg,pat2)

```

```

CALL ishellsort1(cpckg,pat3)

```

```

CALL ishellsort1(cpckg,pat4)

```

```

      do i=1, cpckg
        cntpit(i,1) = pit1(i)
        cntpit(i,2) = pit2(i)
        cntpit(i,3) = pit3(i)
        cntpit(i,4) = pit4(i)
        cntpat(i,1) = pat1(i)
        cntpat(i,2) = pat2(i)
        cntpat(i,3) = pat3(i)
        cntpat(i,4) = pat4(i)
      end do
      RETURN
    END !SUBROUTINE sortout
C
C*****
C
C      SUBROUTINE readbin(flbin, npckg, sumlpit, sum2pit)
C
C      Reads the bin file to get pit history counts
C
C      IMPLICIT NONE
C      INTEGER npckg
C      INTEGER sumlpit(*), sum2pit(*)
C      character*34 flbin
C
C      INTEGER binfid, i, j
C      INTEGER istart, iend, top, bottom
C      INTEGER idum
C
C      open bin file
C
C      binfid = 16
C      OPEN (binfid, FILE = flbin, STATUS = 'OLD')
C
C      For each package in pit file get values and sum
C
C      do i = 1, npckg
C        read(binfid, *) idum, istart, iend
C        do j = istart, iend
C          read(binfid, *) top, bottom
C          sumlpit(j) = sumlpit(j) + top
C          sum2pit(j) = sum2pit(j) + bottom
C        end do
C      end do
C      CLOSE(binfid)
C      RETURN
C      END !SUBROUTINE readbin
C
C*****
C
C      SUBROUTINE readpat(flpat, mpckg,
C      & sumlpit, sum2pit, sum3pat, sum4pat, sum5pat)
C
C      Read pat file and determine patch history and first patch time
C

```

```

      IMPLICIT NONE
      INTEGER mpckg, sum1pat(*)
      INTEGER sum2pat(*), sum3pat(*), sum4pat(*), sum5pat(*)
      character*34 flpat

      INTEGER patfid, i, j
      INTEGER istart, iend, pat1, pat2, pat3, pat4, pat5
      INTEGER idum

c
c  open pat file
c
      patfid = 17
      OPEN (patfid, FILE = flpat, STATUS = 'OLD')
c
c  For each pckgage in pat file get values and sum
c
      do i = 1, mpckg
        read(patfid, *) idum, istart, iend
        do j = istart, iend
          read(patfid, *) pat1, pat2, pat3, pat4, pat5
          sum1pat(j) = sum1pat(j) + pat1
          sum2pat(j) = sum2pat(j) + pat2
          sum3pat(j) = sum3pat(j) + pat3
          sum4pat(j) = sum4pat(j) + pat4
          sum5pat(j) = sum5pat(j) + pat5
        end do
      end do
      CLOSE(patfid)
      RETURN
      END !SUBROUTINE readpat

c
c *****
c
      SUBROUTINE interptime(npckg, wpfc, bin0, binf, nbin, cbin)

      IMPLICIT NONE
      INTEGER npckg, nbin
      DOUBLE PRECISION wpfc(*), bin0, binf, cbin(*)

      INTEGER count(npckg), j, i, ibin
      DOUBLE PRECISION BinFact, BinTerm
      DOUBLE PRECISION ctime(npckg), tb, ta, tbin, lnratio

c
c  Caculate time terms
c
      BinFact = nbin/log(binf/bin0)
      BinTerm = BinFact*log(bin0)

c
c  first reduce wpfc to remove repeat times and create arrays ctime
c  and count which is time versis number of packages
c
      j = 1
      count(1) = 1
      ctime(1) = wpfc(1)

```



```

do i = 2, npckg
  if ( wpfc(i) .ne. wpfc(i-1) ) then
    j = j + 1
    ctime(j) = wpfc(i)
  end if
  count(j) = i
end do

c
c Now use ctime(1:j) and count(1:j) to interpolate values to cbin
c
  i = 2
  tb = ctime(i)
  ta = ctime(i - 1)
  do ibin = 1, (nbin + 1)
    tbin = exp((dble(ibin-1) + BinTerm)/BinFact)
    if (tbin .lt. ctime(1)) then
      cbin(ibin) = 0.00
    else if (tbin .gt. ctime(j)) then
      cbin(ibin) = dble(count(j))
    else
      do while (tb .lt. tbin)
        i = i + 1
        tb = ctime(i)
        ta = ctime(i - 1)
      end do
      lnratio = log(tbin/ta)/log(tb/ta)
      cbin(ibin) = dble(count(i-1)) +
&               dble(count(i) - count(i-1))*lnratio
    end if
  end do
  RETURN
END !SUBROUTINE interptime

c
c*****
c
  SUBROUTINE avgc3(nbin, oldt, oldx, oldy, oldz,
&               nnew, newt, newx, newy, newz)

c
c Reduce the number of values to less than 84
c oldx is the frac of packages
c
  IMPLICIT NONE
  INTEGER nbin, nnew
  DOUBLE PRECISION oldt(*), oldx(*), oldy(*), oldz(*)
  DOUBLE PRECISION newt(*), newx(*), newy(*), newz(*)

  INTEGER i, j, k, n
  DOUBLE PRECISION step, dstep
  DOUBLE PRECISION avgt, avgx, avgy, avgz

c
  newt(1) = 0.00
  newx(1) = 0.00
  newy(1) = 0.00
  newz(1) = 0.00

```

```

i = 1
k = 3
dstep = oldx(nbin)/80.0
step = dstep
do while ((oldx(i) .eq. 0.D0) .and. (i .lt. nbin))
  i = i + 1
end do
newt(2) = oldt(i) - 1
newx(2) = 0.D0
newy(2) = 0.D0
newz(2) = 0.D0
avgt = 0.D0
avgx = 0.D0
avgy = 0.D0
avgz = 0.D0
n = 0
do j = i, nbin
  if ( oldx(j) .ge. step ) then
    newt(k) = (oldt(j) + avgt)/dble(n+1)
    newx(k) = (oldx(j) + avgx)/dble(n+1)
    newy(k) = (oldy(j) + avgy)/dble(n+1)
    newz(k) = (oldz(j) + avgz)/dble(n+1)
    k = k + 1
    step = step + dstep
    avgt = 0.D0
    avgx = 0.D0
    avgy = 0.D0
    avgz = 0.D0
    n = 0
  else
    avgt = avgt + oldt(j)
    avgx = avgx + oldx(j)
    avgy = avgy + oldy(j)
    avgz = avgz + oldz(j)
    n = n + 1
  end if
end do
newt(k) = oldt(nbin + 1)
newx(k) = oldx(nbin + 1)
newy(k) = oldy(nbin + 1)
newz(k) = oldz(nbin + 1)
nnew = k
RETURN
END !SUBROUTINE avgc3

```

```

c
c*****
c
c      SUBROUTINE table3(NL, HL, nchar, flprefix, ver, hdcnt, header,
c      &                  n, t, x, y, z)
c
c      Creates a 3-d RIP table
c
c      IMPLICIT NONE
c      INTEGER NL, HL, nchar, n, hdcnt

```

```

DOUBLE PRECISION t(*), x(*), y(*), z(*)
character flprefix*30, ver*5

INTEGER ntab, d1, d2, i, j, k
character filenm*9, line*NL
character*1 header(HL,NL)

write(*,*) 'Enter table number(00-99):'
read (*,*) ntab
d1 = ntab/10
d2 = ntab - d1*10
filenm = 'table.t'//char(d1+48)//char(d2+48)
write(*,*) 'Creating ',filenm
OPEN (22, file = filenm)
write(22,*) '! From wapdeg file: '//'flprefix(1:nchar)
write(22,*) '! From wapdeg version: '//'ver(1:5)
do i = 1, hdcnt
  do j = 1, NL
    line(j:j) = header(i,j)
  end do
  write(22,*) '! '//'line(1:NL)
end do
write(22,40) 2
write(22,40) 3, n
write(22,40) 1,2,3
c
c This is done so time values do not repeat
c
  k = 1
  write(22,15) t(k)
  do i = 2, n
    if ( t(k) .eq. t(i) ) then
      t(i) = t(i-1) + 1
    else
      k = i
    end if
    write(22,15) t(i)
  end do
  do i = 1, n
    write(22,16) x(i),y(i),z(i)
  end do
  close(22)
15 format (1x, f15.4)
16 format (3(1x, f15.4))
40 format (3(i3,1x))
  RETURN
  END !SUBROUTINE table3
c
c .....
c
  SUBROUTINE dshellsort1(n,arr)
c
c Sorts a dble array arr(1:n) into ascending numerical order.
c

```

```

IMPLICIT NONE
INTEGER n
DOUBLE PRECISION arr(*)

INTEGER gap, i
LOGICAL same
DOUBLE PRECISION hold

gap = n / 2
do while (gap .ge. 1)
  same = .TRUE.
  do i = 1, n - gap
    if (arr(i) .gt. arr(i+gap)) then
      hold = arr(i)
      arr(i) = arr(i+gap)
      arr(i+gap) = hold
      same = .FALSE.
    end if
  end do
  if (same) then
    gap = gap / 2
  end if
end do
RETURN
END !SUBROUTINE dshellsort1

```

```

c
c*****

```

```

c

```

```

SUBROUTINE ishellsort1(n,iarr)

```

```

c

```

```

c Sorts an integer array arr(1:n) into ascending numerical order.

```

```

c

```

```

IMPLICIT NONE
INTEGER n
INTEGER iarr(*)

INTEGER gap, i
LOGICAL same
INTEGER hold

gap = n / 2
do while (gap .ge. 1)
  same = .TRUE.
  do i = 1, n - gap
    if (iarr(i) .gt. iarr(i+gap)) then
      hold = iarr(i)
      iarr(i) = iarr(i+gap)
      iarr(i+gap) = hold
      same = .FALSE.
    end if
  end do
  if (same) then
    gap = gap / 2
  end if
end do

```

```
end do  
RETURN  
END !SUBROUTINE ishellsort1
```

c

c*****