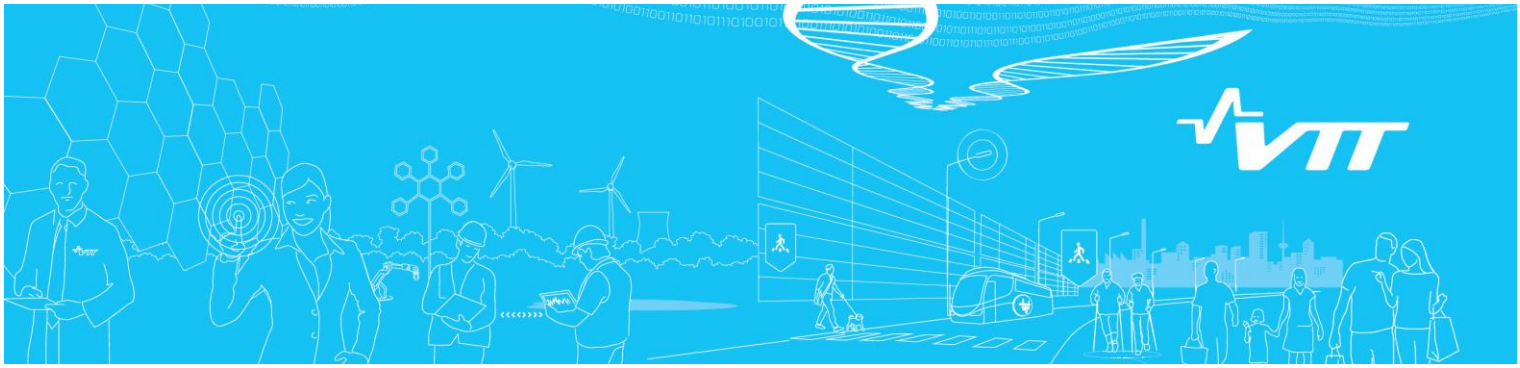
VTT Technical Research Centre of Finland

FINIX - Fuel behavior model and interface for multiphysics applications

Ikonen, Timo; Kättö, Joonas; Loukusa, Henri

Published: 01/01/2015

Document Version
Publisher's final version



RESEARCH REPORT

VTT-R-05887-15

FINIX - Fuel behavior model and interface for multiphysics applications - Code documentation for version 0.15.12

Authors: Timo Ikonen, Joonas Kättö, Henri Loukusa

Confidentiality: Public

Report's title

FINIX - Fuel behavior model and interface for multiphysics applications - Code documentation for version 0.15.12

Customer, contact person, address	Order reference
VYR	SAFIR 15/2015
Project name	**Project number/Short name**
Physics and chemistry of nuclear fuel	102531 / PANCHO
Author(s)	**Pages**
Timo Ikonen, Joonas Kättö, Henri Loukusa	144
Keywords	**Report identification code**
fuel behaviour modelling, FINIX	VTT-R-05887-15

Summary

The FINIX fuel behavior code has been updated to version 0.15.12. In the new version a new cladding mechanical model is implemented which models time-independent plasticity of the cladding. Material properties for Zr1%Nb cladding have also been implemented in this version.

The FINIX code has been designed to provide a fuel behavior module for other simulation codes in multiphysics simulations. The intended use is the improvement of fuel behavior description in neutronics, thermal hydraulics and reactor dynamics codes, without having to employ full-scale fuel performance codes. FINIX couples with the host code on a source code level, and provides an interface of functions that can be used to access the fuel behavior model from the host code. At VTT, FINIX has been integrated into the Monte Carlo reactor physics code Serpent 2, and reactor dynamics codes TRAB-1D, TRAB3D and HEXTRAN.

FINIX consists of several interconnected models that describe the thermo-mechanical behavior of the fuel rod. FINIX solves the transient heat equation, with couplings to the cladding and pellet mechanical behavior through the gap conductance and pressure. Publicly available experimental correlations are used for the material properties, and simple models for the heat transfer from the cladding to the coolant have been included.

FINIX has been verified against the FRAPTRAN fuel performance code in RIA scenarios, and compared against experimental Halden reactor data. Results indicate good agreement with the FRAPTRAN code and experimental measurements. Limitations of the present version in the simulated scenarios have also been identified. Detailed account of the validation results are presented in a separate validation report.

Confidentiality	Public

Espoo 8.12.2015

Written by	Reviewed by	Accepted by
Timo Ikonen Research Scientist	Ville Tulkki Senior Scientist	Timo Vanttola Head of Research Area

VTT's contact address

P.O.Box 1000, 02044 VTT, Espoo

Distribution (customer and VTT)

SAFIR2018/RG3
VTT Archive

Contents

1 Introduction

The assessment of the performance of a fuel rod in the reactor core is an integral part of the design, operation, and safety analysis of the nuclear reactor. To study the behavior of the fuel rod, one typically resorts to using a model in one of the two extremes. On one end are the dedicated fuel performance codes, which take into account the multitude of physical phenomena involved in the thermo-mechanical behavior of the fuel rod: diffusion of heat, elastic and plastic deformation of the pellet and the cladding, release of gaseous fission products into the free volume, the interplay of the gas pressure with the mechanical solution of the pellet and the cladding, the feed-back of the deformations and temperatures to the gap heat conductance, the effect of the cladding surface heat flux to the heat transfer into the surrounding coolant, and so on. All of this is done with a complex, interconnected model, where experimental correlations are used to model the dependencies of the material properties on temperature, pressure, burn-up, etc. On the other end of the spectrum are the models used within, *e.g.*, many thermal-hydraulics or neutronics codes, which are based on simple correlations, non-mechanical thermal elements, or even fixed values of temperature. Although they are quick to understand and efficient to solve, such fuel models may be less-than-realistic in, for instance, transient conditions or, in cases where fuel with extended burn-up should be considered.

The purpose of this work is to develop a fuel performance model to be used in a multiphysics context, allowing it to be coupled to existing thermal-hydraulics, reactor dynamics or neutronics codes used at VTT, such as TRAB, HEXTRAN and SERPENT. The scope of the FINIX code is somewhere between the full-fledged fuel performance codes and the simple thermal element: although FINIX employs many of the same experimental correlations as the full fuel performance codes, and solves the thermal and mechanical behavior of the rod, several simplifications have been made, both to improve the performance of the code, and to expedite its development. These assumptions and approximations are discussed in Section 2.2.

In the first stage of development, the aim has been to develop a model that is capable of solving the transient heat equation, with couplings to the cladding and pellet mechanical behavior through the gap conductance and pressure. Experimental correlations are used for the material properties, and simple models for the heat transfer from the cladding to the coolant have been included. The latter can also be easily replaced by the coupling to a thermal-hydraulics code. The physical models, correlations and their numerical implementation is described in Sections 3–6.

The FINIX code has been designed so that it can be coupled on a source-code level, so that passing input and output files between the codes is not necessary. FINIX includes a collection of built-in functions that can be used for basic setup of the system, and for running the actual simulations, using a fairly high-level syntax. In addition, FINIX has an error message system that can be used to detect beyond-normal operation of the code without aborting program execution. The usage of this high-level interface is described in Section 7. In addition, because of the direct coupling on a source-code level, FINIX allows for low-level (detailed) control of its input and output. Mainly for this purpose, the detailed code description is included in the Appendix of this document. Since FINIX-0.15.6, input and output files for stand-alone usage have been defined and examples are provided with the code.

Assessment of the FINIX-0.13.9 code without external coupling is presented in a separate report [1], with a summary of the results given in Section 8 of this report. FINIX is compared with the FRAPTRAN fuel performance code in several RIA scenarios, and with experimental

Halden reactor data. The results show good agreement both with FRAPTRAN and the experiments. Known limitations of FINIX are also discussed in Section 8 and in the validation report [1]. Validation of FINIX-0.15.12 is to be performed in the future.

2 General model description

2.1 Version

2.1.1 Current version

This document describes version 0.15.12 of the FINIX fuel behavior model. The version number follows the convention where the first number identifies the general stage of development ("0" for early development stage) and the following numbers identify the date of release (year followed by month).

2.1.2 Version history

The following versions of FINIX have been released:

FINIX-0.13.1 (January 2013, reported in Ref. [2]).

FINIX-0.13.9 (September 2013, reported in Ref. [3]).

FINIX-0.15.6 (June 2015, reported in Ref. [4]).

FINIX-0.15.12 (December 2015, this report)

2.2 Model assumptions and approximations

2.2.1 Geometry

The FINIX model solves the heat equation and the cladding mechanical behavior in cylindrical geometry. Furthermore, the heat equation is solved in one dimension, with the temperature having dependence only on the radial coordinate r. The azimuthal (θ-) dependence is completely neglected (implying also the assumption that the central axes of the pellet and the cladding are the same), and the axial (z-) dependence is only included by solving the heat equation independently for several axial slices, or nodes. However, there is no heat flux between the neighboring axial nodes. Therefore, the model is only applicable to scenarios where the axial heat transfer is small compared to the radial heat transfer, and where the boundary conditions and the power distribution are symmetric with respect to the azimuthal rotations.

The rod internal pressure is calculated by taking into account the deformations and temperatures of all axial nodes, and is assumed equal throughout the axial length of the rod. Coupled with

the one dimensional treatment of the heat equation, this constitutes the so-called $1\frac{1}{2}$-dimensional model.

2.2.2 Fuel pellet

The fuel pellet is assumed mechanically rigid, so that it has no response to external stresses. In addition, a number of phenomena that become important in extended operation and at high burn-up have been left out of the model. For instance, accumulation of fuel swelling due to irradiation, densification, and high burn-up structure is not included in the model. However, they can be taken into account effectively through parametrization of the pellet and cladding dimensions from irradiated rods. On the other hand, the effect of accumulated burn-up to material property correlations is taken into account, where appropriate.

The pellet is assumed to be perfectly cylindrical for the purposes of the solution of the heat equation – specifically, dishing, chamfers or hourglassing is not taken into account. In calculating the axial strain, the mechanical connection between consecutive pellets is assumed to occur at the edges – a more realistic modeling of the pellet shape remains to be implemented.

As of version 0.13.9, FINIX supports externally given fuel swelling, densification and relocation strains. Radial pellet relocation can also be calculated from correlations.

2.2.3 Cladding

The mechanical model of the cladding is based on the thick cylindrical shell approximation. The model assumes, for example, that the radial differences in stresses and temperatures across the cladding are small. In addition, the model is valid only when the axial curvature of the cladding is small, *i.e.*, when there is very little bowing or bending of the cladding.

The cladding mechanical response is assumed to elasto-plastic. As of version 0.13.9, plastic strains can be externally given to FINIX, and as of version 0.15.12, time-independent plastic deformation is modeled in accordance with the infinitesimal strain theory, but plastic deformations due to creep are not modeled. The model is inadequate to model transient ruptures as no failure models are implemented. Ballooning or other large deformations may not be correctly modeled, as these are incorrectly described by the infinitesimal strain theory. Also, the model is not designed for modeling long periods of steady state operation, due to the missing creep model.

Oxide formation of the cladding is not modeled, although the effect of the oxide layer is included in the material correlations, where appropriate.

2.2.4 Fill gas and FGR

Since version 0.13.9, material correlations for helium, argon, krypton, xenon, hydrogen, nitrogen and water vapor have been added. The amount of fill gas and fractions of individual species can be given as input, although release of fission gases is not currently modeled. Material changes in the pellet due to accumulating fission products, or the release of the said products into the gas gap is therefore not taken into account.

2.3 Changes from previous versions

2.3.1 Version 0.15.12

Time-independent plasticity of the cladding is modeled in FINIX version 0.15.12. The radial return method is implemented to solve the plasticity equations. The plastic deformation model is described in detail in section 4.3.3. PNNL stress-strain correlation is used to calculate yield stress.

A new data structure Cylindrical was implemented for storing results in each component in cylindrical coordinates. This is used to store various values of stresses and strains, for example.

The pellet axial strain is now taken from the pellet centerline, as this yields the maximum value of axial strain.

Material correlations for Zr1%Nb cladding used in VVER reactors are now implemented. Material-specific correlations for thermal conductivity, heat capacity, Young's modulus, Meyer's hardness, axial and diametral thermal expansion and yield stress are implemented.

Minor bug was fixed in the cladding mechanical model which caused the radial discretization to be converted to constant-distance scheme in the cladding.

A bug in the material density calculation was fixed. Previously the density was underestimated and this affected also the temperature calculations. As a result of the fix, predicted fuel centerline temperatures are higher by approximately 1 % – 7 %.

2.3.2 Version 0.15.6

This chapter introduces the changes from version 0.13.9 to version 0.15.6.

FINIX data structures have been completely redesigned. All the data FINIX requires or calculates is now stored in five structures. These structures hold the data describing the fuel rod, rod boundary conditions, FINIX simulation options, FINIX simulation results, and the data describing the simulated scenario. The old data arrays described in code documentation for version 0.13.9 are no longer available.

Version 0.15.6 introduces the ability to read input from FINIX and FRAPTRAN input files. Input files are still not necessarily needed in a coupled system, where the host code provides the boundary conditions for FINIX. However, the input files contain several templates that can be used as a starting point even in coupled-code simulations, if all necessary fuel performance input data is not available. If FINIX is run as a stand-alone code, the fuel rod characteristics, boundary conditions and model options should be provided through input files. FINIX no longer contains hard coded input.

FINIX version 0.15.6 allows the user to print new types of output files. These files include node-specific output files, a summary file, and a file showing the contents of all FINIX data structures.

Some bug fixes and minor changes have been done in version 0.15.6. An error in the cladding hoop stress equation (Eq. (46)) was corrected. The effect of this correction is described in section 8. Also, FINIX did not use the fast neutron fluence from the FRAPCON restart file previously, but

this value is now used by FINIX.

2.3.3 Version 0.13.9

Version 0.13.9 incorporates the following changes from version 0.13.1.

In the interface, power density array has been replaced with linear power (per axial node) and radial power distribution (separately for each axial node) arrays. This change was necessary to conserve linear power in the axial node with the introduction of the relocation model. Because the relocation model can drastically change the dimensions of the pellet, using just a fixed power density does not conserve linear power over one time step.

Also in the interface, the params array was split into two arrays. The first one (params) only contains values that are not updated by FINIX. The second one (sresults, for scalar results) contains values that can be updated by FINIX. The previous results array was also renamed vresults (vector form results, for each axial node).

Radial relocation of the pellet is now modeled.

Several quantities have been added to improve FINIX's simulation capabilities at accumulated burnup. These include pellet swelling and densification strains, cladding plastic strains, calculation of pressure from moles of gas instead of fill pressure, possibility to use He, Ar, Kr, Xe, H_2, N_2 and H_2O as fill gas. In FINIX 0.13.9 these values are not updated internally by FINIX, but can be given by the user to initialize FINIX of accumulated burnup.

Rod internal pressure calculation is now based on the gas molar content instead of fill pressure. The change was made to allow changes in the amount of fill gas that will be necessary when fission gas release models are added to FINIX. The fill pressure is still given as input.

Gap conductance correlation has been updated to include the above mentioned fill gases. Also, an option has been added to switch between FRAPCON and FRAPTRAN implementations of the gap conductance correlation. The FRAPTRAN correlation is used as a default (see Section 8).

A module for reading FRAPCON/FRAPTRAN restart files has been added. This can be used to initialize FINIX for accumulated burnup by using FRAPCON to provide the data from steady state irradiation.

A FINIX database of simulation and rod data used in the FINIX-0.13.9 validation [1] was created. The database is currently written inside the source code in files *database.c* and *db_functions.c*. The default rod parameters used in system setup that used to be in *defaults.c* in FINIX-0.13.9 were also moved to *database.c*.

The stability of the numerical iteration of the gap conductance in the transient and initial state solvers has been improved. The transient solver now searches for upper and lower bounds for the solution and, once found, switches to the Dekker method [5] from the secant method.

Default nodalization of the radial nodes has been changed to equal volume rings from equal radius rings.

Several auxiliary functions have been added, including functions to calculate burnup from power

history, density of the fuel and cladding, averages over cross sectional areas, and so on.

Small bug fixes, including the correlations for cladding Meyer's hardness, cladding thermal conductivity, cladding diametral thermal strain.

2.4 FINIX publications

In addition to the VTT reports, the following work related to FINIX development has been published:

2015

- E. Syrjälahti and V. Valtavirta and J. Kättö and H. Loukusa and T. Ikonen and J. Leppänen and V. Tulkki, *Multiphysics simulations of fast transients in VVER-1000 and VVER-440 reactors* [6]. (FINIX-0.15.6)
- T. Ikonen, E. Syrjälahti, V. Valtavirta, H. Loukusa, J. Leppänen and V. Tulkki, *Multiphysics simulation of fast transients with the FINIX fuel behaviour module* [7]. Description of FINIX and applications in coupled-code calculations with Serpent 2, TRAB-1D and TRAB3D/SMABRE. (FINIX-0.13.9)
- T. Ikonen, H. Loukusa, E. Syrjälahti, V. Valtavirta, J. Leppänen and V. Tulkki, *Module for thermomechanical modeling of LWR fuel in multiphysics simulations* [8]. Description of FINIX and applications in coupled-code calculations with Serpent 2, TRAB-1D and TRAB3D/SMABRE. (FINIX-0.13.9)

2014

- V. Valtavirta, T. Ikonen, T. Viitanen, J. Leppänen, *Simulating fast transients with fuel behavior feedback using the Serpent 2 Monte Carlo code* [9]. Application of FINIX and Serpent 2 in simulating self-consistently solved temperature and power in a prompt supercritical pin-cell. (FINIX-0.13.9)

2013

- T. Ikonen, V. Tulkki, E. Syrjähti, V. Valtavirta and J. Leppänen, *FINIX – Fuel Behavior Model and Interface for Multiphysics Applications* [10]. Brief description of the FINIX code, its purpose as a universal fuel behavior module in multiphysics applications and first results. (FINIX-0.13.1)

3 Thermal model

3.1 The heat equation

The conduction of heat is described by the heat equation, where the temperature T in general is a function of time t and all three spacial coordinates. In a cylindrical fuel rod, a convenient choice for the coordinate system are the cylindrical coordinates, r, θ and z. In the present case, however,

we make the simplifying assumption that within each axial slice, T has no dependence on z (by assuming the axial heat transfer to be negligible) and no dependence on θ (by assumed symmetry). With these assumptions, the heat equation takes the form

$$C_V(T)\frac{\partial T}{\partial t} - \frac{1}{r}\frac{\partial}{\partial r}\left[\lambda(T) r \frac{\partial T}{\partial r}\right] - s(r) = 0. \tag{1}$$

Here C_V is the volumetric heat capacity, λ the thermal conductivity and s the source term (thermal power line density). The temperature is a function of time t and the radial coordinate r, $T = T(r,t)$. The solution of Eq. (1) is obtained in the fuel pellet and in the cladding by discretizing the equation with the finite element method (FEM) and by solving the system numerically (see Sec. 6.2). The outer surface of the pellet and the inner surface of the cladding are subject to heat transfer boundary conditions

$$q(R_f) = -\lambda(T(r))\frac{\partial T}{\partial r}\bigg|_{r=R_f} = h\left[T(R_f) - T(R_{ci})\right], \tag{2}$$

where $q(r)$ is the heat flux, R_f is the fuel outer radius, R_{ci} the cladding inner radius and the notation $|_{r=R_f}$ denotes evaluation of the preceding expression at $r = R_f$. The gap conductance h is calculated using the model described in Sec. 3.3. The gap is assumed to have negligible heat capacity, and thus the conservation of energy implies that the heat flux across the inner surface of the cladding is given by

$$q(R_{ci}) = \frac{R_f}{R_{ci}} q(R_f). \tag{3}$$

At the outer surface of the cladding, the boundary condition can be set as constant temperature, constant heat flux or by using a heat transfer coefficient. A more thorough explanation is given in Sec. 6.2, where the numerical solution of the heat equation is explained. The remaining boundary condition is the zero heat flux at the inner surface of the pellet (at R_0),

$$q(0) = 0 \Leftrightarrow \frac{\partial T}{\partial r}\bigg|_{r=R_0} = 0. \tag{4}$$

3.2 Plenum temperature

The model for the plenum gas temperature is derived by assuming that the gas within the plenum is well mixed and described by a single temperature, T_{plen}. The gas exchanges heat with the surrounding walls, whose temperatures are taken as given. Furthermore, it is assumed that the heat capacity of the plenum gas is so small that one can neglect the term with the time derivative of T. One therefore has the steady-state heat equation for the plenum gas,

$$A_p h_p (T_p - T_{\text{plen}}) + A_c h_c (T_c - T_{\text{plen}}) = 0, \tag{5}$$

which gives

$$T_{\text{plen}} = \frac{A_p h_p T_p + A_c h_c T_c}{A_p h_p + A_c h_c} \tag{6}$$

for the temperature of the plenum. Here $A_p(A_c)$ is the area of the end of the fuel pellet (cladding inner surface) facing the plenum, $T_p(T_c)$ the temperature of the pellet (cladding), and $h_p(h_c)$ the heat transfer coefficient between the pellet (cladding) and the plenum gas. The areas are given by

$A_p = \pi R_f^2$ and $A_c = \pi R_{ci}^2 + 2\pi l_p R_{ci}$, where l_p is the (axial) length of the plenum. The temperature of the end of the pellet is calculated as an area-weighted mean temperature (cf. Sec. 6.2). Since the cooling of the end of the pellet due to heat flux into the plenum is not taken into account in the 1D heat equation, this leads to slight over-estimation of the temperature T_p. The temperature of the cladding is assumed to be equal to the coolant temperature, which in turn leads to slight under-estimation of the temperature T_c. The uncertainties introduced in these approximation are not too severe, since the plenum affects the thermo-mechanical solution of the fuel rod only through its coupling to the gap pressure, not affecting the temperatures directly. In addition, solving the surface temperatures accurately would require 2D solution of the heat equations within the plenum, which would lead to undesirable increase in the computational intensity of the model.

The heat transfer coefficients are solved with a similar method as the one used in FRAPTRAN [11]. For the pellet-plenum heat transfer, the coefficient is

$$h_p = \begin{cases} 0.27\lambda_{\text{plen}} \frac{(GrPr)^{0.25}}{2R_{ci}}, & \text{for } Gr < 0, \\ 0.54\lambda_{\text{plen}} \frac{(GrPr)^{0.25}}{2R_{ci}}, & \text{for } 0 \leq Gr \leq 2 \cdot 10^7, \\ 0.14\lambda_{\text{plen}} \frac{(GrPr)^{0.33}}{2R_{ci}}, & \text{for } Gr > 2 \cdot 10^7, \end{cases} \tag{7}$$

where λ_{plen} is the thermal conductivity of the plenum gas, Pr is the Prandtl number and

$$Gr = \frac{g\,(T_p/T_{\text{plen}} - 1)\,(2R_{ci})^3}{\nu^2} \tag{8}$$

is the Grashof number. In the latter, g is the gravitational acceleration and ν is the kinematic viscosity of the plenum gas.

For the cladding-plenum heat transfer coefficient, the corresponding equations are

$$h_p = \begin{cases} 0.55\lambda_{\text{plen}} \frac{(GrPr)^{0.25}}{l_p}, & \text{for } Gr \leq 1 \cdot 10^9, \\ 0.021\lambda_{\text{plen}} \frac{(GrPr)^{0.4}}{l_p}, & \text{for } Gr > 1 \cdot 10^9, \end{cases} \tag{9}$$

with

$$Gr = \frac{g\,(T_c/T_{\text{plen}} - 1)\,l_p^3}{\nu^2}. \tag{10}$$

The correlations for the Prandtl number and the kinematic viscosity are given in Sec. 5.3. The value for the heat conductivity λ_{plen} is not needed, as it is canceled from Eq. (6).

3.3 Gas gap conductance

The heat transfer in the gas gap is modelled with Eq. (2), with the heat transfer coefficient h given as a sum of three terms:

$$h = h_{\text{cond}} + h_{\text{rad}} + h_{\text{contact}}. \tag{11}$$

The first term corresponds to heat conduction across the gap, the second to radiation heat transfer between the pellet surface and the cladding inner surface, and the last one to heat transfer due to solid-solid contact of the pellet and the cladding. When the gas gap remains open, the last term is zero.

3.3.1 Conduction through the gas

The term h_{cond} can be calculated from the heat equation in the gas gap. Assuming the heat capacity of the gas to be small and noting that there is no heat produced in the gap, the equation reduces to $\frac{\partial}{\partial r}\left(\lambda_{\mathrm{gap}}(T) r \frac{\partial T}{\partial r}\right) = 0$. Integrating once with respect to r and applying $\lambda_{\mathrm{gap}} \frac{\partial T}{\partial r}\big|_{r=R_f} = h_{\mathrm{cond}}(T_{ci} - T_f)$ gives

$$\lambda_{\mathrm{gap}}(T) dT = h_{\mathrm{cond}}(T_{ci} - T_f) \frac{dr}{r}. \tag{12}$$

By integrating from T_f to T_{ci} and replacing the temperature-dependent heat conductivity with the average $\bar{\lambda}(T_{ci}, T_f)$ as $\int_{T_f}^{T_{ci}} \lambda_{\mathrm{gap}}(T) dT \equiv \bar{\lambda}(T_{ci}, T_f)(T_{ci} - T_f)$, one obtains

$$h_{\mathrm{cond}} = \frac{\bar{\lambda}(T_{ci}, T_f)}{R_f \ln(1 + d/R_f)}, \tag{13}$$

where $d = R_{ci} - R_f$ is the gap width. Since $d \ll R_f$, the term $\ln(1 + d/R_f) \approx d/R_f$, which gives the form used in FRAPTRAN and FRAPCON:

$$h_{\mathrm{cond}} = \frac{\bar{\lambda}(T_{ci}, T_f)}{d}. \tag{14}$$

In practice, the average $\bar{\lambda}(T_{ci}, T_f)$ can be approximated by taking the average of the gap temperature, instead of the heat conductivity: $\bar{\lambda}(T_{ci}, T_f) \approx \lambda_{\mathrm{gap}}(T_{\mathrm{gap}})$. The introduced error is of the order of a few percent, at most.

In FRAPTRAN and FRAPCON, an effective gap width d_{eff} is used instead of the bare d. The same approach is taken also here. The effective gap width is given by the FRAPCON correlation [12]

$$d_{\mathrm{eff}} = e^{-0.00125 P_{\mathrm{contact}}}(\rho_f + \rho_c) + 1.8(g_f + g_c) - b + d, \tag{15}$$

where P_{contact} is the contact pressure between the pellet and the cladding (in kg/cm^2, see Sec. 4.3.2), $\rho_f(\rho_c)$ is the surface roughness of the pellet (cladding) in meters, $g_f(g_c)$ is the temperature jump distance (in meters) at the pellet (cladding) surface. The constant $b = 1.397 \cdot 10^{-6}$ m. The sum of the temperature jump distances is calculated from

$$g_f + g_c = 0.7816 \left(\frac{\lambda\sqrt{T}}{P}\right)\left(\frac{1}{\sum_i a_i f_i M_i^{-1/2}}\right), \tag{16}$$

where λ, T and P are the thermal conductivity, temperature and pressure of the gas in units of W/mK, K and Pa, respectively, and a_i, f_i and M_i are the thermal accommodation coefficients, mole fractions and molecular weights of the gas constituents.

FINIX also has an option to use the FRAPTRAN correlation for the effective gap width. In this case, the gap width is given by

$$d_{\mathrm{eff}} = e^{-0.00125 P_{\mathrm{contact}}}(\rho_f + \rho_c) + 0.0316(g_f + g_c) + d \quad \text{(optional)}. \tag{17}$$

The heat transfer coefficient h_{cond} is then given as

$$h_{\mathrm{cond}} = \frac{\lambda_{\mathrm{eff}}}{d_{\mathrm{eff}}}, \tag{18}$$

where d_{eff} is given by Eq. (15) and λ_{eff} is calculated for the gas mixture of n species as [13]

$$\lambda_{\text{eff}} = \sum_{i}^{n} \frac{\lambda_i x_i}{x_i + \sum_j (1-\delta_{ij})\,\Psi_{ij} x_j} \tag{19}$$

Here λ_i is the heat conductivity and x_i the mole fraction of the species i, δ_{ij} the Kronecker delta and

$$\Psi_{ij} = \psi_{ij}\left(1 + 2.41\frac{(M_i - M_j)(M_i - 0.142M_j)}{(M_i + M_j)^2}\right), \tag{20}$$

$$\psi_{ij} = \frac{\left[1 + (\lambda_i/\lambda_j)^{1/2}(M_i/M_j)^{1/4}\right]^2}{2^{3/2}\,(1 + M_i/M_j)^{1/2}}, \tag{21}$$

where M_i is the molecular weight of the species.

The heat conductivity of the gases (with the exception of steam) is of the form

$$\lambda_i = A_i T^{B_i}, \tag{22}$$

with the coefficients given in Table 1.

Table 1. *The gas conductivity constants of Eq. (22).*

Species	A	B
He	$2.531 \cdot 10^{-3}$	0.7146
Ar	$4.092 \cdot 10^{-4}$	0.6748
Kr	$1.966 \cdot 10^{-4}$	0.7006
Xe	$9.825 \cdot 10^{-5}$	0.7334
H_2	$1.349 \cdot 10^{-4}$	0.8408
N_2	$2.984 \cdot 10^{-4}$	0.7799

For the water vapour (steam), the heat conductivity is given by [13]

$$\lambda_{H_2O} = 4.44 \cdot 10^{-6} T^{1.45} + 9.45 \cdot 10^{-5}(2.1668P/T)^{1.3} \quad (\text{for } T > 973.15 \text{ K}), \tag{23}$$

$$\begin{aligned}\lambda_{H_2O} =& P/T\left(-2.851 \cdot 10^{-8} + 9.424 \cdot 10^{-10} T - 6.005 \cdot 10^{-14} T^2\right) \\ &+ 1.009\,(P/T)^2 / (T - 273.15)^{4.2} + 17.6 \cdot 10^{-3} \\ &+ 5.87 \cdot 10^{-5}\,(T - 273.15) + 1.08 \cdot 10^{-7}\,(T - 273.15)^2 \\ &- 4.5 \cdot 10^{-11}\,(T - 273.15)^3 \quad (\text{for } T \le 973.15 \text{ K}).\end{aligned} \tag{24}$$

Here T is in Kelvin and P in Pascal.

3.3.2 Radiation across the gap

The radiation heat transfer coefficient is given by the gray body radiation formula

$$h_{\text{rad}} = \frac{\sigma_{SB}}{\frac{1}{\epsilon_f} + \frac{R_f}{R_{ci}}\left(\frac{1}{\epsilon_c} - 1\right)} \frac{T_f^4 - T_{ci}^4}{T_f - T_{ci}}, \tag{25}$$

where σ_{SB} is the Stefan-Boltzmann constant and $\epsilon_f(\epsilon_c)$ is the emissivity of the fuel outer surface (cladding inner surface). The emissivities used in FINIX are

$$\epsilon_f = 0.78557 + 1.5263 \cdot 10^{-5} T_f, \tag{26}$$

where the temperature is in Kelvin, and

$$\epsilon_c = 0.809. \tag{27}$$

The correlation for ϵ_f is the same as in FRAPTRAN, while the value for ϵ_c is the same as in the FEMAXI code [14]. The latter was chosen over the FRAPTRAN correlation, which requires knowledge of the cladding oxide thickness, and currently FINIX has no model for cladding oxidation. In FRAPTRAN, the value $\epsilon_c = 0.809$ would correspond to an effective oxide thickness of approximately 3.9 microns.

3.3.3 Contact between the pellet and cladding

The contact heat transfer coefficient h_{contact} is given by

$$h_{\text{contact}} = \begin{cases} 13.740 \dfrac{\lambda_m P_{\text{rel}}^{1/2}}{\rho_f^{-0.528}\sqrt{\rho_f^2+\rho_c^2}}, & \text{for } P_{\text{rel}} \leq 9 \cdot 10^{-6}, \\ 0.041226 \dfrac{\lambda_m}{\rho_f^{-0.528}\sqrt{\rho_f^2+\rho_c^2}}, & \text{for } 9 \cdot 10^{-6} < P_{\text{rel}} \leq 0.003, \\ 4579.5 \dfrac{\lambda_m P_{\text{rel}}^2}{\rho_f^{-0.528}\sqrt{\rho_f^2+\rho_c^2}}, & \text{for } 0.003 < P_{\text{rel}} \leq 0.0087, \\ 39.846 \dfrac{\lambda_m P_{\text{rel}}}{\rho_f^{-0.528}\sqrt{\rho_f^2+\rho_c^2}}, & \text{for } P_{\text{rel}} > 0.0087, \end{cases} \tag{28}$$

where $P_{\text{rel}} = P_{\text{contact}}/H$ is the relative ratio of the contact pressure and Meyer's hardness H_M of the cladding (see Sec. 5.2.4). In addition, $\lambda_m = 2\lambda_f\lambda_c/(\lambda_f + \lambda_c)$ is the geometric mean of the fuel thermal conductivity λ_f and the cladding thermal conductivity λ_c. The correlation of Eq. (28) is the same as in FRAPTRAN, with the numerical constants merged into one.

4 Mechanical model

4.1 Internal pressure

The internal pressure P of the rod is calculated from the ideal gas equation of state, $PV = nRT$, where V is the gas volume, n the amount of substance of the gas, R the ideal gas constant and T the temperature. Assuming that pressure differences between the gap, plenum and central hole equalize immediately, the internal pressure is given by

$$P = \frac{nR}{V_{\text{plen}}/T_{\text{plen}} + \sum_k \left(V_{\text{cent},k}/T_{\text{cent},k} + V_{\text{gap},k}/T_{\text{gap},k}\right)}. \tag{29}$$

Here the plenum volume is

$$V_{\text{plen}} = \pi R_{ci}^2 l_p, \tag{30}$$

the central hole volume

$$V_{\mathrm{cent},k} = \pi R_{0,k}^2 l_{f,k}, \tag{31}$$

and the gap volume

$$V_{\mathrm{gap},k} = \pi l_{f,k} \left(R_{ci,k}^2 - R_{f,k}^2 \right), \tag{32}$$

where k is the index of the axial slice and $l_{f,k}$ is the axial length of the fuel and $R_{0,k}$, $R_{f,k}$ and $R_{ci,k}$ are the pellet inner radius, pellet outer radius and cladding inner radius, respectively, of the k:th axial slice.

4.2 Pellet mechanical model

The fuel pellet is assumed to have an infinite elastic modulus and no stress-induced deformations (the so-called rigid pellet model [11]). Thermal strain and radial relocation of the fuel is taken into account with correlations. Densification and swelling can be given as input, but are not updated in FINIX-0.13.9. The correlation between the thermal strain and temperature is presented in Sec. 5.1.3, and pellet relocation correlation in Sec. 5.1.4. The radial displacement of the pellet outer surface is calculated by integrating the strains over the pellet radius. The axial strain is calculated using the temperature on the center of the pellet for the thermal strain, where the strain is largest. The details of the thermal strain calculation are explained in Sec. 6.4.1. Swelling and densification strains are also included for the axial strain, but pellet relocation strain is neglected.

4.3 Cladding mechanical model

The mechanical model of the cladding is similar to the FRACAS-I model used in both FRAPCON and FRAPTRAN [11, 12], although some further simplifications have been made. For example, only time-independent plasticity of the cladding is modeled.

The cladding mechanical model is further divided into two distinct situations. The first model considers the case when the gap remains open, and the mechanical equilibrium is determined simultaneously with the calculation of the internal pressure. The second model is invoked when the gap is closed, and the equilibrium is determined by a no-slip condition between the pellet and the cladding. The calculation of the plastic strain increment proceeds similarly in the open gap and closed gap cases.

4.3.1 Open gap model

First, we consider the open gap mechanical model. In this case, the displacement of the cladding inner surface depends on the internal pressure, which in turn is a function of the gap volume. Therefore, the mechanical equilibrium has to be determined simultaneously with the internal pressure calculation. In practice, the solution has to be found iteratively, since no closed form solution of the full set of equations is known. This numerical procedure is explained in Sec. 6.4.2. For the purposes of deriving the equations, we take the rod internal pressure P as given and fixed.

The cladding model is based on the thin wall approximation, which implies constant stress, strain and temperature across the cladding radial direction. The temperature is taken as the average

temperature of the cladding from the solution of the heat equation. In addition, the loading and deformation of the cladding is assumed axisymmetric, and bending strains and stresses are neglected.

Given the internal pressure P, the outside (coolant) pressure P_o, and the cladding inner and outer radii, R_{ci} and R_{co}, the hoop stress σ_θ and the axial stress σ_z are obtained as

$$\sigma_\theta = \frac{R_{ci}P - R_{co}P_o}{R_{co} - R_{ci}}, \tag{33}$$

$$\sigma_z = \frac{R_{ci}^2P - R_{co}^2P_o}{R_{co}^2 - R_{ci}^2}. \tag{34}$$

When the stresses are known, the effective stress and the effective strain can be calculated in order to find out whether the yield stress has been exceeded. The effective stress is calculated as described in section 4.3.3. The effective strain is calculated as a sum of plastic, thermal and elastic strains. If the yield stress has been exceeded, a plastic strain increment is calculated as described in section 4.3.3.

The hoop, axial and radial strains are connected to the stresses through relations

$$\epsilon_\theta = \frac{1}{E}(\sigma_\theta - \nu\sigma_z) + \epsilon_\theta^{\mathrm{th}} + \epsilon_\theta^{\mathrm{pl}} + \mathrm{d}\epsilon_\theta^{\mathrm{pl}}, \tag{35}$$

$$\epsilon_z = \frac{1}{E}(\sigma_z - \nu\sigma_\theta) + \epsilon_z^{\mathrm{th}} + \epsilon_z^{\mathrm{pl}} + \mathrm{d}\epsilon_z^{\mathrm{pl}}, \tag{36}$$

$$\epsilon_r = -\frac{\nu}{E}(\sigma_\theta + \sigma_z) + \epsilon_r^{\mathrm{th}} + \epsilon_r^{\mathrm{pl}} + \mathrm{d}\epsilon_r^{\mathrm{pl}}, \tag{37}$$

Here ϵ_i^{th} are the cladding thermal strains (see Sec. 5.2.3), ϵ_i^{pl} the cladding plastic strains, $\mathrm{d}\epsilon_i^{\mathrm{pl}}$ the cladding plastic strain increments (zero if yield stress was not exceeded), E the Young's modulus (Sec. 5.2.5) and ν the Poisson ratio (Sec. 5.2.6). The strains relate the dimensions of the cladding in the hot state to the dimensions in the cold state. The axial strain is essentially decoupled from the hoop and radial strains, so that the axial length of the slice is

$$l_c = (1 + \epsilon_z)l_{c,\mathrm{cold}}, \tag{38}$$

where the subscript k identifying the axial slice has been dropped for convenience [cf. Eq. (32)]. The corresponding relations for the cladding inner and outer radii can be derived from the change in the radius of the cladding midplane, $(R_{co} + R_{ci})/2 \equiv \bar{R} = (1 + \epsilon_\theta)\bar{R}_{\mathrm{cold}}$ and the change in the cladding thickness, $R_{co} - R_{ci} = (1 + \epsilon_r)(R_{co,\mathrm{cold}} - R_{ci,\mathrm{cold}})$. The resulting expressions for R_{ci} and R_{co} are

$$R_{ci} = R_{ci,\mathrm{cold}}\left(1 + \frac{1}{2}\epsilon_\theta + \frac{1}{2}\epsilon_r\right) + R_{co,\mathrm{cold}}\left(\frac{1}{2}\epsilon_\theta - \frac{1}{2}\epsilon_r\right), \tag{39}$$

$$R_{co} = R_{ci,\mathrm{cold}}\left(\frac{1}{2}\epsilon_\theta - \frac{1}{2}\epsilon_r\right) + R_{co,\mathrm{cold}}\left(1 + \frac{1}{2}\epsilon_\theta + \frac{1}{2}\epsilon_r\right). \tag{40}$$

Equations (39) and (40) relate the radii to their cold-state values and the strains, thus completing the model.

The open gap model of Eqs. (33)–(40) is solved iteratively. This is because the values of R_{ci} and R_{co} are used to determine the stresses σ_θ and σ_z, which in turn are used to solve R_{ci} and R_{co}.

4.3.2 Closed gap model

Strong contact. If the open gap model indicates that the inner surface of the cladding is in contact with the pellet, *i.e.*, $R_{ci} \leq R_f + \rho_f + \rho_c$, then the solution of the mechanical equilibrium proceeds with the closed gap model. For the closed gap model, one uses similar relations to the open gap model, albeit with different boundary conditions. Since the gap is closed, the internal gas pressure cannot be used as a boundary condition. Instead, the contact pressure P_{contact} between the pellet and the cladding remains to be determined. However, the inner radius of the cladding is fixed by the contact, so that

$$R_{ci} = R_f + \rho_f + \rho_c. \tag{41}$$

In addition, the axial strain of the cladding, ϵ_z, is determined by the no-slip condition at the pellet-cladding boundary. Any axial strain of the pellet that takes place after the gap has closed is added to the cladding axial strain. The axial strain of the cladding is therefore

$$\epsilon_z = \epsilon_{z,0} + \epsilon_z^{\text{fuel}} - \epsilon_{z,0}^{\text{fuel}}, \tag{42}$$

where the additional subscript 0 indicates the strain just prior to gap closing.

For the closed gap, the cladding outer radius can be solved explicitly. After some algebraic manipulation, one gets

$$\begin{aligned} R_{co} = &\frac{\frac{1}{\nu}(R_{co,\text{cold}} + R_{ci,\text{cold}}) - 2R_{co,\text{cold}}}{\frac{1}{\nu}(R_{co,\text{cold}} + R_{ci,\text{cold}}) - 2R_{ci,\text{cold}}} R_{ci} + \frac{R_{co,\text{cold}}^2 - R_{ci,\text{cold}}^2}{\frac{1}{\nu}(R_{co,\text{cold}} + R_{ci,\text{cold}}) - 2R_{ci,\text{cold}}} \cdot \\ &\left[\frac{1}{\nu} - \epsilon_z + \epsilon_\theta^{\text{th}} + \epsilon_\theta^{\text{pl}} + \mathrm{d}\epsilon_\theta^{\text{pl}} + \epsilon_z^{\text{th}} + \epsilon_z^{\text{pl}} + \mathrm{d}\epsilon_z^{\text{pl}} + \left(\frac{1}{\nu} - 1\right)\left(\epsilon_r^{\text{th}} + \epsilon_r^{\text{pl}} + \mathrm{d}\epsilon_r^{\text{pl}}\right)\right] \end{aligned} \tag{43}$$

From R_{ci} and R_{co} one can then solve the hoop and radial strains using

$$\epsilon_\theta = \frac{R_{co} + R_{ci}}{R_{co,\text{cold}} + R_{ci,\text{cold}}} - 1, \tag{44}$$

$$\epsilon_r = \frac{R_{co} - R_{ci}}{R_{co,\text{cold}} - R_{ci,\text{cold}}} - 1, \tag{45}$$

Equations (44) and (45) are equivalent with Eqs. (35) and (37). The strains then give the stresses as

$$\sigma_\theta = \frac{E}{1+\nu}\left(\left(\epsilon_\theta - \epsilon_\theta^{pl} - \mathrm{d}\epsilon_\theta^{pl}\right) - \left(\epsilon_r - \epsilon_r^{pl} - \mathrm{d}\epsilon_r^{pl}\right)\right), \tag{46}$$

$$\sigma_z = \nu\sigma_\theta + E\left(\epsilon_z - \epsilon_z^{\text{th}} - \epsilon_z^{\text{pl}} - \mathrm{d}\epsilon_z^{\text{pl}}\right), \tag{47}$$

Finally, the contact pressure is calculated from

$$P_{\text{contact}} = \frac{\sigma_\theta\left(R_{co} - R_{ci}\right) + P_o R_{co}}{R_{ci}}. \tag{48}$$

If the effective stress (calculated as described in section 4.3.3) exceeds the yield stress, a plastic strain increment is calculated as described in section 4.3.3. After the plastic strain increment has been calculated, the total strains are calculated from Eqs. (44) and (45), the stresses and contact pressure updated and R_{co} calculated.

Weak contact. If the solution of the closed gap model with the strong contact assumptions gives an interfacial pressure that is lower than the internal gas pressure, $P_{\text{contact}} < P$, then the gas can push the cladding and the pellet slightly apart, allowing them to slide against each other. In this situation, the no-slip condition in the axial direction no longer holds. Instead, the axial strain of the cladding adjusts until the contact pressure equals the internal rod pressure, and the cladding again becomes axially locked with the pellet. In the weak contact case, no more plastic deformation is calculated.

For the weak contact case, the contact pressure $P_{\text{contact}} = P$. The cladding inner radius R_{ci} is given as for the strong contact case. The outer radius R_{co} can be solved from the implicit equation,

$$\begin{aligned} &R_{co}^3 \frac{1}{R_{co,\text{cold}} - R_{ci,\text{cold}}} + R_{co}^2 \left(-\frac{R_{ci}}{R_{co,\text{cold}} - R_{ci,\text{cold}}} - 1 - \epsilon_r^{\text{th}} - \epsilon_r^{\text{pl}} - \mathrm{d}\epsilon_r^{\text{pl}} + \frac{2\nu}{E} P_o \right) \\ &+ R_{co} \left[-\frac{R_{ci}^2}{R_{co,\text{cold}} - R_{ci,\text{cold}}} - \frac{\nu}{E} (P_o - P) \right] \\ &+ \frac{R_{ci}^3}{R_{co,\text{cold}} - R_{ci,\text{cold}}} + \left(1 + \epsilon_r^{\text{th}} + \epsilon_r^{\text{pl}} + \mathrm{d}\epsilon_r^{\text{pl}} + \frac{2\nu}{E} P \right) R_{ci}^2 = 0, \end{aligned} \tag{49}$$

using Newton-Raphson iteration [5]. The stresses and strains can then be solved from

$$\sigma_\theta = \frac{R_{ci} P_{\text{contact}} - R_{co} P_o}{R_{co} - R_{ci}}, \tag{50}$$

$$\sigma_z = \frac{R_{ci}^2 P_{\text{contact}} - R_{co}^2 P_o}{R_{co}^2 - R_{ci}^2}. \tag{51}$$

$$\epsilon_\theta = \frac{1}{E}(\sigma_\theta - \nu\sigma_z) + \epsilon_\theta^{\text{th}} + \epsilon_\theta^{\text{pl}} + \mathrm{d}\epsilon_\theta^{\text{pl}}, \tag{52}$$

$$\epsilon_z = \frac{1}{E}(\sigma_z - \nu\sigma_\theta) + \epsilon_z^{\text{th}} + \epsilon_z^{\text{pl}} + \mathrm{d}\epsilon_z^{\text{pl}}, \tag{53}$$

$$\epsilon_r = -\frac{\nu}{E}(\sigma_\theta + \sigma_z) + \epsilon_r^{\text{th}} + \epsilon_r^{\text{pl}} + \mathrm{d}\epsilon_r^{\text{pl}}. \tag{54}$$

In the closed gap model, since the inner radius R_{ci} is fixed by the boundary condition, there is no need to iterate the solution with the solution of the internal pressure P (irrespective of the strength of the contact).

4.3.3 Plastic deformation model

Only time-independent plasticity of the cladding is modeled in FINIX, and only infinitesimal strains are covered by the approach. The calculations of the increment in plastic strain proceed similarly in both the open gap and the closed gap, strong contact cases. The following assumptions are made in FINIX concerning cladding plasticity:

- The cladding behaves isotropically,
- the cladding is incompressible,

- the cladding follows the strain hardening hypothesis,
- the cladding yields according to the von Mises yield criterion,
- the cladding follows the Prandtl-Reuss flow rule.

The following discussion and definitions are based on Refs. [15] and [16]. The onset of yielding can be determined from a yield function. When the yield function has a value below zero, the cladding behaves elastically. When the yield function is zero, the cladding yields, and positive values are impossible. In FINIX, the von Mises yield function is used. The von Mises yield function, f_{vM}, for a strain-hardening material is as follows:

$$f_{vM} = \sigma_e - \sigma_Y(\epsilon_e). \tag{55}$$

The so-called yield surface is determined by $f_{vM} = 0$. In Eq. (55), σ_Y is the yield stress, which is a function of effective strain, and σ_e is the (von Mises) effective stress. The calculation of the yield stress is discussed in section 5.2.7. The effective stress (and analogously, strain) is defined as

$$\sigma_e = \sqrt{\frac{3}{2}\sum_i s_i s_i}, \tag{56}$$

$$\epsilon_e = \sqrt{\frac{2}{3}\sum_i e_i e_i}, \tag{57}$$

where s_i is the deviatoric stress and e_i the deviatoric strain, or

$$s_i = \sigma_i - \frac{Tr(\boldsymbol{\sigma})}{3} = \sigma_i - p_{hs}, \tag{58}$$

$$e_i = \epsilon_i - \frac{Tr(\boldsymbol{\epsilon})}{3}. \tag{59}$$

Here, $Tr(\boldsymbol{\sigma})$ is the trace of the (Cauchy) stress tensor $\boldsymbol{\sigma}$, or the sum of the principal values of the tensor. The second term in the deviatoric stress equation is therefore equivalent to the hydrostatic pressure, p_{hs}. Note that in this discussion, only the principal values of stresses and strains are used, so $\sigma_i = \sigma_{ii}$, for example.

Using Hooke's law (see Eqs. (35), (36) and(37)) and the definition of deviatoric stress and strain, the following relation between the deviatoric stress and strain can be found:

$$s_i = 2\left(\frac{E}{2\,(1+\nu)}\right) e_i = 2Ge_i, \tag{60}$$

where we have defined the shear modulus, G. For effective stress and effective strain, a similar relationship can be found using their definitions and Eq. (60):

$$\sigma_e = 3G\epsilon_e. \tag{61}$$

When stresses resulting in an effective stress greater than the yield stress is calculated to occur with the elastic solution, FINIX solves the increment in plastic strain by the radial return algorithm in order to return the stresses to the yield surface determined by Eq. (55). The radial return algorithm is an implicit integration algorithm, so all values of stress and strain refer to stresses and strains at the end of the time step. The effective stress in Eq. (55) can be decomposed into a effective trial stress and an effective plastic strain increment, $\mathrm{d}\epsilon_e^{pl}$, that returns the stress to the yield surface (see Ref. [16] for the derivation):

$$\sigma_e = \sigma_{e,\mathrm{trial}} - 3G\mathrm{d}\epsilon_e^{pl}. \tag{62}$$

The effective trial stress can be calculated from the trial stress with Eq. (56). The trial stresses are calculated using the following equation (see Ref. [16] for the derivation):

$$\sigma_{\mathrm{trial},i} = \sigma_i + 2Gd\epsilon_i^{pl}. \tag{63}$$

The plastic strain increment components, $d\epsilon_i^{pl}$, are obtained from the Prandtl-Reuss flow rule, Eq. (65). Substituting Eq. (62) into Eq. (55), we can now solve for the effective plastic strain increment that satisfies the yield function:

$$\sigma_{e,\mathrm{trial}} - 3G\mathrm{d}\epsilon_e^{pl} - \sigma_Y(\epsilon_e^{pl}) = 0. \tag{64}$$

This is called the radial return method, which is widely used in solving the plasticity equations. The effective plastic strain increment is found from Eq. (64) by Newton-Raphson iteration. An initial guess of $1{\cdot}10^{-15}$ is used in FINIX. When the effective plastic strain increment is found, the individual components of the plastic strain increment are obtained from the Prandtl-Reuss flow rule:

$$\mathrm{d}\epsilon_i^{pl} = \frac{3}{2}\frac{s_i}{\sigma_e}\mathrm{d}\epsilon_e^{pl}. \tag{65}$$

With the knowledge of the plastic strain increment components, one may calculate the stresses at the end of the time step by solving for σ_i in Eq. (63). The new stresses and plastic strain increments are then used to calculate the total strains from Hooke's law. In the open gap case, both the inner and outer radius of the cladding are updated with the new strains, but in the closed gap, strong contact case only the cladding outer radius is updated.

The increments in plastic strain are only added to the plastic strain after the solution for the time step has converged. It must be noted that the effective plastic strain must be treated separately from the plastic strains as the effective plastic strain depends on the past deformation and does

not decrease. The effective plastic strain is a sum of all the effective plastic strain increments that have occurred in the past. Therefore the effective strain increment calculated with the radial return method is saved separately and added to the effective plastic strain after the solution has converged.

5 Material correlations

The material correlations used in FINIX have been mostly adopted from the MATPRO library [13] and the FRAPTRAN fuel performance code [11]. VVER correlations have mostly been adopted from Shestopalov *et al.* [17]. Typically, the correlations are taken 'as is'. Since they have been thoroughly tested within other fuel performance codes, detailed study of their applicability was not considered necessary at this point. Their properties are discussed in, *e.g.*, Refs. [13, 18].

5.1 Fuel properties

5.1.1 Specific heat

The fuel specific heat correlation is the same as in FRAPTRAN and MATPRO [11, 13]. The specific heat c_m is given as

$$c_m = K_1 \frac{\Theta^2 \exp(\Theta/T)}{T^2 \left[\exp(\Theta/T) - 1\right]^2} + K_2 T + K_3 \frac{Y E_d}{2RT^2} \exp(-E_d/RT), \tag{66}$$

where T is the temperature in Kelvin, R is the ideal gas constant (≈ 8.314 J/molK) and Y is the oxygen-to-metal ratio. For UO_2, the numerical values of the constants are $K_1 = 296.7$ J/kgK, $K_2 = 0.0243$ J/kgK^2, $K_3 = 8.745 \cdot 10^7$ J/kg, $\Theta = 535.285$ K and $E_d = 1.577 \cdot 10^5$ J/mol.

5.1.2 Thermal conductivity

The FRAPTRAN correlation (see Ref. [13]) is used for the fuel thermal conductivity. The thermal conductivity λ is given by

$$\lambda = 1.0789 \frac{d}{1 + 0.5(1 - d)} \lambda_{95}, \tag{67}$$

where λ_{95} is the thermal conductivity for UO_2 at 95 % of the theoretical density, and d is the as-fabricated density of pellet as a fraction from the theoretical value. The correlation for λ_{95} is

$$\lambda_{95} = \left[A + a \cdot gad + BT + f(Bu) + \left(1 - 0.9e^{-0.04Bu}\right) g(Bu) h(T)\right]^{-1} + \frac{E}{T^2} e^{-F/T}. \tag{68}$$

Here gad is the gadolinia weight fraction, Bu is the burnup (GWd/MTU) and T is the temperature (K). The functions introduced in Eq. (68) are $f(Bu) = 0.00187Bu$, $g(Bu) = 0.038Bu^{0.28}$ and $h(T) = [1 + 396 \exp(-Q/T)]^{-1}$, with $Q = 6380$ K. The constants in Eq. (68) are $A = 0.0452$ mK/W, $a = 1.1599$, $B = 2.46 \cdot 10^{-4}$ m/W, $E = 3.5 \cdot 10^9$ WK/m and $F = 16361$ K.

5.1.3 Thermal strain

The thermal expansion correlation for UO_2 is taken from MATPRO/FRAPTRAN. The correlation is valid in the solid phase (below $T \approx 3110$) of the fuel pellet, and gives zero strain at 300 K. The (linear) thermal strain is

$$\epsilon_{th} = K_1 T - K_2 + K_3 e^{-E_d/k_B T}, \tag{69}$$

where k_B is the Boltzmann constant, $K_1 = 9.8 \cdot 10^{-6}$ 1/K, $K_2 = 2.94 \cdot 10^{-3}$, $K_3 = 0.316$ and $E_d = 1.32 \cdot 10^{-19}$ J. Ref. [13] reports the value $K_2 = 2.61 \cdot 10^{-3}$ for the second constant. However, it is easy to verify that $\epsilon_{th}(T = 300\text{ K}) \approx 0$ is given by $K_2 = 2.94 \cdot 10^{-3}$.

5.1.4 Radial pellet relocation

Cracking and radial relocation of the fuel pellet due to irradiation and thermal stresses is modeled using the FRAPCON correlation for the pellet radial cracking. Relocation is given as the fractional closure of the gap in relation of the as-fabricated gap G as

$$\Delta G/G = \begin{cases} 0.3 + 0.1 f(Bu), & \text{for } \textit{LHR} < 20 \text{ kW/m}, \\ 0.28 + g(LHR) + [0.12 + g(LHR)] f(Bu), & \text{for } 20 \text{ kW/m} < \textit{LHR} < 40 \text{ kW/m}, \\ 0.32 + 0.18 f(Bu), & \text{for } \textit{LHR} > 40 \text{ kW/m}, \end{cases} \tag{70}$$

where $f(Bu) = Bu/5$ for $Bu < 5$ MWd/kgU and $f(Bu) = 1$ otherwise, and $g(LHR) = 0.0025(LHR - 20)$, with the linear hear rate LHR given in kW/m and the burnup Bu in MWd/kgU.

Half of the radial relocation is considered permanent (hard) and half recoverable (soft). In FINIX, the total (soft + hard) relocation is only taken into account in determining the gap conductance, while only the hard relocation is considered in the mechanical model and in determining the reduction in free volume for the pressure calculation.

5.2 Cladding properties

5.2.1 Specific heat

Zircaloy cladding specific heat (in J/kgK) is given as a function of temperature in a tabulated form in Table 2. The data is adopted from Ref. [13]. Between 300 K and 1248 K, the temperature is calculated by linear interpolation from the values of Table 2, while for temperatures lower than 300 K and higher than 1248 K, the specific heat is assumed constant.

For Zr1%Nb cladding, the data is presented similarly in a tabulated form in Table 3. The data is adopted from Ref. [17]. Between 393 K and 1400 K, the temperature is calculated by linear interpolation from the values of Table 3, while for temperatures lower than 393 K and higher than 1400 K, the specific heat is assumed constant.

5.2.2 Thermal conductivity

The thermal conductivity of Zircaloy below 2098 K is given by the FRAPTRAN correlation

$$\lambda = 7.51 + 2.09 \cdot 10^{-2}T - 1.45 \cdot 10^{-5}T^2 + 7.67 \cdot 10^{-9}T^3, \tag{71}$$

where the temperature T is given in Kelvin and the conductivity λ in W/mK [13].

According to Shestopalov *et al.* [17], the thermal conductivity of Zr1%Nb cladding below 2133 K is given by

$$\lambda = 15.0636e^{0.4618 \cdot 10^{-3}T}. \tag{72}$$

For both Zircaloy cladding at temperatures higher than 2098 K and Zr1%Nb cladding at temperatures higher than 2133 K, the conductivity is

$$\lambda = 36.0 \text{ (W/mK)}. \tag{73}$$

5.2.3 Thermal strain

The correlation for Zircaloy cladding thermal strain is based on the correlation used in FRAPTRAN. The FRAPTRAN formulae give non-zero strain at $T = 300$ K, which we assume be the cold reference state of the system. Thus, the constant term is adjusted to give zero strain at 300 K. The difference w.r.t. the FRAP correlations is less than 10^{-4}.

The correlation is given separately for the axial and diametral strains (the strain is assumed to be isotropic on the plane perpendicular to the axial direction, hence the diametral strain is used for both the radial and hoop thermal strain). The correlation is given separately for the α and β phases of zirconium and interpolated in the intermediate regime.

Table 2. *The specific heat capacity of the Zircaloy cladding [13]. The specific heat is assumed to have a constant value both below $T = 300$ K and above $T = 1248$ K.*

Temperature (K)	Specific heat (J/kgK)
300	281
400	302
640	331
1090	375
1093	502
1113	590
1133	615
1153	719
1173	816
1193	770
1213	619
1233	469
1248	356

Table 3. *The specific heat capacity of the Zr1%Nb cladding [17]. The specific heat is assumed to have a constant value both below $T = 393$ K and above $T = 1400$ K.*

Temperature (K)	Specific heat (J/kgK)
393	345
473	360
573	370
673	380
773	383
873	385
883	448
973	680
1025	816
1073	770
1153	400
1173	392
1200	392
1300	393
1400	393

For $T \leq 1073$ K, the strains are

$$\epsilon^{\alpha}_{\text{axial}} = -1.1924085 \cdot 10^{-4} + (T - 273.15) \cdot 4.441 \cdot 10^{-6}, \tag{74}$$

$$\epsilon^{\alpha}_{\text{diametral}} = -1.80459 \cdot 10^{-4} + (T - 273.15) \cdot 6.721 \cdot 10^{-6}, \tag{75}$$

and for $T \geq 1273$ K

$$\epsilon^{\beta}_{\text{axial}} = -8.3942 \cdot 10^{-3} + (T - 273.15) \cdot 9.70 \cdot 10^{-6}, \tag{76}$$

$$\epsilon^{\beta}_{\text{diametral}} = -6.7432 \cdot 10^{-3} + (T - 273.15) \cdot 9.70 \cdot 10^{-6}. \tag{77}$$

For the intermediate temperatures the strains are interpolated from the α and β phase values so that for 1073 K $< T <$ 1273 K

$$\epsilon^{\alpha-\beta}_{\text{axial}} = \frac{1273\text{K} - T}{200\text{K}} \epsilon^{\alpha}_{\text{axial}} + \frac{T - 1073\text{K}}{200\text{K}} \epsilon^{\beta}_{\text{axial}}, \tag{78}$$

$$\epsilon^{\alpha-\beta}_{\text{diametral}} = \frac{1273\text{K} - T}{200\text{K}} \epsilon^{\alpha}_{\text{diametral}} + \frac{T - 1073\text{K}}{200\text{K}} \epsilon^{\beta}_{\text{diametral}}. \tag{79}$$

The correlations for Zr1%Nb cladding thermal strains are provided by Shestopalov *et al.* [17]. The correlations give zero thermal strain at 300 K. In the correlations below, $\Delta T = (T - 883.0)$. The correlations are given for five temperature intervals. Below 573 K, the strains are given by

$$\epsilon_{\text{axial}} = -0.127813365 \cdot 10^{-2} + 3.85875 \cdot 10^{-6} T + 0.1338985 \cdot 10^{-8} T^2, \tag{80}$$

$$\epsilon_{\text{diametral}} = -0.199649865 \cdot 10^{-2} + 5.6539 \cdot 10^{-6} T + 0.3336985 \cdot 10^{-8} T^2, \tag{81}$$

at temperatures above 573 K but below 883 K by

$$\epsilon_{\text{axial}} = 0.13725577 \cdot 10^{-2} + 5.4 \cdot 10^{-6} \left(T - 573.0\right), \tag{82}$$

$$\epsilon_{\text{diametral}} = 0.3336985 \cdot 10^{-8} T^2 + 5.6539 \cdot 10^{-6} T - 0.199649865 \cdot 10^{-2}, \tag{83}$$

at temperatures above 883 K but below 1153 K by

$$\epsilon_{\text{axial}} = 3.0465577 \cdot 10^{-3} + 2.312 \cdot 10^{-8} \Delta T - 7.358 \cdot 10^{-8} \Delta T^2 + 1.7211 \cdot 10^{-10} \Delta T^3, \tag{84}$$

$$\epsilon_{\text{diametral}} = 5.597700 \cdot 10^{-3} + 2.312 \cdot 10^{-8} \Delta T - 7.358 \cdot 10^{-8} \Delta T^2 + 1.7211 \cdot 10^{-10} \Delta T^3, \tag{85}$$

at temperatures above 1153 K but below 2133 K by

$$\epsilon_{\text{axial}} = 1.076549 \cdot 10^{-3} + 9.7 \cdot 10^{-6} \left(T - 1153.0\right), \tag{86}$$

$$\epsilon_{\text{diametral}} = 3.6276 \cdot 10^{-3} + 9.7 \cdot 10^{-6} \left(T - 1153.0\right), \tag{87}$$

and finally at temperatures higher than 2133 K by

$$\epsilon_{\text{axial}} = 1.0582459 \cdot 10^{-2}, \tag{88}$$

$$\epsilon_{\text{diametral}} = 1.3133600 \cdot 10^{-2}. \tag{89}$$

5.2.4 Meyer's hardness

Meyer's hardness H_M is used in the gap conductance model to determine the magnitude of the contact heat transfer coefficient. The following correlation is used for H_M of Zircaloy cladding (note the error in sign of last term in [13]):

$$H_M = \exp\left[26.034 + T(-0.026394 + T(4.3502 \cdot 10^{-5} - 2.5621 \cdot 10^{-8} T))\right], \tag{90}$$

where the dimension of H_M is (N/m^2). In addition, the lower limit of H_M of Zircaloy is set as $1.94 \cdot 10^8$ N/m^2.

Meyer's hardness for Zr1%Nb cladding is given by Shestopalov *et al.* [17] for temperatures below 800 K as

$$H_M = 2172.1 \cdot 10^6 - 10.7055 \cdot 10^6 T + 27650 T^2 - 32.78 T^3 + 0.01423 T^4, \tag{91}$$

and for temperatures over 800 K as

$$H_M = \exp\left(26.034 - 2.6394 \cdot 10^{-2} T + 4.3502 \cdot 10^{-5} T^2 - 2.5621 \cdot 10^{-8} T^3\right), \tag{92}$$

where the dimensions of H_M is (N/m^2). Additionally, a minimum hardness of $1 \cdot 10^5$ N/m^2 is specified for Zr1%Nb cladding.

5.2.5 Young's modulus

The Young's modulus E of Zircaloy cladding is given separately for the α and β phases [13]. Below 1094 K, the correlation used is

$$E^{\alpha} = (1.088 \cdot 10^{11} - 5.475 \cdot 10^{7} T + K_1 + K_2)/K_3, \tag{93}$$

with T in Kelvin and E in N/m^2. The coefficient K_1, K_2 and K_3 are calculated from

$$K_1 = (6.61 \cdot 10^{11} + 5.912 \cdot 10^{8} T)\Delta, \tag{94}$$
$$K_2 = -2.6 \cdot 10^{10} C, \tag{95}$$
$$K_3 = 0.88 + 0.12 e^{-\Phi/10^{25}}. \tag{96}$$

Here Δ is the average oxygen concentration minus the oxygen concentration of as-received cladding (kg oxygen/kg Zircaloy), C is the cold work (dimensionless ratio or areas) and Φ is the fast neutron fluence (n/m^2).

From 1239 K upwards the correlation is

$$E^{\beta} = 9.21 \cdot 10^{10} - 4.05 \cdot 10^{7} T. \tag{97}$$

In the intermediate range (1094 K $< T <$ 1239 K), the value is interpolated as

$$E^{\alpha-\beta} = \frac{1239\text{K} - T}{(1239 - 1094)\text{K}} E^{\alpha} + \frac{T - 1094\text{K}}{(1239 - 1094)\text{K}} E^{\beta}. \tag{98}$$

The Young's modulus correlation for Zr1%Nb cladding is as in FRAPTRAN-1.4 [13]. At temperatures below 1073 K, it is given by

$$E = 1.21 \cdot 10^{11} - 6.438 \cdot 10^{7} T + 3.021 \cdot 10^{12} x_O, \tag{99}$$

where x_O is the mass fraction of oxygen in the cladding. At temperatures above 1073 K, the Young's modulus is given by

$$E = 9.129 \cdot 10^{10} - 4.5 \cdot 10^{7} T. \tag{100}$$

A minimum of 1 Pa is specified.

5.2.6 Poisson's ratio

The correlation for the Poisson's ratio ν is taken from FRAPTRAN [13]:

$$\nu = 0.42628 - 5.556 \cdot 10^{-5} T. \tag{101}$$

The temperature T is given in Kelvin, and ν is dimensionless. The same correlation is used for both Zircaloy and Zr1%Nb claddings.

5.2.7 Yield stress

The PNNL stress-strain correlation [19] is used in the calculation of the yield stress for Zircaloy and Zr1%Nb claddings. Parameters for the correlation are reported by Geelhood *et al.* [19] for Zircaloy and by Shestopalov *et al.* [17] for Zr1%Nb cladding. The yield stress, σ_Y, correlation has the following form:

$$\sigma_Y = K\epsilon^n \left(\frac{\dot{\epsilon}}{10^{-3}} \right)^m \tag{102}$$

where K, n and m are the strength coefficient, strain hardening exponent and strain rate exponent, parameters fitted to experimental data, and $\dot{\epsilon}$ the strain rate. To find the yield stress after an amount of plastic strain has been accumulated, one must take into account that the yield stress then occurs at the intersection of the yield stress curve, described by Eq. (102), and the elastic stress-strain line:

$$\sigma = E\left(\epsilon - \epsilon^{pl}\right) \tag{103}$$

First setting $\sigma = \sigma_Y$ and solving for ϵ in Eq. (103) and then substituting ϵ in Eq. (102) yields an implicit equation for the yield stress, so the yield stress must be solved iteratively. A value of 34.5 MPa is used as an initial guess. Currently, a fixed value of $1 \cdot 10^{-5}$ is used for the strain rate when determining whether the yield stress has been exceeded and a plastic strain increment should be calculated. However, the strain rate is reported to have a small effect on the calculated yield stress [19].

The strain, ϵ, in the correlation is true strain, and the yield stress is true stress. In FINIX, engineering stress and strain are used, so before using this correlation the values of stress and strain are converted into true stress and strain. The relation between true and engineering (or nominal) stress and strain are as follows:

$$\begin{aligned} \epsilon_{\text{true}} &= \ln\left(\epsilon + 1\right) \\ \sigma_{\text{true}} &= \sigma\left(\epsilon + 1\right) \end{aligned} \tag{104}$$

The major difference between engineering and true strain is that true strain is additive, whereas engineering strain is not. However, with very small strains, the error from using engineering strains additively is small. In the plastic strain calculation, the calculation of the plastic strain increment is done using true strain and stress, and the results are returned to FINIX as engineering strain and stress. Also, care is taken to only compare true yield stress with the true effective stress or engineering yield stress with the engineering effective stress.

Strength coefficient. For Zircaloy, the strength coefficient, K, is subdivided into terms that are functions of temperature of the cladding, T, fast neutron fluence, Φ, cold work, CW, and cladding

type. The strength coefficient for Zircaloy is calculated as follows:

$$K = \frac{K_T \left(1 + K_{CW} + K_\Phi\right)}{K_{clad}}. \tag{105}$$

K_{clad} depends on the type of Zircaloy, and for Zircaloy-2 it is 1.305 and for Zircaloy-4 it is 1.0. The temperature-dependent term K_T is reported for four temperature intervals. At temperatures below 750 K, it is given by

$$K_T = 1.17628 \cdot 10^9 + 4.54859 \cdot 10^5 T - 3.28185 \cdot 10^3 T^2 + 1.72752 T^3, \tag{106}$$

at temperatures over 750 K but below 1090 K by

$$K_T = 2.522488 \cdot 10^6 \exp\left(\frac{2.8500027 \cdot 10^6}{T^2}\right), \tag{107}$$

at temperatures over 1090 K but below 1255 K by

$$K_T = 1.841376039 \cdot 10^8 - 1.4345448 \cdot 10^5 T, \tag{108}$$

and finally at temperatures over 1255 K but below 2100 K by

$$K_T = 4.330 \cdot 10^7 - 6.685 \cdot 10^4 T + 37.579 T^2 - 7.33 \cdot 10^{-3} T^3. \tag{109}$$

The fast neutron fluence dependent term is reported for three fluence intervals. At fluences below $0.1 \cdot 10^{25} \frac{n}{m^2}$ it is given by

$$K_\Phi = \left(-0.1464 + 1.464 \cdot 10^{-25} \Phi\right) \cdot \left(2.25 e^{-20CW} \min\left(1, e^{\frac{T-550}{10}}\right) + 1\right), \tag{110}$$

at fluences above $0.1 \cdot 10^{25} \frac{n}{m^2}$ but below $2 \cdot 10^{25} \frac{n}{m^2}$ by

$$K_\Phi = 2.928 \cdot 10^{-26} \Phi, \tag{111}$$

and finally for fluences above $2 \cdot 10^{25} \frac{n}{m^2}$ but below $12 \cdot 10^{25} \frac{n}{m^2}$ by

$$K_\Phi = 0.53236 + 2.6618 \cdot 10^{-27} \Phi. \tag{112}$$

The cold work dependent term is simply

$$K_{CW} = 0.546 CW, \text{ when } 0 < CW < 0.75. \tag{113}$$

For Zr1%Nb cladding, the strength coefficient is reported separately for irradiated and unirradiated cladding. In FINIX, if the fast neutron fluence is greater than zero, the irradiated cladding correlation is used. For unirradiated cladding, the strength coefficient is given for two temperature intervals. For the temperature range $293K < T < 797.9K$ the strength coefficient is

$$K = 898.3710095 \cdot 10^6 - 1.911883946 \cdot 10^6 + 2.024675204 \cdot 10^3 T^2 - 0.9628259856 T^3, \tag{114}$$

and for the temperature range $797.9K < T < 1223K$ it is

$$K = 1.518065748 \cdot 10^{10} e^{-0.0056080697387T}. \tag{115}$$

For irradiated cladding, the strength coefficient of Zr1%Nb is reported for three temperature intervals. For temperatures above 293 K but below 763 K, it is given by

$$K = 916.8547193 \cdot 10^6 - 0.6046334417 \cdot 10^6 T - 247.4820043 T^2, \tag{116}$$

at temperatures above 763 K but below 859.4 K by

$$K = 4.912469131 \cdot 10^{11} e^{-0.009650275477T}, \tag{117}$$

and at temperatures above 859.4 K but below 1223 K by

$$K = 1.518065748 \cdot 10^{10} e^{-0.005608069738T}. \tag{118}$$

Strain hardening exponent. For Zircaloy, the strain hardening exponent, n, is subdivided into terms that are functions of temperature, fast neutron fluence and the cladding type. The strain hardening exponent is calculated as follows for Zircaloy:

$$n = \frac{n_T n_\Phi}{n_{clad}}. \tag{119}$$

n_{clad} depends on the type of Zircaloy, and for Zircaloy-2 it is 1.6 and for Zircaloy-4 it is 1.0. The temperature dependent term, n_T, is given for four temperature intervals. At temperatures below 419.4 K it is

$$n_T = 0.11405. \tag{120}$$

At temperatures above 419.4 K but below 1099.0772 K, it is given by

$$n_T = -9.490 \cdot 10^{-2} + 1.165 \cdot 10^{-3} T - 1.992 \cdot 10^{-6} T^2 + 9.588 \cdot 10^{-10} T^3, \tag{121}$$

and at temperatures above 1099.0772 K but below 1600 K by

$$n_T = -0.22655119 + 2.5 \cdot 10^{-4} T. \tag{122}$$

At higher temperatures than 1600 K, it is

$$n_T = 0.17344880. \tag{123}$$

The fast neutron fluence dependent term, n_Φ, is given for three fluence intervals. For fluence below $0.1 \cdot 10^{25} \frac{n}{m^2}$, the term is given by

$$n_\Phi = 1.321 + 0.48 \cdot 10^{-25} \Phi, \tag{124}$$

while above $0.1 \cdot 10^{25} \frac{n}{m^2}$ but below $2 \cdot 10^{25} \frac{n}{m^2}$ it is given by

$$n_\Phi = 1.369 + 0.096 \cdot 10^{-25} \Phi, \tag{125}$$

and above $2 \cdot 10^{25} \frac{n}{m^2}$ but below $7.5 \cdot 10^{25} \frac{n}{m^2}$ by

$$n_\Phi = 1.5435 + 0.008727 \cdot 10^{-25} \Phi. \tag{126}$$

At fluences higher than $7.5 \cdot 10^{25} \frac{n}{m^2}$, the fluence dependent term is

$$n_\Phi = 1.608953. \tag{127}$$

For Zr1%Nb cladding, the strain hardening exponent is reported separately for irradiated and unirradiated cladding. In FINIX, if the fast neutron fluence is greater than zero, the irradiated cladding correlation is used. For unirradiated cladding, the strain hardening exponent is given by:

$$n = 0.04628421012 + 0.000197951907T - 3.314868215 \cdot 10^{-7}T^2 + 1.3913294 \cdot 10^{-10}T^3. \quad (128)$$

The lower limit for validity of the unirradiated cladding correlation is 293 K, and the upper limit is 1223 K. For irradiated cladding the strain hardening exponent is given for three temperature intervals. At temperatures above 293 K but over 759 K, it is given by

$$n = -0.1255447757 + 0.001350416112T - 3.536814687 \cdot 10^{-6}T^2 + 3.734672258 \cdot 10^{-9}T^3, \quad (129)$$

at temperatures above 759 K but below 879 K by

$$n = -0.239614587 + 0.002839248035T - 8.226160457 \cdot 10^{-6}T^2 + 9.276772204 \cdot 10^{-9}T^3 - 3.588141876 \cdot 10^{-12}T, \quad (130)$$

and at temperatures above 879 K but below 1223 K by

$$n = 0.04628421012 + 0.000197951907T - 3.314868215 \cdot 10^{-7}T^2 + 1.3913294 \cdot 10^{-10}T^3. \quad (131)$$

Strain rate exponent. For both Zircaloy and Zr1%Nb claddings, the strain rate exponent is only dependent on the temperature of the cladding. For Zircaloy, the strain rate exponent is given for three temperature intervals. Below 750 K, it is

$$m = 0.015. \quad (132)$$

Between 750 K and 800 K, the strain rate exponent is given by

$$m = -0.544338 + 7.458 \cdot 10^{-4}T, \quad (133)$$

and at temperatures over 800 K, it is given by

$$m = -0.20701 - 3.24124 \cdot 10^{-4}T. \quad (134)$$

For Zr1%Nb cladding the strain rate exponent is reported for three temperature intervals. Below 752.2 K but over 293 K it is given by

$$m = 0.02280034483 - 3.448275862 \cdot 10^{-7}T, \quad (135)$$

at temperatures over 752.2 K but below 902.1 K by

$$m = -2.534966886 + 0.006626767224T - 5.303091629 \cdot 10^{-6}T^2 + 1.34653092 \cdot 10^{-9}T^3, \quad (136)$$

and finally at temperatures over 902.1 K but below 1223 K by

$$m = -0.1619955889 + 3.080302048 \cdot 10^{-4}T. \quad (137)$$

5.3 Gas gap and plenum properties

The gas conductivity model and correlations are discussed in Section 3.3.

The plenum gas model in FINIX 0.15.12 has not been updated to treat gases other than helium. The plenum gas is thus assumed to consist solely of helium. While this is a crude approximation, it only affects the plenum temperature through the heat transfer coefficients calculated for the pellet surface and the cladding facing the plenum. Since both are in any case in contact with the same gas and thermal equilibrium in the plenum is assumed, the error resulting from the approximation is manageable until correlations for the other species can be introduced. The correlations for helium are taken from Ref. [20]. Compared to the FRAPTRAN correlations, the numerical values are very similar.

The dynamic viscosity μ, density ρ and Prandtl number Pr are given by

$$\mu = 3.674 \cdot 10^{-7} T^{0.7} \text{ (kg/ms)}, \tag{138}$$

$$\rho = 48.14 \cdot 10^{-5} \frac{P}{T} \left[1 + 0.4446 \cdot 10^{-5} \frac{P}{T^{1.2}} \right] \text{ (kg/m}^3\text{)}, \tag{139}$$

$$Pr = \frac{0.717}{1 + 1.123 \cdot 10^{-8} P} T^{-\left(0.01 - 1.42 \cdot 10^{-9} P\right)} \text{ (dimensionless)}. \tag{140}$$

In the above, T is given in Kelvin and P in N/m^2.

The kinematic viscosity is given by $\nu = \frac{\mu}{\rho}$.

5.4 Coolant

FINIX has a rudimentary implementation of a thermal-hydraulic model for calculation of heat transfer coefficients from the cladding to the coolant. The correlations are valid below the critical heat flux (CHF), in the forced convection and nucleate boiling regime. The validity of the correlations is internally checked by calculating the critical heat flux from the EPRI-1 correlation [21], and by comparing with the computed heat flux. Currently, FINIX has no model to calculate changes in bulk coolant temperature. Therefore, the coolant temperature is taken as given by the user.

5.4.1 Heat transfer coefficient

The heat transfer coefficient h is given as the sum of the heat transfer coefficients from the Dittus-Boelter correlation (h_{DB}) and the Thom correlation (h_{Thom}). The latter describes the additional convection in the nucleate boiling regime, and is zero if the cladding surface temperature is below the saturated coolant temperature, *i.e.*, if $T_{co} < T_{sat}$.

The Dittus-Boelter correlation is given in British units as [12]

$$\begin{aligned} h_{\mathrm{DB}} =& [(-5.1889 \cdot 10^{-5} + 6.5044 \cdot 10^{-8} T_w) T_w + (3.5796 \cdot 10^{-7} - 1.0337 \cdot 10^{-9} T_w) P_w \\ &+ 3.2377 \cdot 10^{-2}] \phi_w^{0.8} D_e^{-0.2}, \end{aligned} \tag{141}$$

where T_w and P_w are the temperature and pressure of the coolant (assumed light water in the correlations), ϕ_w is the coolant mass flux and D_e the hydraulic diameter,

$$D_e = \begin{cases} 2\frac{d^2-\pi R_{co}^2}{\pi R_{co}}, & \text{for square lattice,} \\ \frac{\sqrt{3}d^2-2\pi R_{co}^2}{\pi R_{co}}, & \text{for hexagonal lattice,} \end{cases} \tag{142}$$

with d the rod pitch. Although the correlation for h_{DB} is given in British units, the conversion to and from SI units is handled internally by the respective FINIX functions. The output from the calculation is therefore in units of W/m^{2}K.

The Thom correlation, also in British units, is [11]

$$h_{\mathrm{Thom}} = \left(\frac{e^{P_w/1260}}{0.072}(T_{co} - T_{sat}) \right)^2 /(T_{co} - T_w), \tag{143}$$

with T_{sat} given by the Frapcon correlation

$$T_{sat} = \left[((0.4616716 \cdot 10^{-8} P_w - 0.4424529 \cdot 10^{-4})P_w) + 0.19042968\right] P_w + 394.03519, \tag{144}$$

where the pressure is in psia and temperature in °F. For $T_{co} < T_{sat}$, $h_{\mathrm{Thom}} = 0$.

5.4.2 Critical heat flux

The critical heat flux is estimated from the EPRI correlation [21], which is valid for a wide range of experimental parameters. The correlation can be used to derive a criticality criterion for the heat flux. If the relation

$$q_k > \frac{A - x_k}{C} \tag{145}$$

is satisfied, then the heat flux q_k (in units of MBtu/hrft^2) at axial node k exceeds the CHF. Here x_k is the thermal equilibrium quality (the non-dimensional expression of coolant enthalpy),

$$x_k = \frac{h_k - h_{sat}}{h_{vap}} \tag{146}$$

where the bulk fluid enthalpy h_k is evaluated at the axial node k, and the saturated liquid enthalpy, h_{sat}, and the enthalpy of vaporization, h_{vap}, are given by internal correlations. The coefficients A and C in Eq. (146) are given by

$$A = 0.5328(P_w/P_{crit})^{0.1212}\phi^{-0.3040-0.3285(P_w/P_{crit})}, \tag{147}$$

$$C = 1.6151(P_w/P_{crit})^{1.4066}\phi^{0.4843-2.0749(P_w/P_{crit})}, \tag{148}$$

where the critical pressure $P_{crit} = 3208.2$ psia and the coolant mass flux ϕ is given in units of Mlbm/hrft^2.

6 Numerical implementation

6.1 General outline of the execution order

The FINIX calculation of the thermal and mechanical time evolution of the fuel rod proceeds in discrete time steps δt. For each time step, the thermal and mechanical solutions are found

by numerical iteration. The iteration process is schematically presented in Fig. 1. The iteration consists of two main loops. The outer loop consists of solving the thermal properties of the fuel and the cladding, the gap conductance and the heat equation and plenum temperature. The second loop consists of solving the internal pressure and pellet and cladding deformations, and is situated within the outer thermal iteration loop. For the mechanical solution, the internal pressure is used as a convergence criterion, while for the outer loop, convergence of the gap conductivities for all axial nodes is required. On the algorithm level, the iteration is performed using the secant method, which is a method similar to the Newton-Raphson method, but where the function derivative is evaluated numerically instead of analytically. The method is described in detail in, *e.g.*, Ref. [5].

The numerical methods used to solve the individual modules are described in the following Sections.

6.2 Thermal model

6.2.1 FEM discretization of the 1D heat equation

As was discussed in Section 3.1, the temperature in the pellet and cladding is solved in axial slices, in each of which the temperature T is assumed independent of the axial and azimuthal coordinates z and θ. The heat equation then takes the form

$$C_V[T(r)]\frac{\partial T}{\partial t} - \frac{1}{r}\frac{\partial}{\partial r}\left[\lambda[T(r)]r\frac{\partial T}{\partial r}\right] - s(r) = 0, \tag{149}$$

with C_V denoting the volumetric heat capacity and λ the conductivity. Note that neither the heat capacity nor conductivity is assumed constant w.r.t. the coordinate r.

The heat equation (149) is discretized with the Finite Element Method (FEM) [22]. The pellet is divided into $n_f - 1$ radial *elements*, with the i:th element comprising the volume between the *nodes* at $r = r_i$ and $r = r_{i+1}$. The first node is located at the inner surface of the pellet ($r_1 = R_0$), while the last node is at the pellet outer surface ($r_{n_f} = R_f$). The cladding is similarly divided into $n_c - 1$ elements and n_c nodes, with the first cladding node at the cladding inner surface ($r_{n_f+1} = R_{ci}$) and the last at the outer surface ($r_{n_f+n_c} = R_{co}$).

The gas gap element (between nodes n_f and $n_f + 1$) is handled through the gap conductance boundary conditions, as will be discussed below. In what follows, an element will refer to a general element, either within the pellet or in the cladding, unless otherwise indicated. For the pellet elements, the material parameters (conductivity, heat capacity) are be given by the correlations of Sec. 5.1, and for the cladding elements by the correlations of Sec. 5.2.

6.2.2 Discretization of one element

In the finite element method, the continuous equations are discretized first for each element. The global discretization of the whole system is then assembled from the discretized individual elements. For each element, the numerical solution provides the value of the temperature only at the location of the nodes. Within the element, the solution is approximated by *shape functions*, or *basis functions*. In the simplest case, the basis functions are linear, so that within the element the

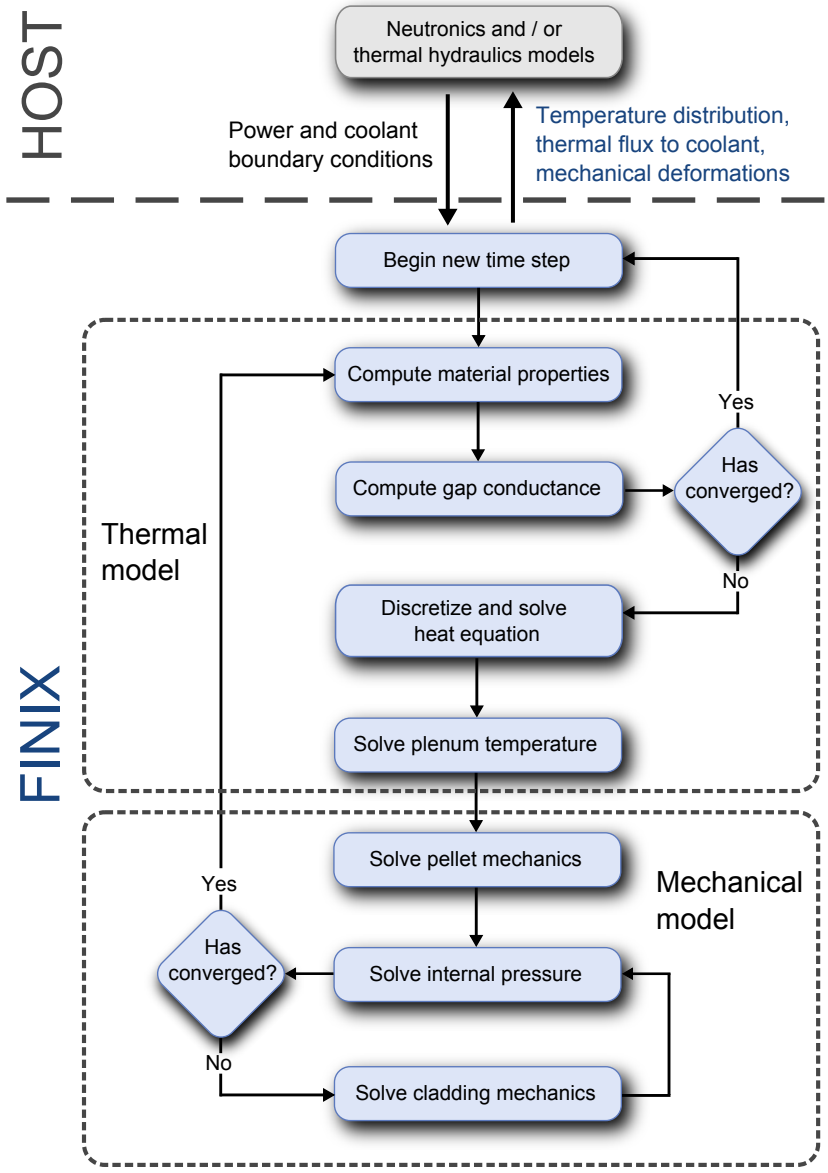


Figure 1. *Flowchart depicting the logic and execution order of the main FINIX modules and the interface between FINIX and the host code. Computation of a new time step begins from the top and proceeds through the modules as indicated by the arrows. Convergence checks are made for the gap conductance in the thermal model, and for the internal pressure in the mechanical model. On the first round of iteration, the convergence is always considered to have failed, so that the full thermo-mechanical model is executed at least once.*

temperature T is assumed to behave linearly as a function of r. The actual finite element equations are then derived by minimizing the residual (discretization error) between the original equations and the discretized equations. The minimization can be done with several different methods. A common choice is the Galerkin method, where the minimization is done by weighting the residual with the basis functions [22]. This is also the method we will use.

The temperature inside the i:th element is approximated with linear basis functions N_i and N_{i+1}

so that

$$T(r) \approx [N_i(r)\ N_{i+1}(r)] \begin{bmatrix} T_i \\ T_{i+1} \end{bmatrix}, \tag{150}$$

where the square brackets indicate row and column vectors, T_i is the temperature at the i:th node, and

$$N_i(r) = \frac{r_{i+1} - r}{r_{i+1} - r_i}, \tag{151}$$

$$N_{i+1}(r) = \frac{r - r_i}{r_{i+1} - r_i}. \tag{152}$$

For Eq. (149), the Galerkin method results in the matrix equation

$$\int \begin{bmatrix} N_i(r) \\ N_{i+1}(r) \end{bmatrix} C_V(r) \, [N_i(r) N_{i+1}(r)] \, dV \frac{\partial}{\partial t} \begin{bmatrix} T_i \\ T_{i+1} \end{bmatrix} - \begin{bmatrix} \lambda_{i,i} & \lambda_{i,i+1} \\ \lambda_{i+1,i} & \lambda_{i+1,i+1} \end{bmatrix} \begin{bmatrix} T_i \\ T_{i+1} \end{bmatrix} - \int \begin{bmatrix} N_i(r) \\ N_{i+1}(r) \end{bmatrix} s(r) dV = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \tag{153}$$

where the matrix elements $\lambda_{i,j}$ are

$$\lambda_{i,j} = -2\pi\Delta z \int_{r_i}^{r_j} r\lambda(r) \frac{\partial N_i(r)}{\partial r} \frac{\partial N_j(r)}{\partial r} dr + 2\pi\Delta z \Big|_{r_i}^{r_j} r\lambda(r) N_i(r) \frac{\partial N_j(r)}{\partial r}, \tag{154}$$

and the integral operator can be written as $\int dV = 2\pi\Delta z \int_{r_i}^{r_{i+1}} r dr$, since the integrands have no axial or azimuthal dependence in the slice of thickness Δz. We also linearize the heat capacity C_V, conductivity λ and the source term s within the element, so that, given the values at the nodes i and $i+1$ (denoted by subscripts), we have

$$C_V(r) \approx C_i + (C_{i+1} - C_i) \frac{r - r_i}{r_{i+1} - r_i}, \tag{155}$$

$$\lambda(r) \approx \lambda_i + (\lambda_{i+1} - \lambda_i) \frac{r - r_i}{r_{i+1} - r_i}, \tag{156}$$

$$s(r) \approx s_i + (s_{i+1} - s_i) \frac{r - r_i}{r_{i+1} - r_i}, \tag{157}$$

for $r_i \leq r \leq r_{i+1}$.

Integration over the element gives the matrix equation

$$(\mathbf{K}_i + \mathbf{C}_i) \begin{bmatrix} T_i^{k+1} \\ T_{i+1}^{k+1} \end{bmatrix} = \mathbf{C}_i \begin{bmatrix} T_i^k \\ T_{i+1}^k \end{bmatrix} + \mathbf{f}_i, \tag{158}$$

where time derivative has also been discretized with the implicit Euler method. The superscript of T indicates the time step of the time-discretized temperature, so that $T_i^k \equiv T(r = r_i, t = k\delta t)$, where δt is the time step. The implicit Euler method remains unconditionally stable with all values of δt [22, 23]. The matrices in Eq. (158) are defined as follows:

$$\mathbf{K}_i = \frac{\lambda_i(2r_i + r_{i+1}) + \lambda_{i+1}(r_i + 2r_{i+1})}{6(r_{i+1} + r_i)} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \tag{159}$$

$$\mathbf{C}_i = \frac{r_{i+1} - r_i}{60\delta t} \begin{bmatrix} c_i(12r_i + 3r_{i+1}) + c_{i+1}(3r_i + 2r_{i+1}) & c_i(3r_i + 2r_{i+1}) + c_{i+1}(2r_i + 3r_{i+1}) \\ c_i(3r_i + 2r_{i+1}) + c_{i+1}(2r_i + 3r_{i+1}) & c_i(2r_i + 3r_{i+1}) + c_{i+1}(3r_i + 12r_{i+1}) \end{bmatrix}, \tag{160}$$

$$\mathbf{f}_i = \frac{r_{i+1} - r_i}{12} \begin{bmatrix} s_i(3r_i + r_{i+1}) + s_{i+1}(r_i + r_{i+1}) \\ s_i(r_i + r_{i+1}) + s_{i+1}(r_i + 3r_{i+1}) \end{bmatrix} + \begin{bmatrix} -r_i q_i \\ r_{i+1} q_{i+1} \end{bmatrix}. \tag{161}$$

The second term in the vector $\mathbf{f}_i$ is determined by the boundary conditions of the element, with q_i the heat flux over the surface at $r = r_i$. The boundary conditions will be discussed in more detail below.

6.2.3 The gas gap element

For $i = \{1, 2, \ldots, n_f - 1, n_f + 1, n_f + 2, \ldots, n_f + n_c\}$, the element discretization is given by the 2×2 matrices derived in the previous Section. For the gas gap, *i.e.*, the n_f:th element, the matrices are derived from the gas gap conductance model described in Sec. 3.3. Given the heat transfer coefficient h, the matrix $\mathbf{K}_{n_f}$ is

$$\mathbf{K}_{n_f} = \begin{bmatrix} hr_{n_f} & -hr_{n_f} \\ -hr_{n_f} & hr_{n_f} \end{bmatrix}. \tag{162}$$

There is no power generated in the gap, and the specific heat of the gas is assumed negligible. Hence, $\mathbf{C}_{n_f} = \mathbf{0}$ and $\mathbf{f}_{n_f} = \mathbf{0}$.

6.2.4 Global matrices

The global matrices for the whole system are assembled from the 2×2 element matrices by taking the sum node by node. The result is an $(n_f + n_c) \times (n_f + n_c)$ tridiagonal matrix. For brevity, we introduce a shorthand notation of the element matrices:

$$\mathbf{K}_i \equiv \begin{bmatrix} \mathrm{K}_i^{(11)} & \mathrm{K}_i^{(12)} \\ \mathrm{K}_i^{(21)} & \mathrm{K}_i^{(22)} \end{bmatrix}, \; \mathbf{C}_i \equiv \begin{bmatrix} \mathrm{C}_i^{(11)} & \mathrm{C}_i^{(12)} \\ \mathrm{C}_i^{(21)} & \mathrm{C}_i^{(22)} \end{bmatrix}. \tag{163}$$

Then, the global matrices are of the tridiagonal form

$$\mathbf{K} = \begin{bmatrix} \mathrm{D}_1 & \mathrm{U}_1 & & & & \mathbf{0} \\ \mathrm{L}_1 & \mathrm{D}_2 & \mathrm{U}_2 & & & \\ & \mathrm{L}_2 & \mathrm{D}_3 & \mathrm{U}_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \mathrm{L}_{n-2} & \mathrm{D}_{n-1} & \mathrm{U}_{n-1} \\ \mathbf{0} & & & & \mathrm{L}_{n-1} & \mathrm{D}_n \end{bmatrix}, \tag{164}$$

where $n = n_f + n_c$ is the total number of nodes. The diagonal (D), upper diagonal (U) and lower diagonal (L) elements are given as

$$\mathrm{D}_i = \mathrm{K}_i^{(11)} + \mathrm{K}_{i-1}^{(22)} \text{ (for } 2 \leq i \leq n-1); \; \mathrm{D}_1 = \mathrm{K}_1^{(11)}; \; \mathrm{D}_n = \mathrm{K}_{n-1}^{(22)}, \tag{165}$$

$$\mathrm{U}_i = \mathrm{K}_i^{(12)} \text{ (for } 1 \leq i \leq n-1), \tag{166}$$

$$\mathrm{L}_i = \mathrm{K}_i^{(21)} \text{ (for } 1 \leq i \leq n-1). \tag{167}$$

The matrix $\mathbf{C}$ is assembled in a similar fashion, with the elements of the matrices $\mathbf{K}_i$ replaced with the elements of $\mathbf{C}_i$.

The load vector is given as

$$\mathbf{f} = \begin{bmatrix} \mathrm{f}_1^{(1)} \\ \mathrm{f}_1^{(2)} + \mathrm{f}_2^{(1)} \\ \mathrm{f}_2^{(2)} + \mathrm{f}_3^{(1)} \\ \vdots \\ \mathrm{f}_{n-2}^{(2)} + \mathrm{f}_{n-1}^{(1)} \\ \mathrm{f}_{n-1}^{(2)} \end{bmatrix}, \tag{168}$$

where $\mathrm{f}_i^{(1)}$ and $\mathrm{f}_i^{(2)}$ are the two components of $\mathbf{f}_i$:

$$\mathbf{f}_i \equiv \begin{bmatrix} \mathrm{f}_i^{(1)} \\ \mathrm{f}_i^{(2)} \end{bmatrix}. \tag{169}$$

Finally, the global matrix equation for the complete system is

$$(\mathbf{K} + \mathbf{C})\mathbf{T}^{k+1} = \mathbf{C}\mathbf{T}^k + \mathbf{f}, \tag{170}$$

where the vector $\mathbf{T}^k$ contains the temperatures at the k:th time step:

$$\mathbf{T}^k \equiv \begin{bmatrix} T_1^k \\ T_2^k \\ \vdots \\ T_n^k \end{bmatrix}. \tag{171}$$

6.2.5 Time discretization

Implicit time discretization of the heat equation is implied in Eq. (170). The time discretization follows the standard implicit finite difference discretization (see, *e.g.*, Ref. [23]), which remains unconditionally stable for all time steps δt. The implicit formulation means that to calculate the temperature at time $t = (k+1)\delta t$, the temperature-dependent terms of the heat equation are evaluated at the same point in time, at $t = (k+1)\delta t$. For constant (non-temperature-dependent) material properties, it is possible to solve the one-dimensional implicit time-discretized matrix equation, Eq. (170), without iteration. However, since the material correlations depend on temperature, and cannot in general be linearized, it is necessary to iterate the solution of $\mathbf{T}^{k+1}$, until the temperature converges. This is part of the iteration procedure described above in Sec. 6.1.

6.2.6 Boundary conditions

The boundary conditions affect the discretization of the elements at the center of the pellet and at the outer surface of the cladding. For the first element, the flux through the first node is set to zero, *i.e.*, $q_1 = 0$ in Eq. (161). For the cladding outer surface, several alternative boundary conditions can be used.

Dirichlet boundary condition For a fixed surface temperature, $T(R_{co}) = T_{\mathrm{surf}}$, the appropriate boundary condition is enforced by setting $\mathrm{L}_n = \mathrm{D}_n = 0$ for $\mathbf{C}$, $\mathrm{L}_n = 0$ and $\mathrm{D}_n = 1$ for $\mathbf{K}$, and $\mathrm{f}^{(2)}_{n-1} = T_{\mathrm{surf}}$ in the load vector $\mathbf{f}$. This is equivalent with the the equation $T^{k+1}_n = T_{\mathrm{surf}}$.

Neumann boundary condition If the heat flux across the cladding outer surface is fixed, then one only needs to assign q_n the desired value in $\mathbf{f}$.

Robin boundary condition The heat flux can also be given as a function of the bulk temperature of the coolant, T_{coolant}, and the heat transfer coefficient h_{co} so that $q_n = h_{co}\left[T_n - T_{\mathrm{coolant}}\right]$. In this case, the last element of $\mathbf{f}$ is $\mathrm{f}^{(2)}_{n-1} = r_n h_{co} T_{\mathrm{coolant}}$, and the remaining $r_n h_{co}$ is added to the last diagonal element D_n in the matrix $\mathbf{K}$.

In the case of the Robin boundary condition, the heat transfer coefficients can be computed from internal correlations (see Sec. 5.4) or, they be given by the user.

6.2.7 Solution of the matrix equations

The resulting matrix equation for the vector $\mathbf{T}^{k+1}$, Eq. (170), can be solved non-iteratively with the tridiagonal matrix algorithm, which a variant of the standard Gaussian elimination method. The algorithm is a standard numerical method, and its details are not explained here. The interested reader is referred to Chapter 2 of Ref. [5].

Although the heat equation itself can be solved non-iteratively, the dependence of the material properties, gap conductance and the pellet and cladding mechanical solution on the temperature requires iteration of the full thermo-mechanical solution. The scheme is described in more detail above, in Sec. 6.1.

6.3 Plenum temperature

The plenum temperature model described in Sec. 5.3 can be solved self-consistently for fixed plenum pressure and cladding and pellet surface temperatures. However, since the properties of the fill gas and thereby the heat transfer coefficients depend on the temperature, the equations do not have a closed form solution. Instead, the solution is found iteratively with the secant method [5]. The area-averaged temperature $T_0 = (A_p T_p + A_c T_c)/(A_p + A_c)$ is used as the initial guess. Typically, the iteration requires two steps or less to converge within the numerical tolerance.

6.4 Mechanical model

6.4.1 Rigid pellet model

The primary assumption of the rigid pellet approximation is that the pellet is rigid (hard) enough to be completely non-deformable under external stresses. Thus, the displacement of the pellet nodes and the outer surface can be calculated directly from the strain correlations, without iteration.

6.4.2 Cladding model

The employed cladding mechanical model depends on the type of contact between the pellet and the cladding. In principle, iteration of the mechanical solution with internal pressure is only necessary when the gap remains open; if the gap is closed, the gap volume is fixed and therefore does not affect the pressure. However, the model typically consists of several axial nodes, and the pressure is function of the displacements in all of those nodes. In some of the nodes, the gap may remain open while in others it is closed (due to, *e.g.*, nonuniform axial power distribution). If the gap remains open even in some of the nodes, the internal pressure has to be solved iteratively. This iteration process then changes the boundary conditions in the closed gap model, which has to be re-solved. Therefore, even the closed gap models are solved several times because of the iteration of the mechanical solution of the full rod. Note that the plastic strain increments calculated on each iteration are only saved to the total plastic strains after the whole solution has converged (at the end of the time step).

In practice, the iteration is performed by first calculating an initial guess for the pressure (using the displacements from, *e.g.*, previous time step), then solving the mechanical model for each axial node independently with fixed pressure, and finally re-calculating the pressure with updated displacements. The process is repeated, using the secant method to predict the pressure, until the internal pressure converges.

7 Usage instructions

7.1 General description

The FINIX fuel behavior model and interface is designed to work primarily as a subprogram within a larger simulation code, to provide or replace the existing fuel performance model or subroutines. In this document, this larger code is referred to as the *host code*.

The purpose of FINIX is to provide the host code all the required fuel behavior subroutines via an interface that integrates directly with the host code on the source code level, and only requires a limited number of function calls between FINIX and the host. All the data between the host code and FINIX is passed as arguments of the function calls; no input or output files for the data exchange are needed. This makes execution of the subroutines faster, because disk access is minimized, and also allows the user of the host code to specify which output from FINIX (if any) is saved as a file.

Although the communication between FINIX and the host code is handled without input/output files, since FINIX 0.13.9 it has been possible to initialize FINIX for accumulated burnup using FRAPCON-generated FRAPTRAN restart files. Since version 0.15.6 it has also been possible to initialize FINIX data structures by using input files.

FINIX also has an error message system, which is designed to provide the host code run-time errors and warnings. For example, convergence failures or invalid correlation parameters would be passed as an error message to the host code. A more detailed description of the error message system is given below.

The setup and usage of the FINIX model is done via built-in functions, which can be used to provide default values for system parameters, give the spatial discretization and, to solve the thermo-mechanical model. A more thorough walk-through of the procedure is given below, in Sec. 7.5. The source code also includes an example file, *host.c*, which provides FINIX with the necessary run-time data and shows how the model is initialized and run.

A more detailed documentation of the source code, with function descriptions and dependencies, is also given in Appendix A of this document.

7.2 Units

FINIX functions always use the base SI units in both input arguments and in output. For example, distances are given in meters (m), temperatures are given in Kelvin (K) and power, linear power, and power density are given in W, W/m and W/m^3, respectively. The user is responsible for writing any unit conversion functions between the host code and FINIX.

In general, the data in FINIX input files should be given in SI units as well. However, time-related data can be given in seconds, hours or days. If the input will be read from a FRAPTRAN input file, it is possible to choose either British or SI units.

In FINIX, the term "power" refers primarily to thermal power, as opposed to fission power. Currently the code makes no distinction between the two. However, in principle the user should always supply FINIX with the *thermal* power history, including the decay heat of the fission products.

The cold state in FINIX specifies the reference temperature for the fill gas properties and thermal strains. The cold state is defined as 300 K.

7.3 Data structures

The main purpose of FINIX is to provide the host code the temperature distribution and spatial deformations of the pellet and cladding by using the power history and coolant conditions provided by the host code as input. In addition to these primary data, there are data that are either required by FINIX as parameters, or are generated by solution of the thermo-mechanical model. Although much of this may be irrelevant to the host code, this data is required to continue the FINIX simulation for several consecutive time steps. All the data that FINIX uses or produces is stored in five structures. These structures are described below. **A more detailed code documentation is given in Appendix A.**

The structure called **Rod** contains the data that describes the physical properties of the fuel rod. These properties include rod dimensions, pellet properties, cladding properties, gas properties and bundle dimensions. This data will not change during simulation. The contents of the Rod structure can be read from the corresponding input file.

The **Boundary_conditions** structure is used for storing the boundary condition data needed by FINIX. This data includes variables that specify the simulation time, power data, and the data related to cladding and coolant conditions. For each FINIX calculation time step, the Boundary_conditions structure needs to be updated either by the host code, or in the FINIX main program

(in *host.c*).

The **Scenario** structure contains power profile and irradiation history data that will be read from an input file if needed. Once this data has been read from an input file, it will not change during simulation. The history data from the Scenario structure should be transferred to Boundary_conditions structure at each time step. If the host code provides the boundary condition data, the Scenario structure is not needed. The contents of the Scenario structure can be read from the corresponding input file.

The **Results** structure contains all the simulation results calculated by FINIX.

The **Options** structure contains the variables that specify the nodalization and the calculation options. This data will not change during simulation. The contents of the Options structure can be read from the corresponding input file.

The **Cylindrical** structure is used in storing values of results in each component in cylindrical coordinates, that is, values for strains and stresses in the mechanical model. Some members of the **Results** structure are Cylindrical structures. The **Cylindrical** structure contains members hoop, axial and radial, and functions for easy manipulation of these structures are implemented. With these functions one can, for example, the copy the values of structures into another or perform simple calculations with them.

7.4 Error message system

FINIX uses an error message system to inform the user of issues such as convergence failures, parameters exceeding correlations' range of validity, etc. The function that encounters the issue returns the pointer to the error message. The message is then passed down, and further error messages are appended to it, until the message reaches the host code. It is then the responsibility of the host code to act on the error message by printing it on screen, writing it on disc, aborting the program execution, or otherwise. FINIX will not terminate the execution, if an issue is encountered. Only the error message will be returned.

The format of the error message is an array of text strings, formally of type char**. If no errors have occurred, then the top-level pointer will have a value of NULL. This can be used to distinguish between error-free and faulty operation of the FINIX functions. If the value is different from NULL, then the return value contains an error message. The returned array then contains pointers to the error message strings (of type char*). The last message is followed by a NULL pointer, which is used as a marker to terminate the message.

In practice, the user can use built-in functions to deal with FINIX the error messages. One merely needs to declare the pointer in the host code. For example, the following lines will declare the pointer, then call the FINIX transient solver function, catch the error message, print it on screen if it is not empty (*i.e.*, non-NULL), and finally free the memory allocated to the error message.

```
char **err=NULL;

err=finix_solve_transient(dt,nnodes,T,r,r_cold,power_dist, linear_power,
                  burnup,params,bcond,sresults,vresults,options);
if(err!=NULL){ finix_printf_err(err);}
finix_free_err(&err);
```

7.5 System setup and simulation

To use the FINIX code in a coupled system, the user must declare the necessary data structures in the host code, initialize the FINIX structures, and calculate the initial state of the simulation by using FINIX functions. After this the model can be run for as many time steps as needed. FINIX comes with an example host code file, *host.c*, which should be replaced in its entirety by the host code. The *host.c* file can be used as an example on how to setup the model. The minimal necessary steps to get FINIX up and running is described in this chapter.

In a coupled system the user is responsible for using an appropriate time step. Although the FINIX algorithms remain stable for very long time steps (longer than several days), discretization error can not be avoided. For relatively slow transients, a time step of the order of 1 millisecond should give very small discretization error during the transient, although for a fast RIA, a considerably smaller time step ($\delta t \approx 10^{-5}$ s) may be needed.

The declaration and the definition of the variables can be done as follows:

```
// Declare and construct FINIX data structures
Rod *rod = finix_rod_construct();
Boundary_conditions *bc = finix_bc_construct();
Scenario *scenario = finix_scenario_construct();
Results *results = finix_results_construct();
Options *options = finix_options_construct();

// Declare FINIX error message strings
char **finix_err=NULL, **new_err=NULL;
```

The next step is to initialize the system. The FINIX data structures can be initialized by calling the function *finix_initialize_data_structures()*. The function reads the input files given by the user, and transfers the data from the input files to the data structures. If no input files are given or some of the input is missing, the above constructor functions initialize the missing data with default values.

In many cases the transient begins from a steady state, which has to be solved before calculating the transient scenario. For this purpose FINIX has the function *finix_solve_initial_steady_state()*, which solves the steady state heat equation and the corresponding mechanical equilibrium for the cladding.

FINIX can be initialized as follows:

```
// Initialize FINIX data structures
finix_err = finix_initialize_data_structures(rod, bc, scenario, results, options);
if (finix_err != NULL) finix_printf_err(finix_err);
finix_free_err(&finix_err);

// Solve initial steady state
finix_err = finix_solve_initial_steady_state(rod, bc, results, options);
if(finix_err != NULL) finix_printf_err(finix_err);
finix_free_err(&finix_err);
```

After the initial state has been solved, the transient simulation can begin. The transient is solved in distinct time steps defined in the host code, by calling the *finix_solve_transient()* function. Before each function call, the power density, boundary conditions and other parameters and options can

be changed. However, for one time step, *i.e.*, for one function call, they are constant. After each function call, the output of the FINIX model may be saved. For example, one can write FINIX summary files and node-spesific files by calling a function *finix_output_print()*.

The transient can be solved for each time step as follows:

```
// Solve time step with finix_solve_transient()
finix_err=finix_solve_transient(bc->dt, rod, bc, results, options);
if (finix_err != NULL) finix_printf_err(finix_err);
finix_free_err(&finix_err);

// Update the cumulative simulation time
bc->time += bc->dt;
```

When the transient simulation has been completed and FINIX is no longer needed, the memory allocated for FINIX structures must be free'd. This can be done by using the function *finix_data_structures_destruct()*.

```
// Free allocated memory
finix_data_structures_destruct(rod, bc, scenario, results, options);
```

As mentioned before, the above lines of code are the minimal setup that is needed to run FINIX in a coupled system. However, if the host code cannot provide all the necessary boundary conditions, the missing boundary conditions must be given in FINIX input file finix_scenario.inp. The boundary conditions must be updated for each time step by calling the function finix_update_bc() as follows:

```
// Update boundary conditions before calling finix_solve_transient()
finix_err = finix_update_bc(bc, scenario, options, rod, results);
if(finix_err != NULL) finix_printf_err(finix_err);
finix_free_err(&finix_err);
```

7.6 Input instructions

FINIX distribution includes three input files called finix_options.inp, finix_rod.inp, and finix_scenario.inp. These input files can be used to control the simulation. Each input file contains data for several fuel rods. When a new input is created for a new fuel rod, the input should be appended at the end of the files. An alternative way to control the simulation is to use FRAPTRAN input files.

In a coupled system, where all boundary conditions are given by the host code, no input files are required. IIf no input files are used, default values for simulation options and rod properties will be used. However, it is advisable to specify simulation options and rod properties in files finix_options.inp and finix_rod.inp (or otherwise in the host code) to get more realistic results. If the host code cannot provide all the necessary boundary conditions, i.e., the contents of the Boundary_conditions structure, the finix_scenario.inp file must be used to specify the missing boundary conditions.

If a stand-alone version of FINIX is used, the scenario boundary conditions must be given in file finix_scenario.inp. Again, it is advisable but not mandatory to give the simulation options and rod properties in files finix_options.inp and finix_rod.inp. Alternatively, input can be given in a FRAPTRAN input file. However, even though the input is given in a FRAPTRAN input file, it

is advisable to specify FINIX simulation options in finix_options.inp because FINIX simulation options cannot be set in a FRAPTRAN input file. All the data given in FRAPTRAN input file overwrites the default data and the data read from FINIX input files.

The contents of FINIX input files is organized in data blocks. Each block of data describes one fuel rod. Therefore FINIX needs to know which block of data to read from a file. The data blocks are of the following form:

```
begin rod_name1
data_1 = x
data_2 = y
data_n = z
end rod_name1
```

As can be seen above, every data block begins with "begin" statement and ends with "end" statement. These statements are followed by the name of the rod. Note that the rod name must not include white spaces. As the data in input files has been grouped into blocks that describe each fuel rod, one should tell FINIX from which data block to read the data. This can be done with the "USE" statement as follows:

```
USE rod_name1

begin rod_name1
data_1 = x
data_2 = y
data_n = z
end rod_name1

begin rod_name2
data_1 = x
data_2 = y
end rod_name2
```

The "USE" statement tells FINIX that it should read the data from a data block that begins with "begin rod_name1" and ends with "end rod_name1". Every FINIX input file should begin with the "USE" statement.

The data in input files can be arranged in any order. After a keyword, there should be an "=" sign followed by the data that corresponds to the keyword. If the data consists of several values, they should be separated with commas. The lines in input files that begin with "!", "*", or "/" are commented lines, and FINIX will ignore them. The contents of the three input files is described in the following chapters.

7.6.1 Input file finix_options.inp

Input file finix_options.inp can be used to set FINIX nodalization and to select the models FINIX uses during simulation. Table 4 shows the data that can be given in finix_options.inp. If default value will be used, it is not necessary to add that data in the input file. The contents of a file could

look like this:

```
USE rod_123

begin rod_123
axial_nodes = 10
pellet_radial_nodes = 15
boundary_option = 0
end rod_123
```

Now the rod will be divided to 10 axial nodes and the fuel pellet will be divided to 15 radial nodes. The rod outer surface temperature will be given by the user. For the rest of the keywords default values shown in Table 4 will be used.

Input file finix_options.inp has also a special purpose. It can be used to tell FINIX that the input will be read from a FRAPTRAN input file. In this case the the first line of the file should read "USE FRAPTRAN". When FINIX simulation begins, it asks the user the name of the FRAPTRAN input file and the name of the FRAPCON generated restart file. If FRAPTRAN input file will be read, the contents of the finix_options.inp could look like this:

```
USE FRAPTRAN
```

or like this:

Table 4. *Data that can be given in finix_options.inp*

Keyword	**Units**	**Default**	**Description**
axial_nodes	-	11	Number of axial nodes.
pellet_radial_nodes	-	17	Number of radial nodes in fuel pellet.
clad_radial_nodes	-	5	Number of radial node in the cladding.
boundary_option	-	3	Specifies the type of the boundary conditions. 0 = user-given rod outer surface temperature, 1 = user-given heat flux between rod outer surface and coolant, 2 = user-given heat transfer coefficient and coolant bulk temperature, 3 = user-given bulk temperature and calculated heat transfer coefficient from internal correlations (needs inlet mass flux).
temperature_iteration	-	1	Temperature iteration. 0 = off, 1 = on.
gap_conductance_model	-	2	Gap conductance model. 1 = FRAPCON model, 2 = FRAPTRAN model, -n = negative value switches off contact conductance
clad_elasticity_model	-	1	Clad elasticity model. 0 = off, 1 = on.
plenum_model	-	1	Plenum model. 0 = off, 1 = on.
pellet_relocation_model	-	1	Pellet relocation model. 0 = off, 1 = on.
clad_plasticity_model	-	1	Cladding plastic deformation model. 0 = off, 1 = on.

```
USE FRAPTRAN

begin FRAPTRAN
axial_nodes = 10
pellet_radial_nodes = 15
boundary_option = 0;
end FRAPTRAN
```

In the first case all the data will be read from a FRAPTRAN input file. In the second case nodalization and boundary option will be given in finix_options.inp. If nodalization is also given in FRAPTRAN input file, the data given in FRAPTRAN input file overwrites the data given in finix_options.inp. Note that the variables that specify FINIX model selections should always be given in finix_options.inp. If the input will be read from the FRAPTRAN input file, input files finix_rod.inp and finix_scenario.inp will be ignored.

7.6.2 Input file finix_rod.inp

Input file finix_rod.inp can be used to determine the fuel rod properties. Table 5 shows the data that can be given in finix_rod.inp. The contents of a file could look like this:

```
USE rod_123

begin rod_123
fuel_length = 4.0
clad_length = 4.0
plenum_length = 0.4
end rod_123
```

In this case default values will be used for all the parameters except for fuel, clad, and plenum length.

7.6.3 Input file finix_scenario.inp

Input file finix_scenario.inp can be used to determine the rod conditions during an irradiation or experiment period. As mentioned before, this data must be given if FINIX is used as a stand-alone version, or the host code cannot provide all the necessary boundary conditions. Table 6 shows the data that can be given in finix_scenario.inp. Note that it is not necessary to give all the data presented in Table 6 to fully determine the boundary conditions. For example, if the user has chosen to use coolant bulk temperature as boundary condition (boundary_option = 3), it is not necessary to determine the clad temperature history in finix_scenario.inp. The contents of the file finix_scenario.inp could look like this:

```
USE rod_xyz

begin rod_xyz
restartfile = imprestart.rod_xyz
end_time = 1.0
```

Table 5. *Data that can be given in finix_rod.inp*

Keyword	**Units**	**Default**	**Description**
pellet_inner_radius	m	0.0	Radius of fuel pellet center hole.
pellet_outer_radius	m	0.0047	Pellet outer radius.
clad_inner_radius	m	0.00479	Fuel cladding inner radius.
clad_outer_radius	m	0.00546	Fuel cladding outer radius.
fuel_length	m	3.6576	Length of rod fuel column.
clad_length	m	3.6576	Length of fuel cladding wall in active area.
plenum_length	m	0.2	Plenum length.
pellet_roughness	m	2.0e-6	Arithmetic mean roughness of fuel pellet surface.
fractional_density	-	0.938	Fractional theoretical density of fuel pellet.
gadolinia_weight_fraction	-	0.0	Weight fraction of gadolinia (Gd_2O_3) in fuel pellets.
clad_roughness	m	0.5e-6	Arithmetic mean roughness of cladding inner surface.
coldwork	-	0.5	Reduction of cross-sectional area of cladding by cold working process.
clad_oxygen_concentration	-	0.0012	Cladding average oxygen concentration.
fast_neutron_fluence	n/m^2	0.0	Fast neutron fluence that the cladding was exposed to during lifetime.
clad_type	-	0	Cladding material identifier. 0 = Zircaloy, 1 = Zr1%Nb.
fill_gas_pressure	N/m^2	1.207e6	As-fabricated fill gas pressure.
fill_gas_temperature	K	300.0	As-fabricated fill gas temperature.
gas_fraction_He	-	1.0	Fraction of gas that is helium.
gas_fraction_Ar	-	0.0	Fraction of gas that is argon.
gas_fraction_Kr	-	0.0	Fraction of gas that is krypton.
gas_fraction_Xe	-	0.0	Fraction of gas that is xenon.
gas_fraction_H2	-	0.0	Fraction of gas that is hydrogen.
gas_fraction_N2	-	0.0	Fraction of gas that is air.
gas_fraction_H2O	-	0.0	Fraction of gas that is water vapor.
pitch	m	14.43e-3	Center-to-center spacing of fuel rods.
rods_in_unit_cell	-	1.0	Number of rods in one unit cell (1.0 for square lattice, 0.5 for triangular lattice).

```
time_step_history =
0.0001, 0.0,
0.00001, 0.0702,
0.0001, 0.0815,
0.001, 0.210

power_history =
0.0, 0.0,
524934.384, 0.066,
23622047.28, 0.079,
5511811.032, 0.087,
262467.192, 0.095,
0.0, 0.1

axial_power_profile =
0.728, 0.0,
0.975, 0.099,
1.156, 0.259,
1.053, 0.420,
0.668, 0.569

coolant_temperature_history_zones = 0.373, 0.569
coolant_temperature_history(1) =
553.15, 0.0,
1023.15, 0.2,
1048.15, 0.4,
753.15, 1.0
coolant_temperature_history(2) =
553.15, 0.0,
878.15, 0.2,
923.15, 0.4,
948.15, 1.0

heat_transfer_coefficient_history_zones = 0.569
heat_transfer_coefficient_history(1) = 2000000.0, 0.0

coolant_pressure_history = 0.5e6, 0.0
end rod_xyz
```

As can be seen in the example above, the history data is always given in data pairs. For example, the time step history is given so that the first value of the data pair is the length of the time step, and the second value is the time at which this time step length takes effect. This time step length is used until a new data pair is given. Similar logic applies to other history data as well.

All time-related FINIX input can be given in seconds, hours or days. The units of the time can be selected by modifying the value of time_unit in finix_scenario.inp. Note that all the time-related data in Table 6 is expressed in seconds (default) only for clarity. The time units of the data shown

in Table 6 depends on the value of time_unit. The value of time_unit also determines the units of the time in output files finix.z* and finix.sum.

In the example above one can also see that some of the history data, such as the coolant temperature history, can be given separately for different axial zones. In this case the coolant temperature history has been given for axial zones 1 and 2. The number of zones does not have to match the number of axial nodes given in finix_options.inp. FINIX will automatically calculate history data for each axial node based on the data given for each zone. Note that one should also determine the top elevations of each zone from the rod bottom. In the example above, the top of the fist coolant temperature history zone is 0.373 m above the rod bottom, while the top of the second zone is 0.569 m above the rod bottom.

Table 6. *Data that can be given in finix_scenario.inp*

Keyword	Units	Default	Description
restartfile	-	-	Name of the Frapcon generated restart file (omit if there is no need to read a restart file).
steady_state_simulation	-	0	Type of the scenario. 0 = transient, 1 = steady-state.
time_unit	-	0	Units of all time-related FINIX input. Determines also the units of time in output files finix.z* and finix.sum. 0 = seconds, 1 = hours, 2 = days.
end_time	s	0.15	Simulation end time.
time_step_history	s, s	0.0001, 0.0	Time step history. The first value is the size of the timestep. The second value is the time at which this time step size takes effect. Each time step size is used until a new data pair is given.
power_factor	-	1.0	Multiplier for linear power.
power_history	W/m, s	0.1e6, 0.0	Rod average linear heat generation rate history. Each linear heat generation rate is used until a new data pair is given.
axial_power_profile	-, m	-	Axial power profile. The first value is the axial power factor normalized to rod-average. The second value is the node top elevation beginning from the rod bottom. Begin insering data pairs from the rod bottom towards the top, until the the axial power profile is fully defined.
radial_power_profile	-, m	-	Radial power profile for all axial nodes. The first value is the radial power factor. The second value is the distance from the fuel centerline to the radial node periphery. Begin inserting data pairs from fuel centerline to the edge.
coolant_temperature_ history_zones	m	-	Top elevation of each coolant temperature history zone. Enter as many values as there will be coolant temperature history zones.
coolant_temperature_ history(n)	K, s	561.0, 0.0	Coolant temperature and time data pairs for each coolant temperature history zone. Each temperature will be used until a new data pair is given. Enter as many data sets as there are coolant temperature history zones. coolant_temperature_history(1) starts input for zone 1, coolant_temperature_history(n) starts input for zone n.

Table 6 (continued). *Data that can be given in finix_scenario.inp*

clad_temperature_history_zones	m	-	Top elevation of each clad temperature history zone. Enter as many values as there will be clad temperature history zones.
clad_temperature_history(n)	K, s	561.0, 0.0	Clad temperature and time data pairs for each clad temperature history zone. Each temperature will be used until a new data pair is given. Enter as many data sets as there are clad temperature history zones. clad_temperature_history(1) starts input for zone 1, clad_temperature_history(n) starts input for zone n.
heat_transfer_coefficient_history_zones	m	-	Top elevation of each heat transfer coefficient history zone. Enter as many values as there will be heat trasfer coefficient history zones.
heat_transfer_coefficient_history(n)	W/(m^{2}K), s	2e4, 0.0	Heat transfer coefficient and time data pairs for each heat transfer coefficient history zone. Each heat transfer coefficient will be used until a new data pair is given. Enter as many data sets as there are heat transfer coefficeint history zones. heat_transfer_coefficient_history(1) starts input for zone 1, heat_transfer_coefficient_history(n) starts input for zone n.
heat_flux_history_zones	m	-	Top elevation of each heat flux history zone. Enter as many values as there will be heat flux history zones.
heat_flux_history(n)	W/m^2, s	0.0, 0.0	Heat flux (between rod outer surface and coolant) and time data pairs for each heat flux history zone. Each heat flux will be used until a new data pair is given. Enter as many data sets as there are heat flux history zones. heat_flux_history(1) starts input for zone 1, heat_flux_history(n) starts input for zone n.
coolant_pressure_history	N/m^2, s	15.51e6, 0.0	Enter coolant pressure and time data pairs. Each value of pressure is used until a new data pair is given.
coolant_mass_flux_history	kg/(m^{2}s), s	3460.0, 0.0	Enter coolant mass flux and time data pairs. Each value of mass flux is used until a new data pair is given.

7.7 Output files

FINIX includes functions that can be used to print output files for many purposes. One aim of this chapter is to describe the contents of these output files. In addition, this chapter shows how to call the output writing functions from the host code.

7.7.1 finix.sum and finix.z*

The data describing the behaviour of a fuel rod as a function of time is presented in output files finix.sum and finix.z*.The contents of these files is presented in Tables 7 and 8. Output file finix.sum contains rod summary data, while output files finix.z* contain node-specific data. Node-specific output files are written for each axial node. Asterisk (*) in the aforementioned file names represents the node number.

By default, the cumulative simulation time in files finix.sum and finix.z* is given in seconds. However, the simulation time in the aforementioned files will be given in hours or days if a non-default value is given for keyword "time_unit" in finix_scenario.inp. In other words, the same units of time are used in files finix_scenario.inp, finix.sum and finix.z*.

To print these output files, the following additional lines of code must be included in the host code:

```
// Declare a structure containing file pointers to output files
Output *files;

// Initialize output writing to finix.sum and finix.z files
files = finix_output_initialize(options);
```

In the initialization call above the type of the function parameter is Options*. After declarations and initializations the output writing function can be called for each time step as follows:

```
finix_output_print(files, rod, bc, options, results);
```

Here the types of the function parameters are Output*, Rod*, Boundary_conditions*, Options*, and Results*. After the output files have been printed, they must be closed and the memory must be free'd. This can be done as follows:

```
// Close files and free allocated memory
finix_output_close(files, options);
```

7.7.2 finix_data_structures.dbg

Output file finix_data_structures.dbg shows all the data stored in FINIX data structures at the time of the output writing call. The function is called during FINIX initialization by default, but it can be called later again if needed. The output file is especially useful for checking that the input files have been read corretly. The output file finix_data_structures.dbg can be printed by calling a funtion

```
finix_print_data_structures(rod, bc, scenario, results, options);
```

Table 7. *Contents of the output file finix.sum*

Column name	Units	Description
Step	-	Time step number.
Time	s or h or d	Cumulative simulation time.
Buav	MWd/kgU	Rod average burn-up.
Qav	W/m	Average linear power.
FGR%av	%	Percentage value of released fission gases. NOTE: not supported yet.
Intpr	N/m^2	Rod fill gas pressure pressure.
Coolpr	N/m^2	Coolant pressure.
Fuext	m	Fuel axial elongation.
Clext	m	Cladding axial elongation.
Tplen	K	Fill gas temperature.
Tzon	-	Axial node where maximum temperature occurs.
Qlo	W/m	Maximum local linear power.
BUlo	MWd/kgU	Maximum local burn-up.
Tmax	K	Maximum temperature.
FGR%lo	%	Maximum local fission gas release. NOTE: not supported yet.
Gap	m	Average pellet-cladding gap width.
Gapcon	$W/(m^2K)$	Gap average conductance.
Tclav	K	Clad average temperature.
Tcool	K	Coolant average temperature.

where the function parameter types are Rod*, Boundary_conditions*, Scenario*, Results*, and Options*.

7.7.3 finix_stripfile.txt

Output file finix_stripfile.txt contains data in FRAPTRAN stripfile format. The file describes the state of the fuel rod at various time steps.

To print this output file, the following additional lines of code must be included in the host code:

```
// Declare a variable for printing the results
FILE *writefile;

// Open the file
writefile = fopen("finix_stripfile.txt","w");

// Print the header lines
finix_db_fprintf_stripfile(0, writefile, bc->time, results, rod, options, bc, scenario);
```

After these steps the output writing function can be called for each time step as follows:

```
finix_db_fprintf_stripfile(1, writefile, bc->time, results, rod, options, bc, scenario);
```

After the file has been printed, it must be closed:

Table 8. *Contents of the output file finix.z**

Column name	Units	Description
Step	-	Time step number.
Time	s or h or d	Cumulative simulation time.
Burnup	MWd/kgU	Average fuel burn-up in node.
Linrat	W/m	Node linear power.
Tcool	K	Coolant temperature at the node elevation.
Tclou	K	Temperature at the cladding outer surface.
Tclav	K	Cladding average temperature.
Tclin	K	Temperature at the cladding inner surface.
Tfout	K	Temperature at the pellet surface.
Tfav	K	Pellet average temperature.
Tcent	K	Fuel centerline temperature.
FGR%lo	%	Fission gas release. NOTE: not supported yet.
Gap	m	Pellet-cladding gap width in node.
DTgap	K	Temperature difference over gap.
Gapcon	$W/(m^2K)$	Gap conductance.
Conpr	N/m^2	Pellet-cladding contact pressure.
Hoopstrs	N/m^2	Clad hoop stress.
Dradcl	m	Change in cladding radius.
Buav	MWd/kgU	Rod average burn-up.

```
fclose(writefile);
```

8 Code assessment

8.1 General performance

The validation of FINIX-0.13.9 is presented in a separate validation report [1], where the detailed results can be found. Version 0.15.12, where the cladding mechanical model has been changed from the previous version, will be validated in the future. Some remarks on the new plastic deformation model in version 0.15.12 and a brief summary of FINIX-0.13.9 validation are given below.

In figure 2 the plastic hoop strain calculated by FINIX in the CABRI REP-Na3 scenario is compared to FRAPTRAN. Note that the plastic strain reference point in FINIX is the unirradiated rod, whereas in FRAPTRAN it is the irradiated rod before the transient. In table 9 the calculated plastic strains in both FINIX and FRAPTRAN are compared to experimental values for some scenarios.

In failed rods FINIX may calculate higher plastic deformations than FRAPTRAN, as no burst model is implemented in FINIX. At the time of burst the internal pressure in FRAPTRAN is set to equal the coolant pressure, and according to the FRAPTRAN model the deviatoric stresses — the driving force of the deformation — are then zero. In intact rods FINIX calculates slightly smaller plastic deformations than FRAPTRAN, even though the same stress-strain correlation is

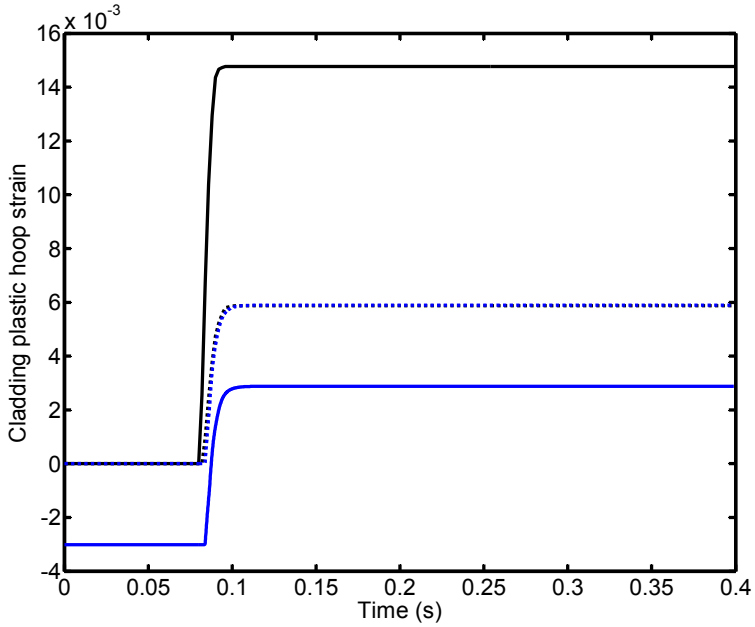


Figure 2. *Plastic hoop strain in CABRI REP-Na3 scenario as calculated by FRAPTRAN (black lines) and FINIX (blue lines). Black solid line corresponds to FRAPTRAN-1.5 as is, whereas the dotted line is FRAPTRAN-1.5 without the pellet RIA thermal expansion model. The solid line is FINIX-0.15.12 as is, and the dotted line is the same result normalized to give zero strain at the beginning to better visualize the comparison to FRAPTRAN-1.5. The dotted lines overlap, but have small differences.*

Table 9. *Maximum plastic hoop strains in some scenarios as determined by experiment or as calculated by FRAPTRAN-1.5 (without the pellet RIA thermal expansion model) or FINIX. Since FINIX has no criteria for rod failure, the strain in the BIGR scenarios would continue to increase if the calculation was continued. Also, FRAPTRAN does not continue the plastic strain calculation after the moment of rod failure.*

	Experiment		FRAPTRAN without pellet RIA thermal expansion model		FINIX
Scenario	Rod failure	Max. residual cladding hoop strain	Rod failure	Max. plastic cladding hoop strain	Max. plastic cladding hoop strain
CABRI REP-Na1	yes	n/a	yes	0.06%	0.631%
CABRI REP-Na3	no	2.2%	no	0.588%	0.589%
NSRR HBO-1	yes	much less than 1%	no	0.009%	0.027%
NSRR FK-1	no	0.3-0.85 %	no	0.260%	0.084%
BIGR RT-4	no	5.5%	yes	0%	1.345%
BIGR RT-8	yes	11.1%	yes	0%	57.932%

used. This is partially because of the slightly smaller fuel centerline temperatures calculated by FINIX: when the gap is closed, the cladding axial strain calculation is based on fuel axial thermal strain at the pellet centerline. The magnitude of this strain has a significant effect on the calculated plastic strain. Another effect lowering the calculated plastic strain is of the fast neutron fluence read in from the FRAPCON restart file. It is not clear how FRAPTRAN uses this value in the calculation of the yield stress correlation coefficients, and the yield stress calculated by FINIX seems to be higher than that output by FRAPTRAN. If the fast neutron fluence is not read in into FINIX, the yield stress values used by FINIX and output by FRAPTRAN are almost the same.

FINIX-0.13.9 has been compared against experimental centerline temperature data from Halden steady state irradiation experiments IFA-429 and IFA-432. The agreement between simulated and experimental results is good. For low burnup, the results match very well. With increasing burnup, the match becomes worse, although FINIX and the experimental value typically agree within roughly 100 K, with FINIX having a slight tendency to underestimate the centerline temperature. The poorer performance at higher burnups is expected, as FINIX does not have models to describe many of the burnup-dependent phenomena, such as fuel densification and swelling, cladding creep and fission gas release. Although these can be taken into account when initializing FINIX for transient calculations, simulating their behavior during long-term irradiation is not possible in the present version.

FINIX-0.13.9 was also validated against FRAPTRAN simulations of selected reactivity initiated accidents. The results for FRAPTRAN are described in the FRAPTRAN code assessment document [24], while the FINIX results and the comparison are discussed in the FINIX-0.13.9 validation report [1]. The cases consisted of RIA's with western and VVER type fuel rods, some of which had failed during the experiment, and some had not. In some of scenarios significant plastic deformation of the cladding was indicated by FRAPTRAN, while in some very little permanent deformation occurred. All the cases were initialized for non-fresh fuel using FRAPCON for steady state irradiation.

The comparison between FINIX and FRAPTRAN shows very good agreement between the codes. In almost all the cases, the fuel and cladding temperatures are very closely reproduced. Even in the cases where the cladding deforms plastically, FINIX succeeds in calculating the temperatures with good accuracy, although the differences in the gap dimensions and conductance are clearly seen. In addition, the cases where the rod fails are calculated very similarly up to the point of failure, after which FRAPTRAN switches to a different model. FINIX-0.13.9 has no criteria to determine rod failure, and therefore the calculation proceeds somewhat differently from the FRAPTRAN solution.

According to the validation results, the gap conductance model of the FRAPTRAN showed the best performance, both against FRAPTRAN (obviously) and the experimental Halden data. The FRAPTRAN gap conductance correlation is therefore recommended. It has also been set as the default model of FINIX-0.13.9. It should be noted that the FRAPTRAN correlation also ignores the soft relocation part of the FRAPCON relocation model.

8.2 Solved issues from previous versions

The axial thermal expansion of the pellet was previously underestimated by FINIX. The cause was that the axial thermal expansion of the pellet was calculated from the edge node (presumably due

to strong pellet dishing), while the central node (assumed negligible dishing) would give better results. In FINIX-0.15.12, the pellet axial thermal expansion is calculated from the central node.

A bug in the material density calculation was fixed. Previously the density was underestimated and this affected also the temperature calculations. As a result of the fix, predicted fuel centerline temperatures are higher by approximately 1 % – 7 %.

8.3 Known issues and possible caveats

As shown in [1], the performance is very good for temperature calculations. However, a number of issues remain to solved. These are:

- The pressure calculated by FINIX typically differs somewhat from that calculated by FRAPTRAN. The most probable cause for this is a difference in the rod free volume, possibly in the volume of the plenum. The plenum temperature is also calculated in a different way, but the difference exists already for zero power.
- The plastic strains read from a FRAPCON restart file are treated differently by FINIX and FRAPTRAN. It is not exactly clear how FRAPTRAN treats the plastic deformations, but it seems that this is a source of some of the discrepancies between FINIX and FRAPTRAN.
- The steady state solver of FINIX can be very slow to converge. A rewrite of the solver is needed before serious steady state irradiation simulations are performed. Of course, this has to be complemented with the addition of the relevant physical models.

In addition to the technical issues, one should keep in mind the limitations of the FINIX models:

- In many cases, when the range of validity of a model is exceeded, FINIX will not crash or abort execution. Instead, the solver will do its task and pass an error message. It is the responsibility of the user to catch the message and act accordingly. **Calls to FINIX functions should always be accompanied by error message checking.**
- The steady state solver of FINIX-0.15.12 should only be used to solve the initial steady state for a transient calculation. Because key physical models are missing, it should not be used to evaluate the effects of long-term irradiation.
- The coolant model of FINIX is very limited. The model is not reliable beyond nucleate boiling. A warning message is issued if the critical heat flux is exceeded. Also, in FINIX-0.15.12 the temperature of the coolant is not affected by the outward heat flux from the rod. This will affect temperatures at the upper part of the rod, if no external model for the coolant temperature is used.
- FINIX-0.15.12 has no criteria for rod failure.
- FINIX-0.15.12 does not take into account the effect of strain rate on the yield stress in plastic deformation calculations. The strain rate is reported to raise the yield stress a maximum of 10 %.
- FINIX-0.15.12 plastic deformation model is limited to infinitesimal deformations. Scenarios such as cladding ballooning where finite deformation takes place are therefore not

realistically modeled.

- FINIX-0.15.12 has no models for fission gas release. In transients, this may lead to underestimation of the pressure.
- The burnup calculation of FINIX does not differentiate between the thermal and fission power. The results are indicative, and generally accurate within roughly 5–10 % of the experimentally determined values.

9 Summary

The FINIX fuel behavior code has been updated to version 0.15.12. In this version a time-independent plastic deformation model has been implemented in FINIX.

Validation of the stand-alone FINIX-0.13.9 has been done in a separate report [1]. Results show good performance in RIA and steady state scenarios. Especially the temperature distributions are reliably calculated. Limitations have been discussed in Section 8. The validation of version 0.15.12 is to be done in the future.

The primary purpose of the FINIX code is to provide a fuel behavior module for other simulation codes in multiphysics simulations. The intended use is the improvement of fuel behavior description in neutronics, thermal hydraulics and reactor dynamics codes, without having to employ the available full-scale fuel performance codes. FINIX couples with the host code on a source code level, and provides an interface of functions that can be used to access the fuel behavior model from the host code. The required knowledge on the correlation-level details and rod parameters has been minimized by defining default templates that can be used without having information on all model-specific details.

Currently FINIX has been integrated into the Monte Carlo reactor physics code Serpent 2, where FINIX serves as the default fuel behavior module. In addition to Serpent, FINIX has been integrated into VTT's reactor dynamics codes TRAB-1D, TRAB3D and HEXTRAN. Results have been reported, for example, in Refs. [8–10].

References

[1] H. Loukusa. Validation of the FINIX fuel behavior code version 0.13.9. Technical Report VTT-R-06565-13, VTT Technical Research Centre of Finland, 2013.

[2] T. Ikonen. Finix fuel behavior model and interface for multiphysics applications. Code documentation for version 0.13.1. Technical Report VTT-R-00730-13, VTT Technical Research Centre of Finland, 2013.

[3] T. Ikonen. Finix fuel behavior model and interface for multiphysics applications. Code documentation for version 0.13.9. Technical Report VTT-R-06563-13, VTT Technical Research Centre of Finland, 2013.

[4] T. Ikonen, J. Kättö, and H. Loukusa. Finix fuel behavior model and interface for multiphysics applications. Code documentation for version 0.15.6. Technical Report VTT-R-02988-15, VTT Technical Research Centre of Finland, 2015.

[5] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C, 2nd Edition*. Cambridge University Press, 2002.

[6] E. Syrjälahti, V. Valtavirta, J. Kättö, H. Loukusa, T. Ikonen, J. Leppänen, and V. Tulkki. Multiphysics simulations of fast transients in VVER-1000 and VVER-440 reactors. In *11th International Conference on WWER Fuel Performance, Modelling and Experimental Support, Varna, Bulgaria*, 2015.

[7] T. Ikonen, E. Syrjälahti, V. Valtavirta, H. Loukusa, J. Leppänen, and V. Tulkki. Multiphysics simulation of fast transients with the FINIX fuel behaviour module. In *TopFuel 2015, Zurich, Switzerland*, 2015.

[8] T. Ikonen, H. Loukusa, E. Syrjälahti, V. Valtavirta, J. Leppänen, and V. Tulkki. Module for thermomechanical modeling of lwr fuel in multiphysics simulations. *Annals of Nuclear Energy*, 84:111–121, 2015.

[9] V. Valtavirta, T. Ikonen, T. Viitanen, and J. Leppänen. Simulating fast transients with fuel behavior feedback using the serpent 2 monte carlo code. In *PHYSOR2014, Kyoto, Japan*, 2014.

[10] T. Ikonen, V. Tulkki, E. Syrjälahti, V. Valtavirta, and J. Leppänen. FINIX – fuel behavior model and interface for multiphysics applications. In *2013 Fuel Performance Meeting / TopFuel, Charlotte, USA*, 2013.

[11] K.J. Geelhood, W.G. Luscher, C.E. Beyer, and J.M. Cuta. Fraptran 1.4: A computer code for the transient analysis of oxide fuel rods. Technical Report NUREG-CR-7023, Vol. 1, Pacific Northwest National Laboratory, 2011.

[12] K.J. Geelhood, W.G. Luscher, and C.E. Beyer. Frapcon-3.4: A computer code for the calculation of steady-state thermal-mechanical behavior of oxide fuel rods for high burnup. Technical Report NUREG-CR-7022, Vol. 1, Pacific Northwest National Laboratory, 2011.

[13] W.G. Luscher and K.J. Geelhood. Material property correlations: Comparisons between

FRAPCON-3.4, FRAPTRAN 1.4, and MATPRO. Technical Report NUREG-CR-7024, Pacific Northwest National Laboratory, 2011.

[14] M. Suzuki and H. Saitou. *Light Water Reactor Fuel Analysis Code FEMAXI-6 (Ver.1) - Detailed Structure and User's Manual*. Japan Atomic Energy Agency, 2006.

[15] A. S. Khan and S. Huang. *Continuum theory of plasticity*. John Wiley & Sons, Inc., 1995.

[16] F. Dunne and N. Petrinic. *Introduction to computational plasticity*. Oxford University Press, 2005.

[17] A. Shestopalov, K. Lioutov, and Yegorova. Adaptation of USNRC's FRAPTRAN and IRSN's SCANAIRtransient codes and updating of MATPRO package for modeling of LOCA and RIA validation cases with Zr-1%Nb (VVER type) cladding. Technical Report NUREG/IA-0209, US NRC, 2003.

[18] V. Valtavirta. Designing and implementing a temperature solver routine for Serpent. Master's thesis, Aalto University, 2012.

[19] K. J. Geelhood, C. E. Beyer, and W. G. Luscher. PNNL stress/strain correlation for Zircaloy. Technical Report PNNL-17700, Pacific Northwest National Laboratory, 2008.

[20] H. Petersen. The properties of helium: density, specific heats, viscosity, and thermal conductivity at pressures from 1 to 100 bar and from room temperature to about 1800 K. Technical Report RISO-224, Danish Atomic energy Commission Research Establishment Risö, 1970.

[21] D.G. Reddy and C.F. Fighetti. Parametric study of CHF data. volume 2. a generalized subchannel CHF correlation for PWR and BWR fuel assemblies. final report. Technical Report EPRI-NP-2069-Vol. 2, Columbia Univ., New York (USA). Dept. of Chemical Engineering, 1983.

[22] R.W. Lewis, P. Nithiarasu, and K.N. Seetharamu. *Fundamentals of the Finite Element Method for Heat and Fluid Flow*. John Wiley & Sons, 2004.

[23] J.C. Tannehill, D.A. Anderson, and R.H. Pletcher. *Computational Fluid Mechanics and Heat Transfer*. Taylor and Francis, 1997.

[24] K.J. Geelhood, W.G. Luscher, and C.E. Beyer. Fraptran 1.4: Integral assessment. Technical Report NUREG-CR-7023, Vol. 2, Pacific Northwest National Laboratory, 2011.

A FINIX code documentation

The implementation-level documentation for the FINIX code is included at the end of this document.

FINIX-0.15.12

Generated by Doxygen 1.8.7

Mon Dec 7 2015 14:59:52

Contents

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

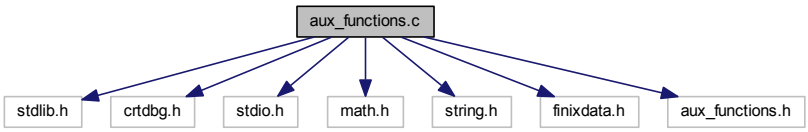
Chapter 2

File Documentation

2.1 aux_functions.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "finixdata.h"
#include "aux_functions.h"
```

Include dependency graph for aux_functions.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- int finix_append_err (char ***old_err, char ***new_err)
- char ** finix_create_err (const char *msg)
- int finix_printf_err (char **err)
- int finix_free_err (char ***err)
- int finix_printf_array (Options *options, double **array)
- int finix_fprintf_array (FILE *stream, Options *options, double time, double **array)
- double finix_interpolate_lookup (double **array, double x, int len, int q)
- double finix_calculate_volume_average (double **r, double **A, int zind, int rind_min, int rind_max)
- double finix_calculate_linear_power (int zind, Options *options, Results *results, double **power_den)
- double finix_calculate_average_power (int mode, Options *options, Boundary_conditions *bc, Results *results, Rod *rod)

2.1.1 Macro Definition Documentation

2.1.1.1 #define _CRTDBG_MAP_ALLOC

2.1.2 Function Documentation

2.1.2.1 int finix_append_err (char *** *old_err,* char *** *new_err*)

Appends a new error to previous FINIX error.

Parameters

old_err	array of previous error messages. will contain the messages in new_err after the function call.
new_err	array of new error messages to be appended to old_err. will be free'd upon function call.

Returns

0 if new message appended, 1 if there was nothing to append

2.1.2.2 double finix_calculate_average_power (int *mode,* Options * *options,* Boundary_conditions * *bc,* Results * *results,* Rod * *rod*)

Calculate rod average power

Parameters

mode	calculation mode
rod	data structure containing rod related data
bc	data structure containing rod boundary conditions
results	data structure containing FINIX simulation results
options	data structure containing the simulation options

Returns

average power

2.1.2.3 double finix_calculate_linear_power (int *zind,* Options * *options,* Results * *results,* double ** *power_den*)

Calculation of linear power from power density. Added 31 May 2013 by Timo Ikonen.

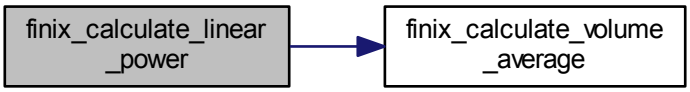
Parameters

zind	axial node index
options	FINIX calculation options
results	data structure containing FINIX simulation results
power_den	power density at each node (W/m^3)

Returns

linear power at axial node zind

Here is the call graph for this function:



2.1.2.4 double finix_calculate_volume_average (double ** *r*, double ** *A*, int *zind*, int *rind_min*, int *rind_max*)

Calculation of volume weighted average over the given radial nodes for the specified axial node. Assumes linear behavior between nodes. Added 30 May 2013 by Timo Ikonen.

Parameters

r	positions of the nodes
A	array holding the quantity to be averaged
zind	index of the axial node
rind_min	minimum index of the radial nodes to be included in the average
rind_max	maximum index of the radial nodes to be included in the average

Returns

the volume weighted average of the quantity A

2.1.2.5 char** finix_create_err (const char * *msg*)

Creates a new FINIX error message

Parameters

msg	new error message

Returns

error message in FINIX error format

2.1.2.6 int finix_fprintf_array (FILE * *stream*, Options * *options*, double *time*, double ** *array*)

Prints out the values of the given array at each node to target stream (works similarly to fprintf in C).

Parameters

stream	target stream
options	FINIX calculation options
time	simulation time (or other identifier of type double that distinguishes between function calls)
array	the array to be printed out

Returns

0

2.1.2.7 int finix_free_err (char *** *err*)

Free's the memory allocated to FINIX error message. Leaves the pointer in state NULL.

Parameters

err	FINIX error to be cleared (free'd)

Returns

0 if error cleared, 1 is error was already cleared (NULL)

2.1.2.8 double finix_interpolate_lookup (double ** *array,* double *x,* int *len,* int *q*)

Linear interpolation of a sorted lookup table with the binary search method.

Parameters

array	lookup table, with array[0] containing the index (or 'x' coordinate) and array[1] the value (or 'y' coordinate). the array should be sorted by array[0] in ascending order
x	the index or 'x' coordinate to be found in array[0]
len	length of the array
q	initial guess for the index

Returns

the interpolated value corresponding to x

2.1.2.9 int finix_printf_array (Options * *options,* double ** *array*)

Prints out the values of the given array at each node to stdout

Parameters

options	FINIX calculation options
array	the array to be printed out

Returns

0

2.1.2.10 int finix_printf_err (char ** *err*)

Prints the FINIX error message to stdout.

Parameters

err	FINIX error to be printed

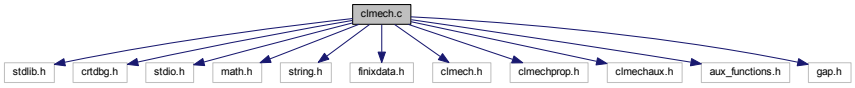
Returns

0 if message printed, 1 is message was empty (NULL)

2.2 clmech.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "finixdata.h"
#include "clmech.h"
#include "clmechprop.h"
#include "clmechaux.h"
#include "aux_functions.h"
#include "gap.h"
```

Include dependency graph for clmech.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- char ** finix_mech_solve_cladding_stress_strain (int zind, double **power_den, Rod *rod, Boundary_↩ conditions *bc, Results *results, Options *options)
- char ** finix_mech_radial_return (double E, double xnu, double G, double Tclad, Cylindrical *stress_eng, Cylindrical strain, Cylindrical *dplstrain_eng, double *stress_eff_eng, double strain_eff, double thstrain_eff, double plstrain_eff, double *dplstrain_eff_eng, double *YS_eng, Rod *rod, Boundary_conditions *bc)
- char ** finix_mech_closed_gap_strong_contact (double E, double xnu, double Ri, double *Ro, double Ric, double Roc, Cylindrical *stress, Cylindrical *strain, Cylindrical thstrain, Cylindrical plstrain, Cylindrical dplstrain)
- char ** finix_mech_closed_gap_weak_contact (double E, double xnu, double Ri, double *Ro, double Ric, double Roc, double p_contact, double p_coolant, Cylindrical *stress, Cylindrical *strain, Cylindrical thstrain, Cylindrical plstrain, Cylindrical dplstrain)
- char ** finix_mech_solve_thin_cladding (double **power_den, Rod *rod, Boundary_conditions *bc, Results *results, Options *options)

2.2.1 Macro Definition Documentation

2.2.1.1 #define _CRTDBG_MAP_ALLOC

2.2.2 Function Documentation

2.2.2.1 char** finix_mech_closed_gap_strong_contact (double *E*, double *xnu*, double *Ri*, double * *Ro*, double *Ric*, double *Roc*, Cylindrical * *stress*, Cylindrical * *strain*, Cylindrical *thstrain*, Cylindrical *plstrain*, Cylindrical *dplstrain*)

Calculates the stress and strain components when the current axial strain, cladding inner radius and the hoop, axial, and radial thermal and plastic strains are known, otherwise known as the closed gap, strong contact case.

Parameters

E	Young's modulus
xnu	Poisson's ratio
Ri	cladding inner radius (fixed by contact)
Ro	cladding outer radius (calculated)
Ric	cladding cold state inner radius
Roc	cladding cold state outer radius
stress	stresses in each direction (hoop and axial calculated)
strain	strains in each direction (hoop and radial calculated, axial fixed by contact)
thstrain	thermal strains in each direction
plstrain	plastic strains in each direction
dplstrain	plastic strain increments in each direction

Returns

error string, NULL for no errors

2.2.2.2 char** finix_mech_closed_gap_weak_contact (double *E*, double *xnu*, double *Ri*, double * *Ro*, double *Ric*, double *Roc*, double *p_contact*, double *p_coolant*, Cylindrical * *stress*, Cylindrical * *strain*, Cylindrical *thstrain*, Cylindrical *plstrain*, Cylindrical *dplstrain*)

Iterates the cladding outer radius with Newton-Raphson iteration and calculates stresses and strains in the closed gap, weak contact case.

Parameters

E	Young's modulus
xnu	Poisson's ratio
Ri	cladding inner radius (fixed by contact)
Ro	cladding outer radius (calculated)
Ric	cladding cold state inner radius
Roc	cladding cold state outer radius
p_contact	the current contact pressure
p_coolant	coolant pressure
stress	stresses in each direction (calculated)
strain	strains in each direction (hoop and radial calculated, axial fixed by contact)
thstrain	thermal strains in each direction
plstrain	plastic strains in each direction
dplstrain	plastic strain increments in each direction

Returns

error string, NULL for no errors

2.2.2.3 char** finix_mech_radial_return (double *E,* double *xnu,* double *G,* double *Tclad,* Cylindrical * *stress_eng,* Cylindrical *strain,* Cylindrical * *dplstrain_eng,* double * *stress_eff_eng,* double *strain_eff,* double *thstrain_eff,* double *plstrain_eff,* double * *dplstrain_eff_eng,* double * *YS_eng,* Rod * *rod,* Boundary_conditions * *bc*)

Calculates the plastic strain increment with the radial return algorithm. Input stresses and strains that are output are input as pointers (marked with (*)), values that are not output are input as values. Input variables that are not output are converted into true stresses and strains in the function, input variables that are output have corresponding true stress/strain variables in the function.

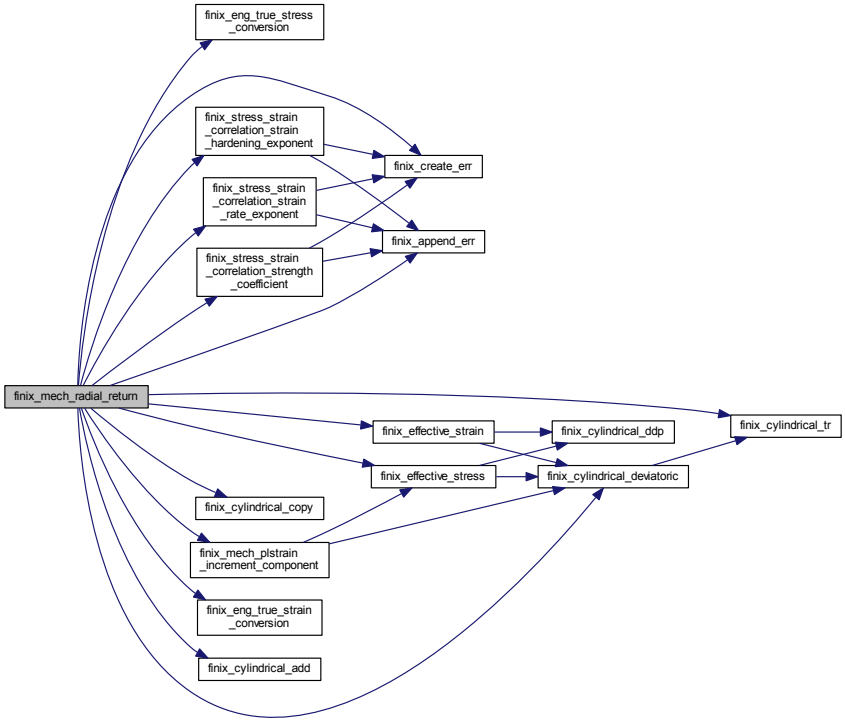
Parameters

E	Young's modulus (MPa)
xnu	Poisson's ratio (dimensionless)
G	shear modulus (MPa)
Tclad	cladding average temperature (K)
stress_eng	pointer to stress tensor (*)
strain	components of strain
dplstrain_eng	pointer to plastic strain increment tensor (*)
stress_eff_eng	pointer to effective stress (*)
strain_eff	effective strain
thstrain_eff	effective thermal strain
plstrain_eff	effective plastic strain
dplstrain_eff_eng	pointer to effective plastic strain increment (*)
YS_eng	pointer to yield stress (*)
rod	the Rod structure
bc	the Boundary_conditions structure

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.2.2.4 char** finix_mech_solve_cladding_stress_strain (int *zind,* double ** *power_den,* Rod * *rod,* Boundary_conditions * *bc,* Results * *results,* Options * *options*)

Solves the stresses, strains and contact pressures of the cladding for one axial node using the thin cladding approximation. Updated 7 June 2013 to include plastic strains as constant input (Timo Ikonen).

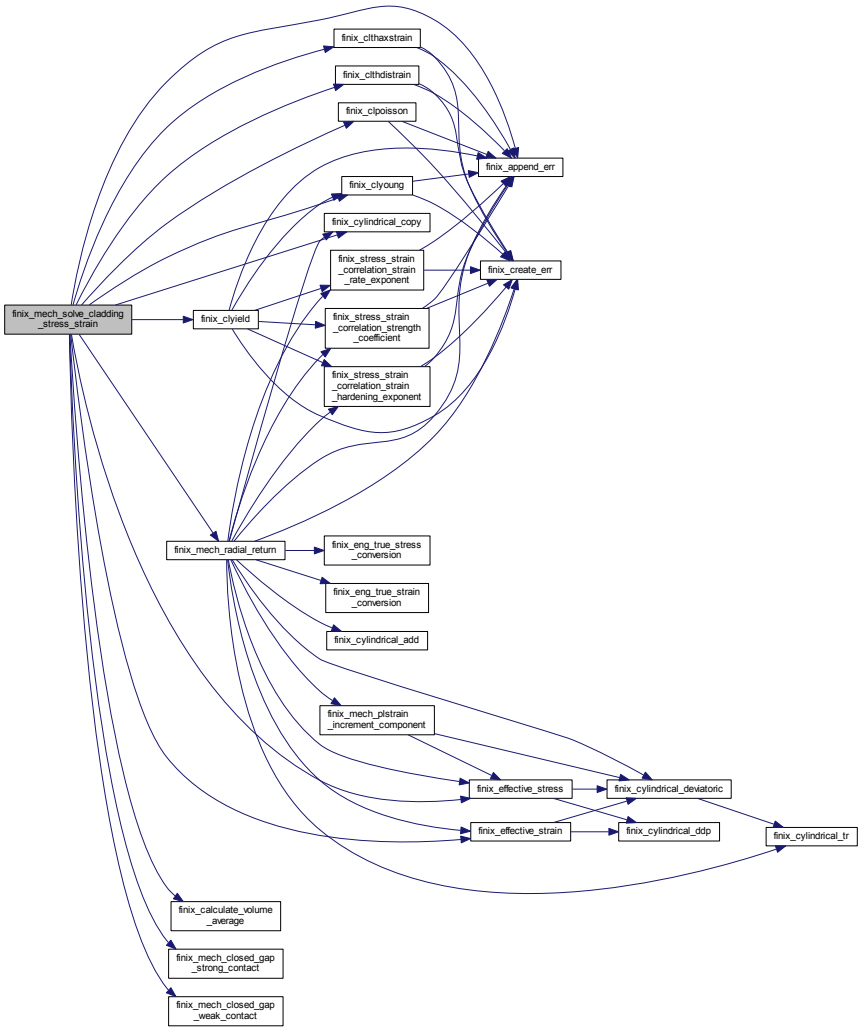
Parameters

zind	axial node index
power_den	power density (W/m^3)
rod	data structure containing fuel rod properties
bc	data structure containing rod boundary conditions
results	data structure containing FINIX simulation results
options	FINIX calculation options

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.2.2.5 char** finix_mech_solve_thin_cladding (double ** *power_den,* Rod * *rod,* Boundary_conditions * *bc,* Results * *results,* Options * *options*)

Solves the mechanical behaviour of the cladding with the thin cladding appoximation.

Parameters

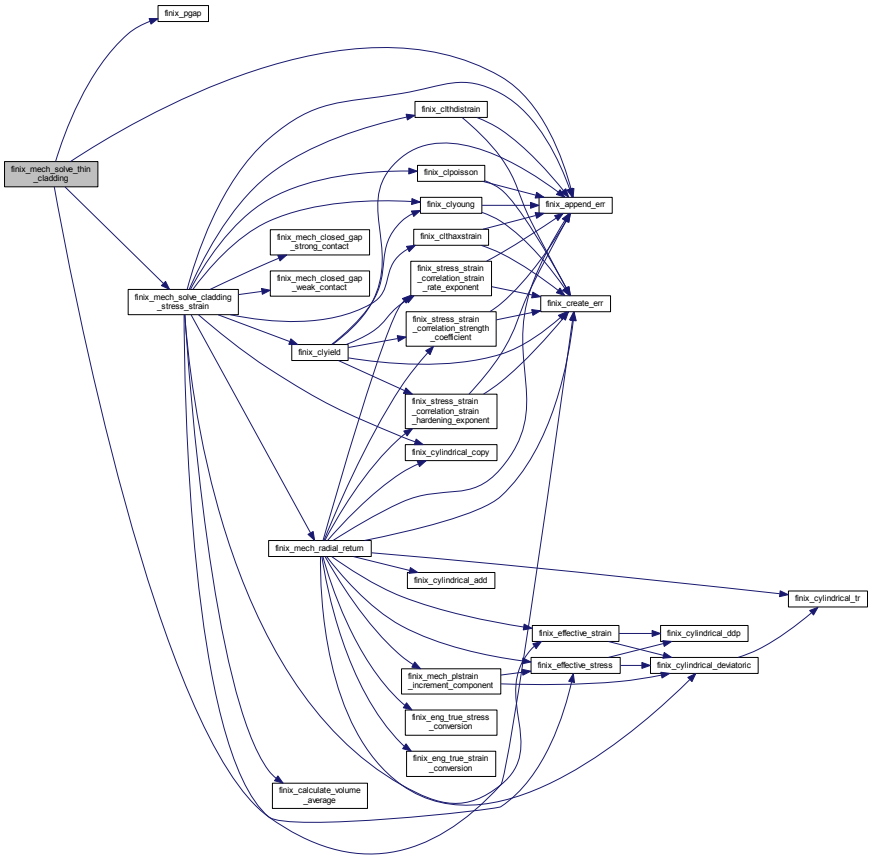
power_den	power density (W/m^3)
rod	data structure containing fuel rod properties
bc	data structure containing rod boundary conditions
results	data structure containing FINIX simulation results

options	FINIX calculation options

Returns

error string, NULL for no errors

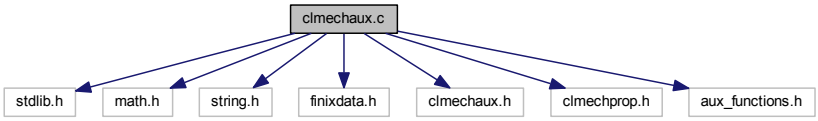
Here is the call graph for this function:



2.3 clmechaux.c File Reference

```
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "finixdata.h"
#include "clmechaux.h"
#include "clmechprop.h"
#include "aux_functions.h"
```

Include dependency graph for clmechaux.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- void finix_mech_plstrain_increment_component (Cylindrical stress, double dplstrain_eff, Cylindrical *dplstrain)
- double finix_effective_stress (Cylindrical stress)
- double finix_effective_strain (Cylindrical strain)
- void finix_eng_true_stress_conversion (Cylindrical inputstress, Cylindrical inputstrain, Cylindrical *outputstress, char output)
- void finix_eng_true_strain_conversion (Cylindrical inputstrain, Cylindrical *outputstrain, char output)

2.3.1 Macro Definition Documentation

2.3.1.1 #define _CRTDBG_MAP_ALLOC

2.3.2 Function Documentation

2.3.2.1 double finix_effective_strain (Cylindrical *strain*)

Calculates the (von Mises) effective strain from strain tensor expressed as a Cylindrical struct.

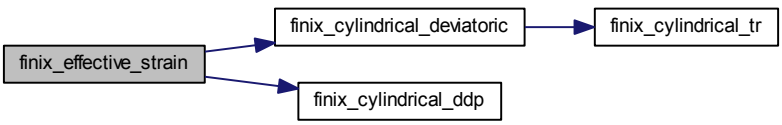
Parameters

strain	the strain tensor in Cylindrical form

Returns

the effective strain

Here is the call graph for this function:



2.3.2.2 double finix_effective_stress (Cylindrical *stress*)

Calculates the (von Mises) effective stress from a stress tensor expressed as a Cylindrical struct.

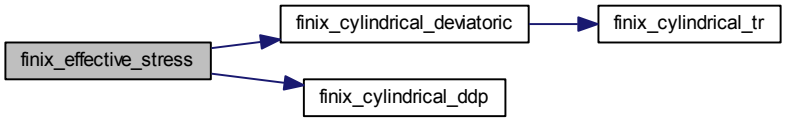
Parameters

stress	the stress tensor in Cylindrical form

Returns

the effective stress

Here is the call graph for this function:



2.3.2.3 void finix_eng_true_strain_conversion (Cylindrical *inputstrain*, Cylindrical * *outputstrain*, char *output*)

Converts engineering strain to true strain or true strain to engineering strain. Input values as Cylindrical structures.

Parameters

inputstrain	the engineering or true strain tensor as a Cylindrical struct
outputstrain	pointer to the true or engineering strain tensor as a Cylindrical struct
output	option to convert from true to eng (set to 'E') or from eng to true (set to 'T')

2.3.2.4 void finix_eng_true_stress_conversion (Cylindrical *inputstress*, Cylindrical *inputstrain*, Cylindrical * *outputstress*, char *output*)

Converts engineering stress to true stress or true stress to engineering stress using the current engineering strains. Input values as Cylindrical structures.

Parameters

inputstress	the engineering or true stress (MPa)
inputstrain	the engineering strain (NOTE: always engineering strain)
outputstress	pointer to the true or engineering stress tensor as a Cylindrical struct (MPa)
output	option to convert from true to eng (set to 'E') or from eng to true (set to 'T')

2.3.2.5 void finix_mech_plstrain_increment_component (Cylindrical *stress*, double *dplstrain_eff*, Cylindrical * *dplstrain*)

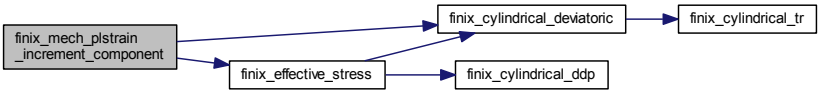
Components of the plastic strain increment

Calculates the plastic strain increment by the Prandtl-Reuss flow rule from the total plastic strain increment for each strain component.

Parameters

stress	stress tensor in Cylindrical form (Pa)
dplstrain_eff	the effective plastic strain increment (unitless)
dplstrain	pointer to the plastic strain increment tensor in Cylindrical form (unitless)

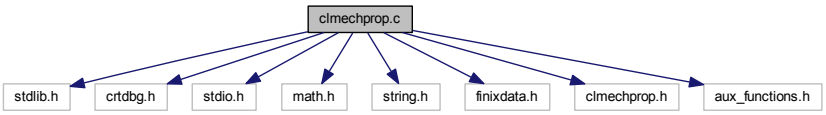
Here is the call graph for this function:



2.4 clmechprop.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "finixdata.h"
#include "clmechprop.h"
#include "aux_functions.h"
```

Include dependency graph for clmechprop.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- char ** finix_clmeyer (double T, double *HM, int cladtype)
- char ** finix_clpoisson (double T, double *nu, int cladtype)
- char ** finix_clyoung (double T, double oxygen_con, double coldwork, double fnf, double *E, int cladtype)
- char ** finix_clthaxstrain (double T, double *strain, int cladtype)
- char ** finix_clthdistrain (double T, double *strain, int cladtype)
- char ** finix_stress_strain_correlation_strength_coefficient (double T, double coldwork, double fnf, double *K, int cladtype)
- char ** finix_stress_strain_correlation_strain_hardening_exponent (double T, double fnf, double *n, int cladtype)
- char ** finix_stress_strain_correlation_strain_rate_exponent (double T, double *m, int cladtype)
- char ** finix_clyield (double T, double effective_stress, double effective_strain, double plstrain_eff, double strain_rate, double *yield_stress, Rod *rod, Boundary_conditions *bc)

2.4.1 Macro Definition Documentation

2.4.1.1 #define _CRTDBG_MAP_ALLOC

2.4.2 Function Documentation

2.4.2.1 char** finix_clmeyer (double *T*, double * *HM*, int *cladtype*)

Cladding Meyer's hardness

Calculates the Meyer's hardness of the cladding. FRAPTRAN-1.4 correlations for Zircaloy and Zr1Nb (E110) are implemented. See Shestopalov, A. et al. (2003), NUREG/IA-0209, for E110 correlation.

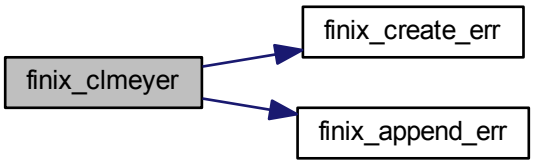
Parameters

T	temperature (K)
HM	Meyer's hardness (N/m^2)
cladtype	cladding type, 0 = Zircaloy, 1 = Zr-1Nb

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.4.2.2 char** finix_clpoisson (double *T*, double * *nu*, int *cladtype*)

Cladding Poisson's ratio

Calculates the Poisson's ratio of the cladding. FRAPTRAN-1.4 correlation for Zircaloy is implemented, and this correlation is also used for Zr1Nb (E110).

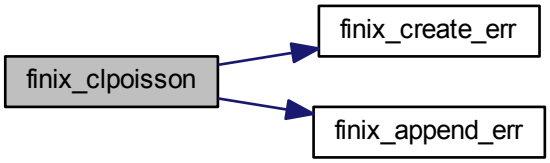
Parameters

T	temperature (K)
nu	Poisson's ratio (dimensionless)
cladtype	cladding type, 0 = Zircaloy, 1 = Zr-1Nb

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.4.2.3 char** finix_clthaxstrain (double *T*, double * *strain*, int *cladtype*)

Cladding thermal axial strain

Calculates the axial thermal strain of the cladding. FRAPTRAN-1.4 (MATPRO) correlations for Zircaloy and Zr1Nb (E110) are implemented. See Shestopalov, A. et al. (2003), NUREG/IA-0209, for E110 correlation.

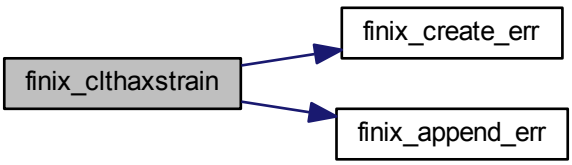
Parameters

T	temperature (K)
strain	thermal axial strain
cladtype	the cladding type (0 = Zircaloy, 1 = E110)

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.4.2.4 char** finix_clthdistrain (double *T*, double * *strain*, int *cladtype*)

Cladding thermal diametral strain

Calculates the diametral thermal strain of the cladding. FRAPTRAN-1.4 (MATPRO) correlations for Zircaloy and Zr1Nb (E110) are implemented. See Shestopalov, A. et al. (2003), NUREG/IA-0209, for E110 correlation.

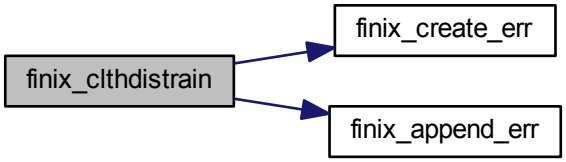
Parameters

T	temperature (K)
strain	thermal axial strain
cladtype	the cladding material (0 = Zircaloy, 1 = E110)

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.4.2.5 char** finix_clyield (double *T*, double *effective_stress*, double *effective_strain*, double *plstrain_eff*, double *strain_rate*, double * *yield_stress*, Rod * *rod*, Boundary_conditions * *bc*)

Yield stress

Calculates the yield stress of the cladding. Inputs are given in engineering stress/strain, and converted to true stress/strain for the correlation. Yield stress is output as engineering stress.

Zircaloy: The PNNL correlation is used. See Geelhood, Luscher & Beyer (2007), PNNL-17700. Zr1Nb (E110): The RRC-KI correlation is used. See Shestopalov, A. et al. (2003), NUREG/IA-0209.

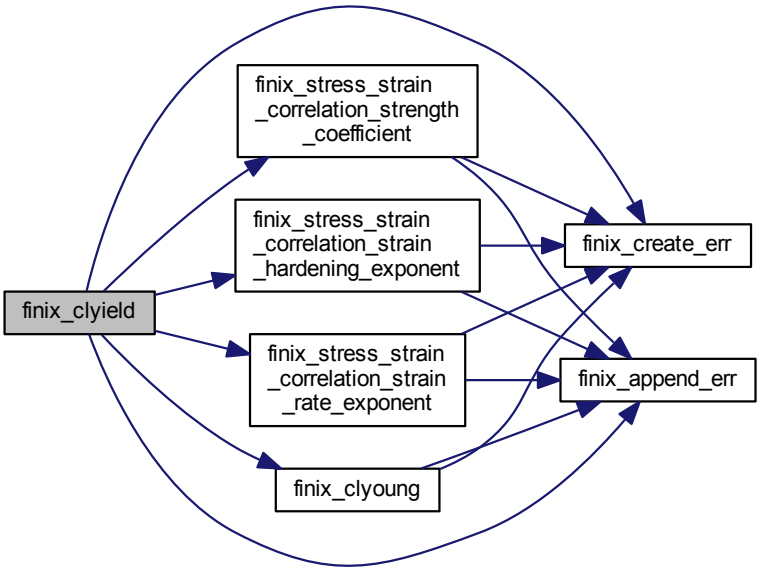
Parameters

T	temperature of the cladding (K)
effective_stress	the effective engineering stress at this time step (Pa)
effective_strain	the effective engineering strain at this time step (unitless)
plstrain_eff	the total effective plastic engineering strain (set to zero for yield stress without prior deformation)
strain_rate	the current engineering strain rate (s^-1)
stress	the calculated engineering yield stress (Pa)
rod	the Rod structure
bc	the Boundary_conditions structure

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.4.2.6 char** finix_clyoung (double *T*, double *oxygen_con*, double *coldwork*, double *fnf*, double * *E*, int *cladtype*)

Cladding Young's modulus

Calculates the Young's modulus of the cladding. FRAPTRAN-1.4 correlations for Zircaloy and Zr1Nb (E110) are implemented.

Parameters

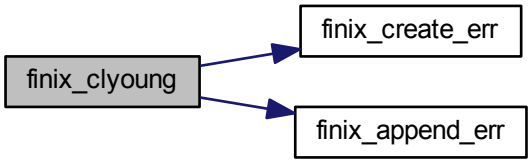
T	temperature (K)
oxygen_con	average oxygen concentrantion minus oxygen concentration of as-received cladding (kg oxygen / kg Zircaloy)
coldwork	cladding cold work (unitless ratio of areas)
fnf	fast neutron fluence (n/m^2)
E	Young's modulus (N/m^2)
cladtype	the cladding type (0 = Zircaloy, 1 = E110)

Returns

error string, NULL for no errors

assume as-received oxygen concentration of 0.0012.

Here is the call graph for this function:



2.4.2.7 char** finix_stress_strain_correlation_strain_hardening_exponent (double *T,* double *fnf,* double * *n,* int *cladtype*)

Stress-strain curve plastic region power law strain hardening exponent

Calculates the strain hardening exponent of the power law used to calculate the stress-strain curve of the cladding. Zircaloy: The PNNL correlation is used. See Geelhood, Luscher & Beyer (2007), PNNL-17700. Zr1Nb (E110): The RRC-KI correlation is used. See Shestopalov, A. et al. (2003), NUREG/IA-0209.

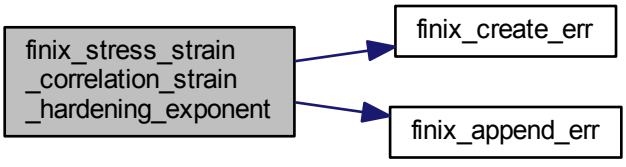
Parameters

T	temperature (K)
fnf	the fast neutron fluence (n/m^2)'
n	the calculated strain hardening exponent
cladtype	cladding type, 0 = Zircaloy, 1 = Zr-1Nb

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.4.2.8 char** finix_stress_strain_correlation_strain_rate_exponent (double *T,* double * *m,* int *cladtype*)

Stress-strain curve plastic region power law strain rate exponent

Calculates the strain rate exponent of the power law used to calculate the stress-strain curve of the cladding. Zircaloy: The PNNL correlation is used. See Geelhood, Luscher & Beyer (2007), PNNL-17700. Zr1Nb (E110): The RRC-KI correlation is used. See Shestopalov, A. et al. (2003), NUREG/IA-0209.

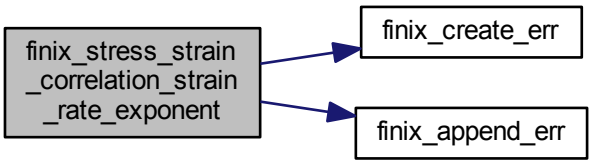
Parameters

T	temperature (K)
m	the calculated strain rate exponent
cladtype	cladding type, 0 = Zircaloy, 1 = Zr-1Nb

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.4.2.9 char** finix_stress_strain_correlation_strength_coefficient (double *T,* double *coldwork,* double *fnf,* double * *K,* int *cladtype*)

Stress-strain curve plastic region power law strength coefficient

Calculates the strength coefficient of the power law used to calculate the stress-strain curve of the cladding. Zircaloy: The PNNL correlation is used. See Geelhood, Luscher & Beyer (2007), PNNL-17700. Zr1Nb (E110): The RRC-KI correlation is used. See Shestopalov, A. et al. (2003), NUREG/IA-0209.

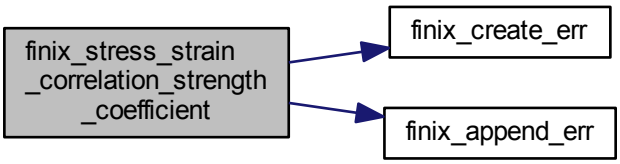
Parameters

T	temperature (K)
coldwork	cold work parameter (unitless)
fnf	the fast neutron fluence (n/m^2)'
K	the calculated strength coefficient
cladtype	cladding type, 0 = Zircaloy, 1 = Zr-1Nb

Returns

error string, NULL for no errors

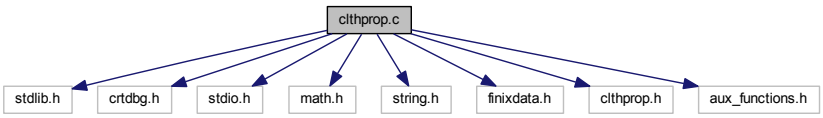
Here is the call graph for this function:



2.5 clthprop.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "finixdata.h"
#include "clthprop.h"
#include "aux_functions.h"
```

Include dependency graph for clthprop.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- char ** finix_clthcond (double T, double *lambda, int cladtype)
- char ** finix_clcp (double T, double *cp, int cladtype)

2.5.1 Macro Definition Documentation

2.5.1.1 #define _CRTDBG_MAP_ALLOC

2.5.2 Function Documentation

2.5.2.1 char** finix_clcp (double *T*, double * *cp*, int *cladtype*)

Cladding specific heat

Calculates the specific heat of the cladding. The FRAPTRAN-1.4 correlation for Zircaloy is implemented (see NUREG/CR-7024) and the RRC-KI correlation for Zr1Nb (E110) is implemented (see Shestopalov et al. (2003) NUREG/IA-0209).

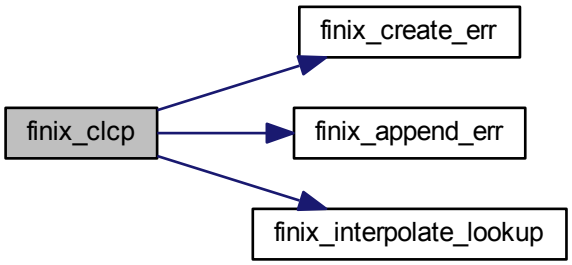
Parameters

T	temperature (K)
cp	specific heat
cladtype	the cladding type (0 = Zircaloy, 1 = Zr1Nb)

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.5.2.2 char** finix_clthcond (double *T*, double * *lambda*, int *cladtype*)

Cladding thermal conductivity

Calculates the thermal conductivity of the cladding. The FRAPTRAN-1.4 correlation for Zircaloy is implemented (see NUREG/CR-7024) and the RRC-KI correlation for Zr1Nb (E110) is implemented (see Shestopalov et al. (2003) NUREG/IA-0209).

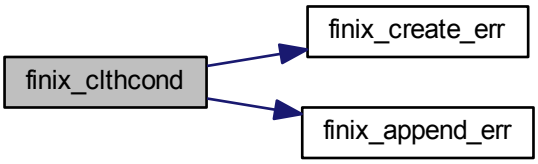
Parameters

T	temperature (K)
lambda	thermal conductivity (W/mK)
cladtype	the cladding type (0 = Zircaloy, 1 = Zr1Nb)

Returns

error string, NULL for no errors

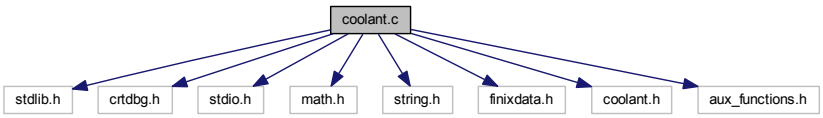
Here is the call graph for this function:



2.6 coolant.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "finixdata.h"
#include "coolant.h"
#include "aux_functions.h"
```

Include dependency graph for coolant.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- char ** finix_hcoolant (int zind, Options *options, Results *results, Boundary_conditions *bc, Rod *rod)
- char ** finix_update_bcond (int zind, Boundary_conditions *bc, Rod *rod, Options *options, Results *results)

2.6.1 Macro Definition Documentation

2.6.1.1 #define _CRTDBG_MAP_ALLOC

2.6.2 Function Documentation

2.6.2.1 char** finix_hcoolant (int *zind*, Options * *options*, Results * *results*, Boundary_conditions * *bc*, Rod * *rod*)

Calculates the heat transfer coefficient between the cladding outer surface and the coolant. Uses Dittus-Boelter for single-phase convection and Thom correlation for nucleate boiling.

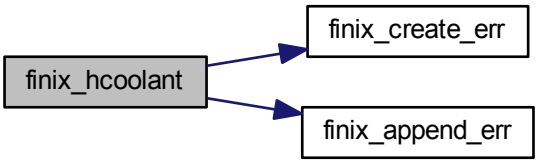
Parameters

zind	index of the axial node
options	FINIX calculation options
results	data structure including array radial_node_position which holds the node positions
bc	boundary conditions, including the temperatures of the cladding and the coolant (must be given), and the heat transfer coefficient (function output)
rod	data structure containing fuel rod properties

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.6.2.2 char** finix_update_bcond (int *zind*, Boundary_conditions * *bc*, Rod * *rod*, Options * *options*, Results * *results*)

Calculates the heat transfer coefficient between the cladding outer surface and the coolant. Uses Dittus-Boelter for single-phase convection and Thom correlation for nucleate boiling.

Parameters

zind	index of the axial node
results	data structure including array radial_node_position which holds the node positions
bc	boundary conditions, including the temperatures of the cladding and the coolant (must be given), and the heat transfer coefficient (function output)
rod	data structure containing fuel rod properties
options	Model options. The boundary conditions are selected in options->boundary_option.

Returns

error string, NULL for no errors

If options->boundary_option==0, use user-given rod outer surface temperature as boundary condition

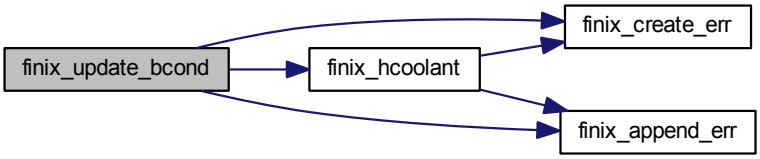
If options->boundary_option==1, use user-given heat flux between rod outer surface and coolant

If options->boundary_option==2, use user-given heat transfer coefficient and coolant bulk temperature as boundary condition

If options->boundary_option==3, use user-given bulk temperature and calculate heat transfer coefficient from internal correlations (needs inlet mass flux)

If options->boundary_option==4, use user-given inlet temperature and mass flux to calculate heat transfer coefficient and coolant bulk temperature downstream (NOT IMPLEMENTED)

Here is the call graph for this function:



2.7 finix_initialization.c File Reference

```
#include "finixdata.h"
#include "finix_initialization.h"
#include "aux_functions.h"
#include "gap.h"
#include "finix_output.h"
#include "fumech.h"
#include "clmech.h"
#include "futhprop.h"
#include "clthprop.h"
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <stddef.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include <sys/stat.h>
```

Include dependency graph for finix_initialization.c:

Macros

- #define M_PI 3.14159265358979323846

Functions

- char ** finix_initialize_data_structures (Rod *rod, Boundary_conditions *bc, Scenario *scenario, Results *results, Options *options)
- char ** finix_read_input (Rod *rod, Scenario *scenario, Options *options, int mode, int *existing_files)

- Input * finix_inputstring_construct (void)
- void finix_inputstring_destruct (Input *input)
- char ** finix_read_input_file (Rod *rod, Scenario *scenario, Options *options, Input *input, int mode, int *existing_files)
- void finix_read_file_names (Scenario *scenario)
- int finix_does_file_exist (const char *filename)
- void finix_insert_line_to_inputstring (Input *input, const char *element)
- void finix_keywords_initialize ()
- void finix_keywords_destruct ()
- char ** finix_find_data_0d (Rod *rod, Scenario *scenario, Options *options, Input *input, const char *substring, int mode)
- char ** finix_find_data_1d (Rod *rod, Scenario *scenario, Options *options, Input *input, const char *substring, int mode)
- char ** finix_find_data_2d (Rod *rod, Scenario *scenario, Options *options, Input *input, const char *substring, int mode)
- char ** finix_find_data_3d (Rod *rod, Scenario *scenario, Options *options, Input *input, const char *substring, int mode)
- double ** finix_matrix_initialize_2d (int dim1, int dim2)
- int finix_find_the_number_of_substrings (Input *input, const char *substring)
- int finix_find_the_number_of_values (char *location)
- char * finix_read_one_value_from_inputstring (double *value, char *location)
- void finix_read_word_from_inputstring (char *word, char *location)
- void finix_change_to_si_units (const Rod *rod, Options *options, double *value, const char *substring)
- void finix_change_to_seconds (Scenario *scenario, double *value, const char *substring)
- void finix_allocate_memory_for_bc_and_results (Options *options, Boundary_conditions *bc, Results *results)
- char ** finix_set_defaults_for_bc_and_results (Boundary_conditions *bc, Results *results, Options *options, Rod *rod)
- char ** finix_get_default_positions (Options *options, Results *results, Rod *rod)
- char ** finix_check_data_validity (Options *options, Rod *rod, Boundary_conditions *bc, Results *results)
- char ** finix_read_frapcon_restart (Rod *rod, Boundary_conditions *bc, Results *results, Options *options, Scenario *scenario)

2.7.1 Macro Definition Documentation

2.7.1.1 #define M_PI 3.14159265358979323846

2.7.2 Function Documentation

2.7.2.1 void finix_allocate_memory_for_bc_and_results (Options * *options,* Boundary_conditions * *bc,* Results * *results*)

Function allocates memory for Boundary_conditions and Results data structures.

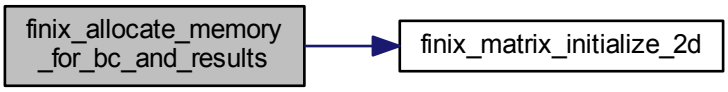
Parameters

bc	data structure containing rod boundary conditions
results	data structure containing FINIX simulation results
options	data structure containing the simulation options

Returns

nothing

Here is the call graph for this function:



2.7.2.2 void finix_change_to_seconds (Scenario * *scenario,* double * *value,* const char * *substring*)

Function converts hours and days to seconds.

Parameters

options	data structure containing the simulation options
value	the original value converted to seconds
substring	a keyword identifying the value to be converted

Returns

nothing

2.7.2.3 void finix_change_to_si_units (const Rod * *rod,* Options * *options,* double * *value,* const char * *substring*)

Function converts Fraptran input data to SI-units.

Parameters

rod	data structure containing rod related data
options	data structure containing the simulation options
value	the original value converted to SI-units
substring	a keyword identifying the value to be converted

Returns

nothing

2.7.2.4 char** finix_check_data_validity (Options * *options,* Rod * *rod,* Boundary_conditions * *bc,* Results * *results*)

Function checks the validity of the data in FINIX data structures.

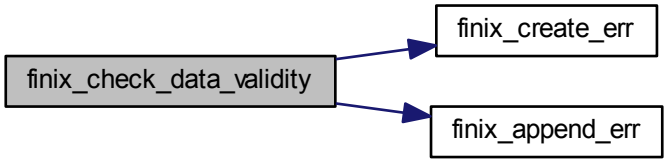
Parameters

rod	data structure containing rod related data
bc	data structure containing rod boundary conditions
results	data structure containing FINIX simulation results
options	data structure containing the simulation options

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.7.2.5 int finix_does_file_exist (const char * *filename*)

Function checks if the file determined in the parameter list exists.

Parameters

filename	the name of the file whose existence will be checked

Returns

nonzero if file exists, zero othewise

2.7.2.6 char** finix_find_data_0d (Rod * *rod*, Scenario * *scenario*, Options * *options*, Input * *input*, const char * *substring*, int *mode*)

Function searches for a keyword from an input string. If the keyword is found, the corresponding value will be stored in FINIX data structures.

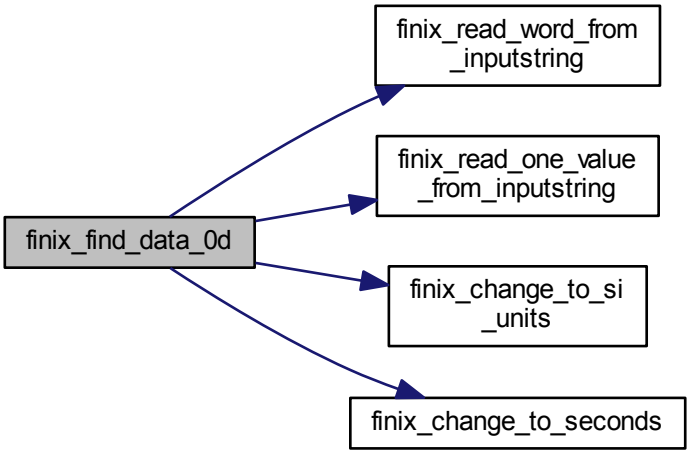
Parameters

rod	data structure containing rod related data
scenario	data structure containing rod history and power data
options	data structure containing the simulation options
input	a data structure containing the contents of an input file in a string format
substring	the keyword to be searched from the input string
mode	an integer defining the input to be read

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.7.2.7 char** finix_find_data_1d (Rod * *rod,* Scenario * *scenario,* Options * *options,* Input * *input,* const char * *substring,* int *mode*)

Function searches for a keyword from an input string. If the keyword is found, the corresponding 1D array of values will be stored in FINIX data structures.

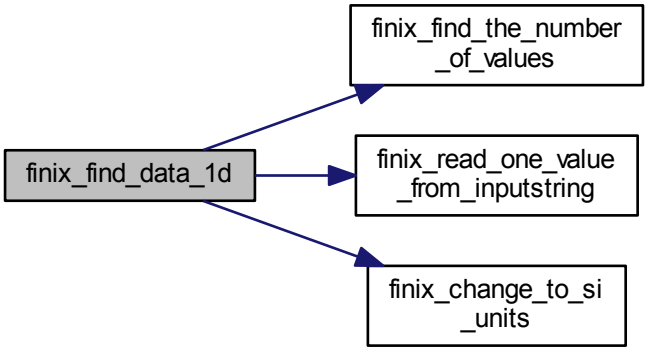
Parameters

rod	data structure containing rod related data
scenario	data structure containing rod history and power data
options	data structure containing the simulation options
input	a data structure containing the contents of an input file in a string format
substring	the keyword to be searched from the input string
mode	an integer defining the input to be read

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.7.2.8 char** finix_find_data_2d (Rod * *rod,* Scenario * *scenario,* Options * *options,* Input * *input,* const char * *substring,* int *mode*)

Function searches for a keyword from an input string. If the keyword is found, the corresponding 2D array of values will be stored in FINIX data structures.

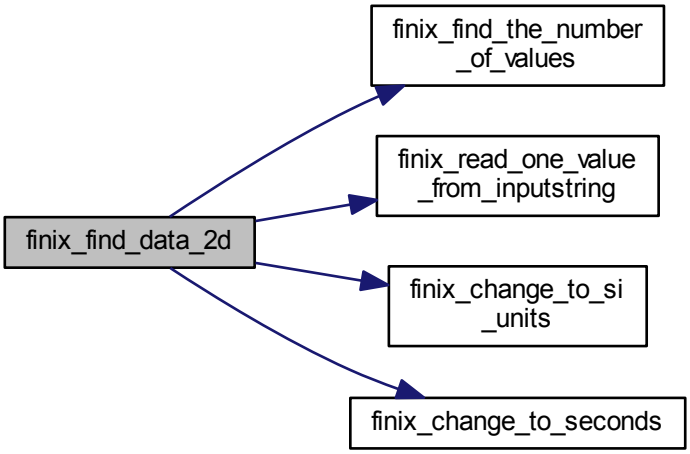
Parameters

rod	data structure containing rod related data
scenario	data structure containing rod history and power data
options	data structure containing the simulation options
input	a data structure containing the contents of an input file in a string format
substring	the keyword to be searched from the input string
mode	an integer defining the input to be read

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.7.2.9 char** finix_find_data_3d (Rod * *rod,* Scenario * *scenario,* Options * *options,* Input * *input,* const char * *substring,* int *mode*)

Function searches for a keyword from an input string. If the keyword is found, the corresponding 3D array of values will be stored in FINIX data structures.

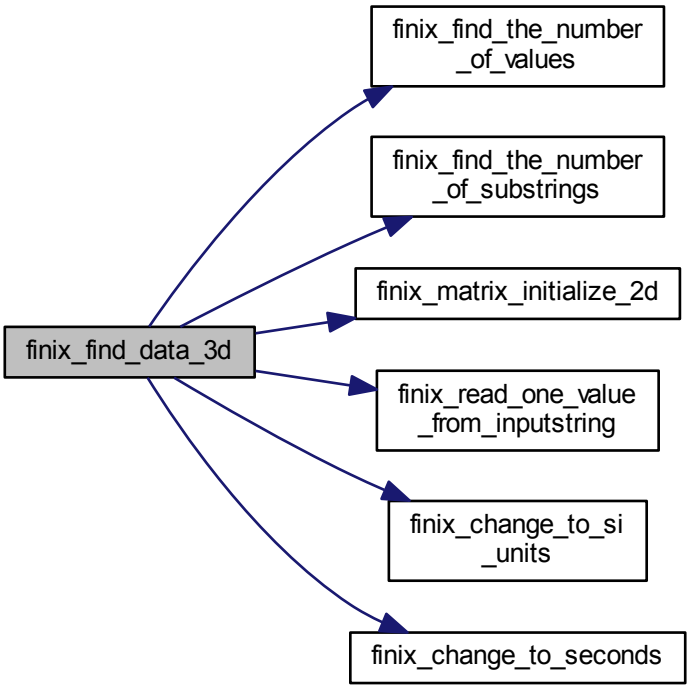
Parameters

rod	data structure containing rod related data
scenario	data structure containing rod history and power data
options	data structure containing the simulation options
input	a data structure containing the contents of an input file in a string format
substring	the keyword to be searched from the input string
mode	an integer defining the input to be read

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.7.2.10 int finix_find_the_number_of_substrings (Input * *input*, const char * *substring*)

Function finds the number of the occurrences of a substring in the input string

Parameters

input	a data structure containing the contents of an input file in a string format
substring	the string to be searched from the input string

Returns

the number of the occurences of a substring in the input string

2.7.2.11 int finix_find_the_number_of_values (char * *location*)

Function goes throug an input-string and calculates how many values are related to a keyword.

Parameters

location	a pointer to the beginning of the reading block

Returns

the number values related to a keyword

2.7.2.12 char** finix_get_default_positions (Options * *options,* Results * *results,* Rod * *rod*)

Function sets default values for fuel rod radial node positions.

Parameters

results	data structure containing FINIX simulation results
options	data structure containing the simulation options
rod	data structure containing rod related data

Returns

error string, NULL for no errors

2.7.2.13 char** finix_initialize_data_structures (Rod * *rod,* Boundary_conditions * *bc,* Scenario * *scenario,* Results * *results,* Options * *options*)

The main subroutine for reading data from input files to FINIX data structures. If input files are not provided, the data structures will be initialized with default values.

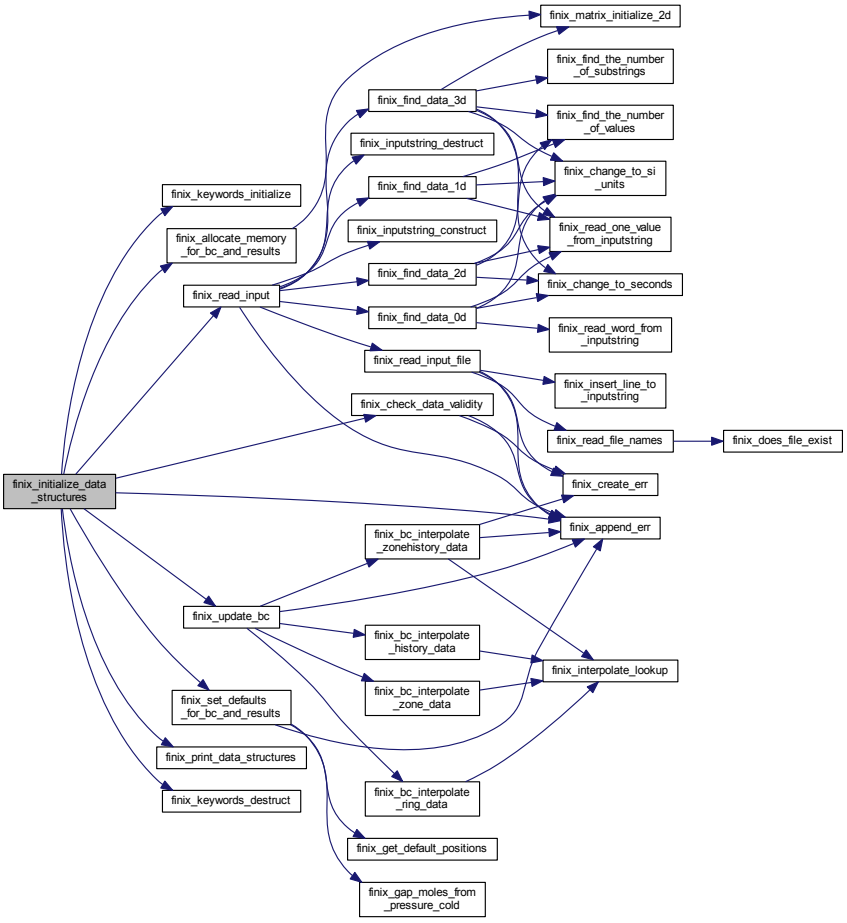
Parameters

rod	data structure containing rod related data
bc	data structure containing rod boundary conditions
scenario	data structure containing rod history and power data
results	data structure containing FINIX simulation results
options	data structure containing the simulation options

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.7.2.14 Input* finix_inputstring_construct (void)

Function creates a new Input data structure. The data structure contains a string where the data from the input file will be stored.

Returns

new_input a data structure that will contain the contents of an input file in a string format

2.7.2.15 void finix_inputstring_destruct (Input * *input*)

Function frees all the memory related to the Input data structure.

Parameters

input	a data structure containing the contents of an input file.

Returns

nothing.

2.7.2.16 void finix_insert_line_to_inputstring (Input * *input,* const char * *element*)

Function appends one line to an input string.

Parameters

input	a data structure containing the contents of and input file
element	a string that will be appended to the input string

Returns

nothing

2.7.2.17 void finix_keywords_destruct ( )

Function frees all the memory allocated to keylist data structure

Returns

nothing

2.7.2.18 void finix_keywords_initialize ( )

Function determines the keywords to be searched from the input files. These keywords are stored in data structure keylist for later use.

Returns

nothing

2.7.2.19 double** finix_matrix_initialize_2d (int *dim1,* int *dim2*)

Function allocates memory for a 2D matrix.

Parameters

dim1	number of columns
dim2	number of rows

Returns

matrix with allocated memory

2.7.2.20 void finix_read_file_names (Scenario * *scenario*)

Function asks the user the name of the Fraptran input file and the name of the Frapcon generated restart file. These file names are stored in scenario data structure.

Parameters

scenario	data structure where the names of the input files will be stored.

Returns

nothing

Here is the call graph for this function:

2.7.2.21 char** finix_read_frapcon_restart (Rod * *rod,* Boundary_conditions * *bc,* Results * *results,* Options * *options,* Scenario * *scenario*)

Reads the FRAPCON-generated FRAPTRAN style restart file for initializing the FINIX model for nonzero burnup

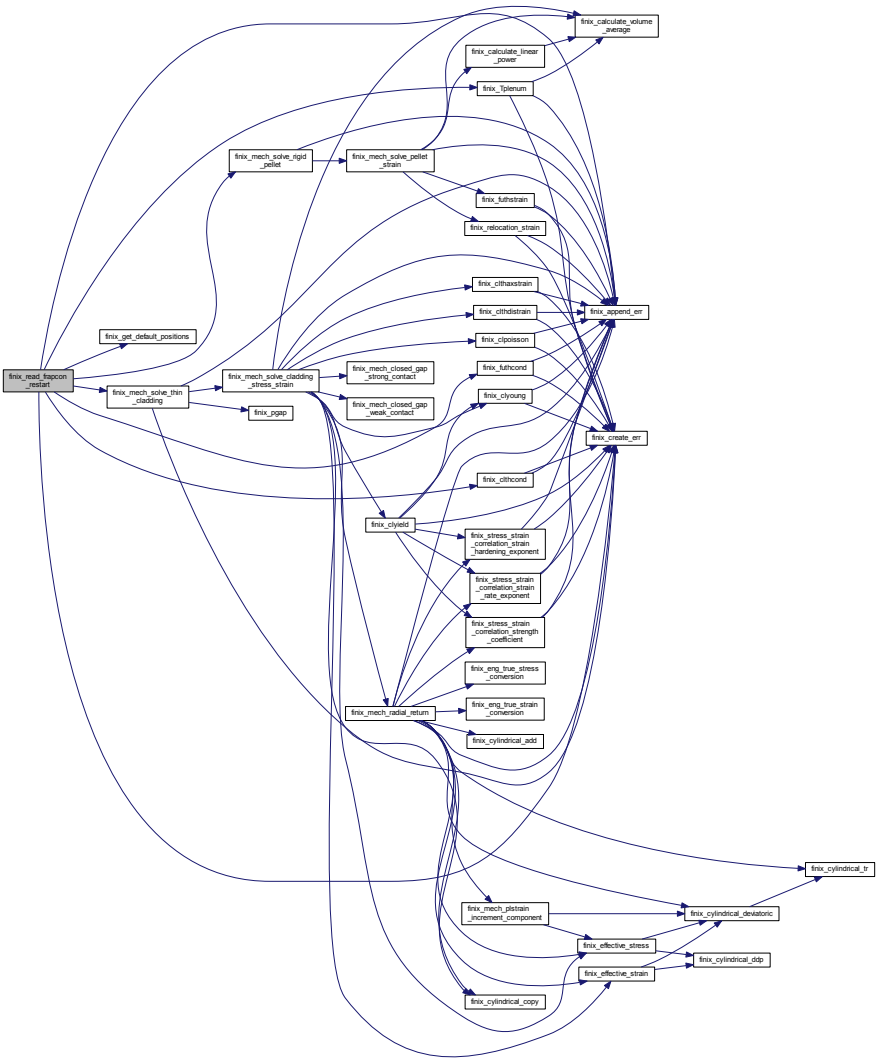
Parameters

options	FINIX calculation options
rod	data structure containing fuel rod properties
bc	coolant boundary conditions
results	miscellaneous computed values
scenario	data structure containing the name of the Frapcon generated restart file

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.7.2.22 char** finix_read_input (Rod * *rod,* Scenario * *scenario,* Options * *options,* int *mode,* int * *existing_files*)

Function reads one input file and strores the data in FINIX data structures

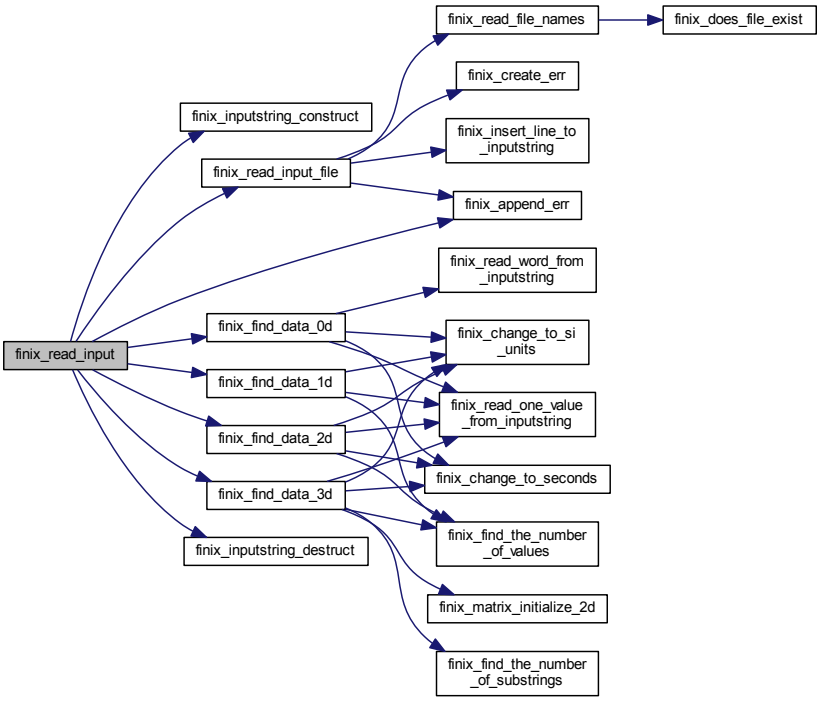
Parameters

rod	data structure containing rod related data
scenario	data structure containing rod history and power data
options	data structure containing the simulation options
mode	an integer defining the input to be read
existing_files	an array containing the information about existing input files

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.7.2.23 char** finix_read_input_file (Rod * *rod*, Scenario * *scenario*, Options * *options*, Input * *input*, int *mode*, int * *existing_files*)

Function reads the contents of an input file (exculding commented lines). The input will be stored in string format in Input data structure.

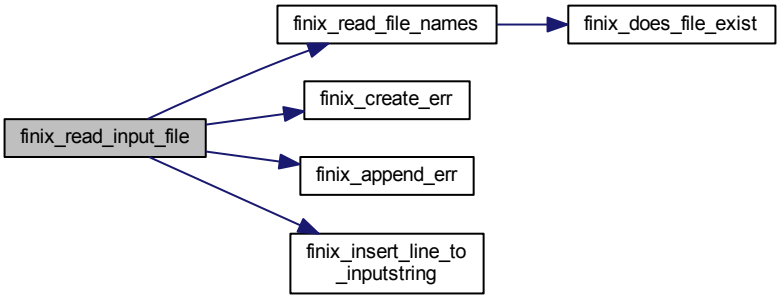
Parameters

rod	data structure containing rod related data
scenario	data structure containing rod history and power data
options	data structure containing the simulation options
input	a data structure containing the contents of an input file
mode	an integer defining the input to be read
existing_files	an array containing the information about existing input files

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.7.2.24 char* finix_read_one_value_from_inputstring (double * *value,* char * *location*)

Function reads one value from an input-string. The value that will be read is pointed by parameter "location", and will be stored in parameter "value".

Parameters

value	the value that will be read from the input-string
location	a pointer to the beginning of the reading block

Returns

pointer to the end of the reading block

2.7.2.25 void finix_read_word_from_inputstring (char * *word,* char * *location*)

Function reads one word from the input-string.

Parameters

word	the word that will be read from the input-string
location	a pointer to the beginning of the reading block

Returns

nothing

2.7.2.26 char** finix_set_defaults_for_bc_and_results (Boundary_conditions * *bc,* Results * *results,* Options * *options,* Rod * *rod*)

Function sets default values for Boundary_conditions and Results data structures.

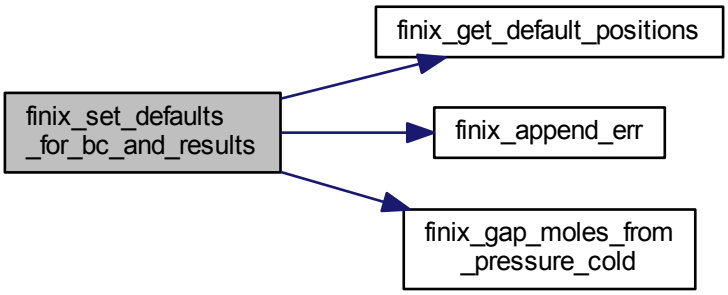
Parameters

bc	data structure containing rod boundary conditions
results	data structure containing FINIX simulation results
options	data structure containing the simulation options
rod	data structure containing rod related data

Returns

error string, NULL for no errors

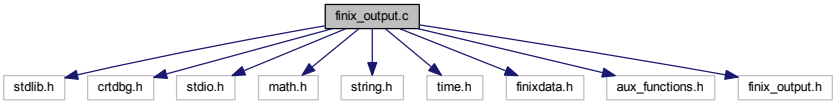
Here is the call graph for this function:



2.8 finix_output.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include "finixdata.h"
#include "aux_functions.h"
#include "finix_output.h"
```

Include dependency graph for finix_output.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- Output * finix_output_initialize (Options *options)
- void finix_output_print (Output *files, Rod *rod, Boundary_conditions *bc, Options *options, Results *results, Scenario *scenario)
- void finix_output_close (Output *files, Options *options)
- char ** finix_db_fprintf_stripfile (int printoption, FILE *writefile, Results *results, Rod *rod, Options *options, Boundary_conditions *bc, Scenario *scenario)
- void finix_print_data_structures (Rod *rod, Boundary_conditions *bc, Scenario *scenario, Results *results, Options *options)

2.8.1 Macro Definition Documentation

2.8.1.1 #define _CRTDBG_MAP_ALLOC

2.8.2 Function Documentation

2.8.2.1 char** finix_db_fprintf_stripfile (int *printoption*, FILE * *writefile*, Results * *results*, Rod * *rod*, Options * *options*, Boundary_conditions * *bc*, Scenario * *scenario*)

Outputs results to writefile in the same format as a FRAPTRAN stripfile. Note that many of the parameters cannot be output from FINIX, so they are set to zero.

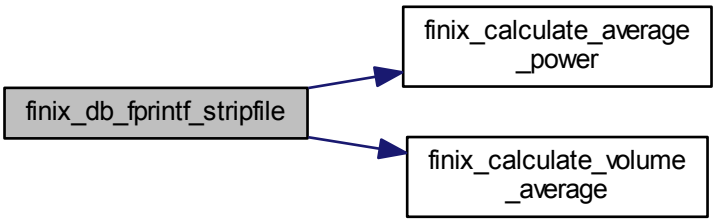
Parameters

printoption	printing option, 0 = print the beginning of the file, 1 = print the data section of the file
writefile	the file to output to
time	time step
results	results calculated by FINIX
rod	data structure containing fuel rod properties
options	FINIX calculation options
bc	boundary conditions
scenario	data structure containing rod history and power data

Returns

always returns NULL

Here is the call graph for this function:



2.8.2.2 void finix_output_close (Output * *files*, Options * *options*)

Function closes the finix.sum and finix.zx output files and frees the memory allocated to Output data structure. This function must be used if function finix_output_initialize has been called earlier.

Parameters

files	a data structure containing an array of file pointers.
options	a data structure containing simulation options.

Returns

nothing

2.8.2.3 Output* finix_output_initialize (Options * *options*)

Function prints the header lines in finix.sum and finix.zx output files. This function must be used if these files are printed with finix_output_print, because it allocates memory for a structure containing an array of file pointers. After calling this function, the memory of the Output structure and the memory of file pointer array must be freed by calling the function finix_output_close.

Parameters

options	a data structure containing simulation options.

Returns

files a data structure containing an array of file pointers.

2.8.2.4 void finix_output_print (Output * *files,* Rod * *rod,* Boundary_conditions * *bc,* Options * *options,* Results * *results,* Scenario * *scenario*)

Function prints the data lines in output files finix.sum and finix.zx. NOTE: because the function uses the file pointers in Output data structure, the data structure must first be initialized by calling the function finix_output_initialize.

Parameters

files	a data structure containing an array of file pointers.
rod	a data structure containing the fuel rod data.
bc	a data structure containing boundary condition data.
options	a data structure containing simulation options.
results	a data structure containing simulation results.

Returns

nothing

2.8.2.5 void finix_print_data_structures (Rod * *rod,* Boundary_conditions * *bc,* Scenario * *scenario,* Results * *results,* Options * *options*)

The function prints all the data stored in FINIX data structures.

Parameters

rod	data structure containing rod related data
bc	data structure containing rod boundary conditions
scenario	data structure containing rod history and power data

results	data structure containing FINIX simulation results
options	data structure containing the simulation options

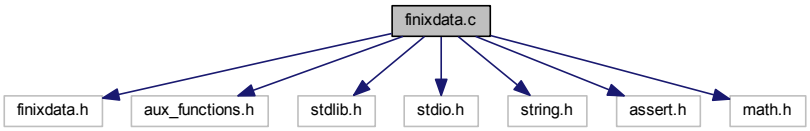
Returns

nothing

2.9 finixdata.c File Reference

```
#include "finixdata.h"
#include "aux_functions.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <math.h>
```

Include dependency graph for finixdata.c:



Functions

- Rod * finix_rod_construct ()
- Boundary_conditions * finix_bc_construct ()
- Scenario * finix_scenario_construct ()
- Results * finix_results_construct ()
- Options * finix_options_construct ()
- void finix_data_structures_destruct (Rod *rod, Boundary_conditions *bc, Scenario *scenario, Results *res, Options *options)
- void finix_rod_destruct (Rod *rod)
- void finix_bc_destruct (Boundary_conditions *bc, Options *opt)
- void finix_scenario_destruct (Scenario *s)
- void finix_results_destruct (Results *res, Options *options)
- void finix_matrixDestruct_2d (double **matrix, int dim1)
- char ** finix_update_bc (Boundary_conditions *bc, Scenario *scen, Options *opt, Rod *rod, Results *res)
- char ** finix_bc_interpolate_zonehistory_data (double **zoneInterp, double **interp, Scenario *scen, Rod *rod, Boundary_conditions *bc, Options *opt, char *option)
- char ** finix_bc_interpolate_ring_data (double **interp, Scenario *scen, Boundary_conditions *bc, Results *res, Options *opt, char *option)
- char ** finix_bc_interpolate_history_data (double **interp, Scenario *scen, Boundary_conditions *bc, Options *opt, char *option)
- char ** finix_bc_interpolate_zone_data (double **interp, Scenario *scen, Rod *rod, Boundary_conditions *bc, Options *opt, char *option)
- Cylindrical finix_cylindrical_copy (Cylindrical t)
- void finix_cylindrical_add (Cylindrical *t, Cylindrical dt)
- void finix_cylindrical_add_scalar (Cylindrical *t, double s)

- void finix_cylindrical_substract (Cylindrical *t, Cylindrical dt)
- void finix_cylindrical_substract_scalar (Cylindrical *t, double s)
- double finix_cylindrical_ddp (Cylindrical t1, Cylindrical t2)
- Cylindrical finix_cylindrical_scalar_multiply (Cylindrical t_orig, double s)
- double finix_cylindrical_tr (Cylindrical t)
- Cylindrical finix_cylindrical_deviatoric (Cylindrical t)

2.9.1 Function Documentation

2.9.1.1 Boundary_conditions* finix_bc_construct ( )

Constructor function for the boundary conditions data structure. The data structure contains data related to rod boundary conditions.

Returns

boundary conditions data structure

2.9.1.2 void finix_bc_destruct (Boundary_conditions * *bc*, Options * *opt*)

Destructor function for the boundary_contions data structure. Frees all boundary_conditions-related memory.

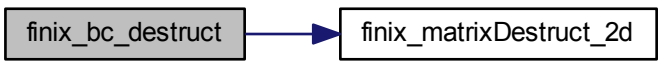
Parameters

bc	data structure containing rod boundary conditions
opt	data structure containing the simulation options

Returns

nothing

Here is the call graph for this function:



2.9.1.3 char** finix_bc_interpolate_history_data (double ** *interp*, Scenario * *scen*, Boundary_conditions * *bc*, Options * *opt*, char * *option*)

Function to interpolate data in Scenario struct members of type struct history_data to Boundary_condition struct.

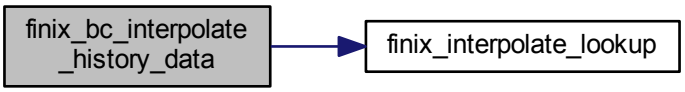
Parameters

interp	array of pointers for interpolation in time
scen	Scenario struct
bc	Boundary_conditions struct
opt	Options struct
option	string to identify the member of Scenario struct

Returns

return_err error string

Here is the call graph for this function:



2.9.1.4 char** finix_bc_interpolate_ring_data (double ** *interp*, Scenario * *scen*, Boundary_conditions * *bc*, Results * *res*, Options * *opt*, char * *option*)

Function to interpolate data in Scenario struct members of type struct ring_data to radial boundary condition arrays (in an array[i][j] i = axial index, j= radial index).

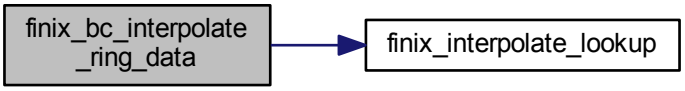
Parameters

interp	array of pointers for interpolation in time
scen	Scenario struct
res	Results struct, the cold radial node positions are used
bc	Boundary_conditions struct
opt	Options struct
option	string to identify the member of Scenario struct

Returns

return_err error string

Here is the call graph for this function:



2.9.1.5 char** finix_bc_interpolate_zone_data (double ** *interp*, Scenario * *scen*, Rod * *rod*, Boundary_conditions * *bc*, Options * *opt*, char * *option*)

Function to interpolate data in Scenario struct members of type struct zone_data to axial boundary condition vectors.

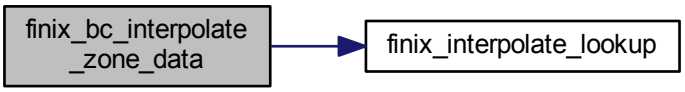
Parameters

interp	array of pointers for interpolation in time
scen	Scenario struct
rod	Rod struct
bc	Boundary_conditions struct
opt	Options struct
option	string to identify the member of Scenario struct

Returns

return_err error string

Here is the call graph for this function:



2.9.1.6 char** finix_bc_interpolate_zonehistory_data (double ** *zoneInterp,* double ** *interp,* Scenario * *scen,* Rod * *rod,* Boundary_conditions * *bc,* Options * *opt,* char * *option*)

Function to interpolate data in Scenario struct members of type struct zonehistory_data to axial boundary condition vectors.

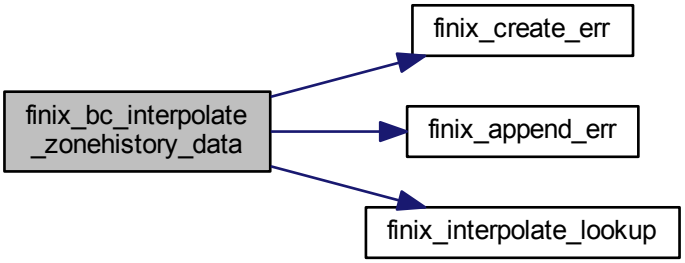
Parameters

zoneInterp	work array for saving values at each zone interpolated to current simulation time
interp	array of pointers for interpolation in time
scen	Scenario struct
rod	Rod struct
bc	Boundary_conditions struct
opt	Options struct
option	string to identify the member of Scenario struct

Returns

return_err error string

Here is the call graph for this function:



2.9.1.7 void finix_cylindrical_add (Cylindrical * *t*, Cylindrical *dt*)

Adds a Cylindrical struct to another Cylindrical struct.

Parameters

t	the original struct in Cylindrical form
dt	the struct containing values to add to t in Cylindrical form

2.9.1.8 void finix_cylindrical_add_scalar (Cylindrical * *t*, double *s*)

Adds a scalar value to a Cylindrical struct.

Parameters

t	the original struct in Cylindrical form
s	a scalar

2.9.1.9 Cylindrical finix_cylindrical_copy (Cylindrical *t*)

Copies a Cylindrical struct to another Cylindrical struct.

Parameters

t	the original struct in Cylindrical form

Returns

c the copied struct in Cylindrical form

2.9.1.10 double finix_cylindrical_ddp (Cylindrical *t1*, Cylindrical *t2*)

Calculates the double dot product of two tensors with zeroes as the non-diagonal elements. The tensors are expressed in Cylindrical form.

Parameters

t1	a struct in Cylindrical form
t2	another struct in Cylindrical form

Returns

product the double dot product

2.9.1.11 Cylindrical finix_cylindrical_deviatoric (Cylindrical *t*)

Calculates the deviatoric values of a tensor in Cylindrical form (with zeroes as the non-diagonal elements).

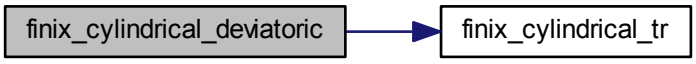
Parameters

t	the tensor in Cylindrical form

Returns

deviatoric the deviatoric tensor in Cylindrical form

Here is the call graph for this function:



2.9.1.12 Cylindrical finix_cylindrical_scalar_multiply (Cylindrical *t_orig*, double *s*)

Multiplies a Cylindrical struct with a scalar.

Parameters

t_orig	the original struct in Cylindrical form
s	the scalar value with which the struct is multiplied

Returns

t_multiplied the multiplied struct in Cylindrical form

2.9.1.13 void finix_cylindrical_substract (Cylindrical * *t*, Cylindrical *dt*)

Substracts a Cylindrical struct form another Cylindrical struct.

Parameters

t	the original struct in Cylindrical form
dt	the struct containing values to substract from t in Cylindrical form

2.9.1.14 void finix_cylindrical_substract_scalar (Cylindrical * *t*, double *s*)

Substracts a scalar value from a Cylindrical struct.

Parameters

t	the struct in Cylindrical form
s	a scalar

2.9.1.15 double finix_cylindrical_tr (Cylindrical *t*)

Calculates the trace of a tensor in Cylindrical form.

Parameters

t	a tensor in Cylindrical form

Returns

tr the trace of tensor t

2.9.1.16 void finix_data_structures_destruct (Rod * *rod*, Boundary_conditions * *bc*, Scenario * *scenario*, Results * *res*, Options * *options*)

Function frees all the memory allocated to FINIX data structures.

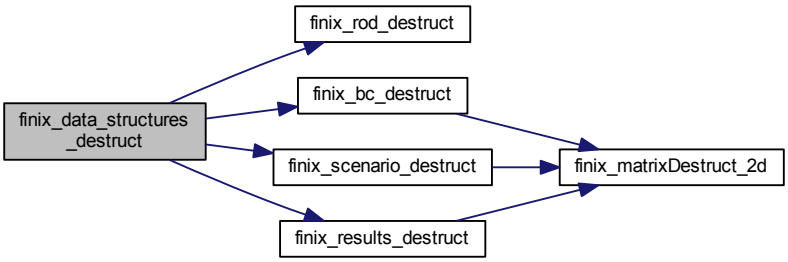
Parameters

rod	data structure containing rod related data
bc	data structure containing rod boundary conditions
scenario	data structure containing rod history and power data
res	data structure containing FINIX simulation results
options	data structure containing the simulation options

Returns

nothing

Here is the call graph for this function:



2.9.1.17 void finix_matrixDestruct_2d (double ** *matrix,* int *dim1*)

Frees the memory of a 2D matrix.

Parameters

matrix	a pointer to the matrix that will be destroyed
dim1	number of matrix columns

Returns

nothing

2.9.1.18 Options* finix_options_construct ( )

Constructor function for the options data structure. The data structure contains the FINIX simulation options.

Returns

options data structure

2.9.1.19 Results* finix_results_construct ( )

Constructor function for the results data structure. The data structure contains the FINIX simulation results.

Returns

results data structure

2.9.1.20 void finix_results_destruct (Results * *res,* Options * *options*)

Destructor function for the results data structure. Frees all results-related memory.

Parameters

res	data structure containing FINIX simulation results
options	data structure containing the simulation options

Returns

nothing

Here is the call graph for this function:

2.9.1.21 Rod* finix_rod_construct ( )

Contructor function for the rod data structure. The data structure contains data related to a fuel rod.

Returns

rod data structure

2.9.1.22 void finix_rod_destruct (Rod * *rod*)

Destructor function for the rod data structure. Frees all rod-related memory.

Parameters

rod	data structure containing rod related data

Returns

nothing

2.9.1.23 Scenario* finix_scenario_construct ( )

Constructor function for the scenario data structure. The data structure contains the rod history and power data.

Returns

scenario data structure

2.9.1.24 void finix_scenario_destruct (Scenario * *s*)

Destructor function for the scenario data structure. Frees all scenario-related memory.

Parameters

s	data structure containing rod history and power data

Returns

nothing

Here is the call graph for this function:

2.9.1.25 char** finix_update_bc (Boundary_conditions * *bc,* Scenario * *scen,* Options * *opt,* Rod * *rod,* Results * *res*)

Updating boundary conditions for the current time step

This function updates the boundary conditions in Boundary_conditions struct with values interpolated from history data in the Scenario struct. All given boundary conditions are updated. An error is shown if boundary conditions other than specified in opt->boundary_condition are given. If time step or node location is below or over the minimum/maximum value given in the history, the first/last time step/node location value is used for all such values.

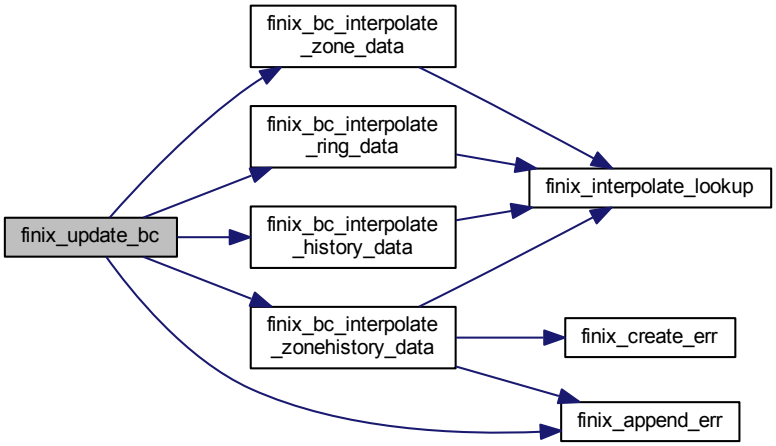
Parameters

bc	boundary condition struct (actually used by FINIX)
scen	scenario struct with history data
opt	options struct
rod	rod parameter struct, rod length and pellet radius used
res	results struct, cold state rod radii are used

Returns

error string

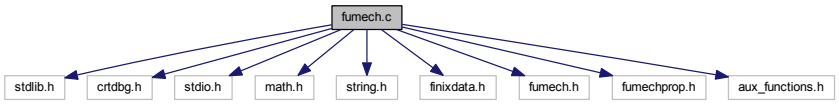
Here is the call graph for this function:



2.10 fumech.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "finixdata.h"
#include "fumech.h"
#include "fumechprop.h"
#include "aux_functions.h"
```

Include dependency graph for fumech.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- char ∗∗ finix_mech_solve_pellet_strain (int zind, double ∗∗power_den, Results ∗results, Options ∗options)
- char ∗∗ finix_mech_solve_rigid_pellet (double ∗∗power_den, Rod ∗rod, Results ∗results, Options ∗options)

2.10.1 Macro Definition Documentation

2.10.1.1 #define _CRTDBG_MAP_ALLOC

2.10.2 Function Documentation

2.10.2.1 char∗∗ finix_mech_solve_pellet_strain (int *zind,* double ∗∗ *power_den,* Results ∗ *results,* Options ∗ *options*)

Solves the strains for one axial node with the rigid pellet appoximation.

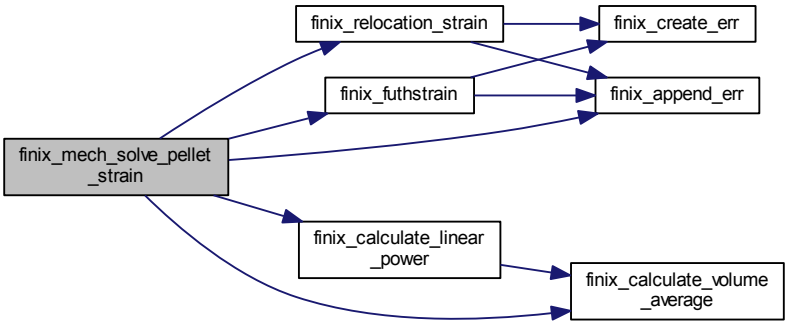
Parameters

zind	index of the axial slice
power_den	power density (W/m^3)
results	computed results
options	FINIX calculation options

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.10.2.2 char** finix_mech_solve_rigid_pellet (double ** *power_den,* Rod * *rod,* Results * *results,* Options * *options*)

Solves the pellet mechanical deformations with the rigid pellet appoximation.

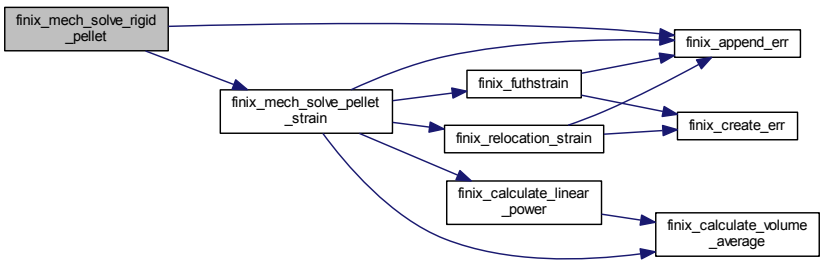
Parameters

power_den	power density (W/m^3)
rod	data structure containing fuel rod properties
results	computed results
options	FINIX calculation options

Returns

error string, NULL for no errors

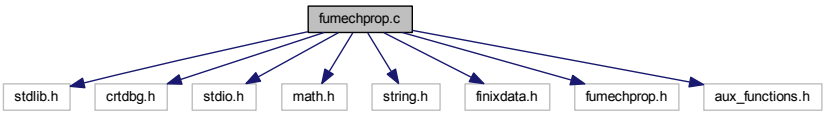
Here is the call graph for this function:



2.11 fumechprop.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "finixdata.h"
#include "fumechprop.h"
#include "aux_functions.h"
```

Include dependency graph for fumechprop.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- char ** finix_futhstrain (double T, double *strain)
- char ** finix_relocation_strain (double lhr, double bu, double rfo, double rci, Options *options, double *strain)
- char ** finix_calculate_density (Options *options, Rod *rod, Results *results, double **den)

2.11.1 Macro Definition Documentation

2.11.1.1 #define _CRTDBG_MAP_ALLOC

2.11.2 Function Documentation

2.11.2.1 char** finix_calculate_density (Options * *options*, Rod * *rod*, Results * *results*, double ** *den*)

Pellet and cladding density calculation from grid point locations assuming conservation of mass. Added 10 June 2013 by Timo Ikonen.

Calculates the radial pellet relocation.

Parameters

options	FINIX calculation options
rod	data structure containing fuel rod properties
results	computed results
den	density at to be calculated for each node (kg/m^3)

Returns

error string, NULL for no errors

2.11.2.2 char** finix_futhstrain (double *T*, double * *strain*)

Fuel thermal strain

Calculates the thermal strain of UO2 with the FRAPTRAN-1.4 correlation.

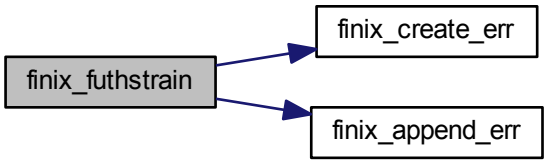
Parameters

T	temperature (K)
strain	thermal strain

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.11.2.3 char** finix_relocation_strain (double *lhr*, double *bu*, double *rfo*, double *rci*, Options * *options*, double * *strain*)

Fuel relocation strain. Added 30 May 2013 by Timo Ikonen.

Calculates the radial pellet relocation.

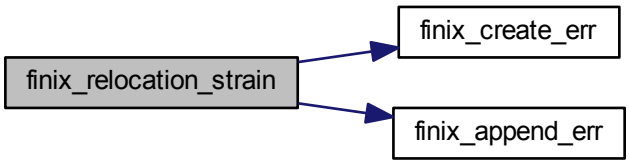
Parameters

lhr	pellet average linear heat rate (W/m)
bu	pellet average burnup (MWd/kgU)
rfo	fuel outer radius in the cold state (m)
rci	cladding inner radius in the cold state (m)
options	model options
strain	relocation strain (output)

Returns

error string, NULL for no errors

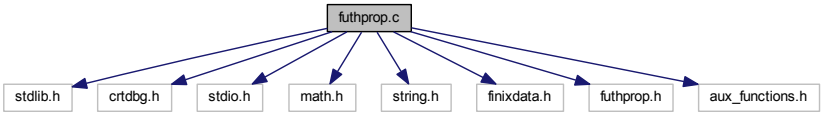
Here is the call graph for this function:



2.12 futhprop.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "finixdata.h"
#include "futhprop.h"
#include "aux_functions.h"
```

Include dependency graph for futhprop.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- char ** finix_futhcond (double T, double Bu, double den, double gad, double *lambda)
- char ** finix_fucp (double T, double *cp)

2.12.1 Macro Definition Documentation

2.12.1.1 #define _CRTDBG_MAP_ALLOC

2.12.2 Function Documentation

2.12.2.1 char** finix_fucp (double *T*, double * *cp*)

Fuel specific heat

Calculates the specific heat of UO2 with the FRAPTRAN-1.4 correlation.

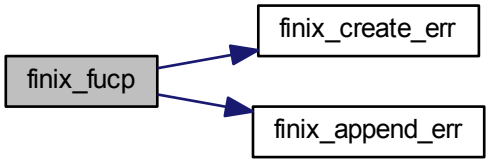
Parameters

T	temperature (K)
cp	specific heat

Returns

error string, NULL for no errors

Here is the call graph for this function:



2.12.2.2 char** finix_futhcond (double *T*, double *Bu*, double *den*, double *gad*, double * *lambda*)

Fuel thermal conductivity

Calculates the thermal conductivity of UO_2 with the FRAPTRAN-1.4 correlation.

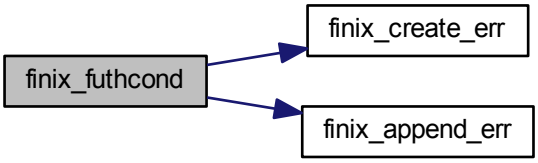
Parameters

T	temperature (K)
den	as-fabricated density as a fraction from the theoretical value
gad	gadolinia weight fraction
Bu	burn-up (GWd/MTU)
lambda	thermal conductivity (W/mK)

Returns

error string, NULL for no errors

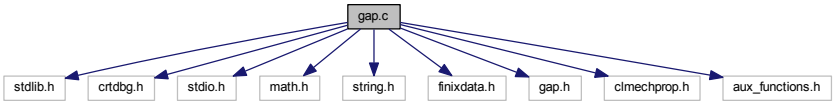
Here is the call graph for this function:



2.13 gap.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "finixdata.h"
#include "gap.h"
#include "clmechprop.h"
#include "aux_functions.h"
```

Include dependency graph for gap.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- char ** finix_hgap (int zind, double **lambda, Rod *rod, Results *results, Options *options)
- char ** finix_Tplenum (Options *options, double **lambda, Results *results, Boundary_conditions *bc)
- char ** finix_pgap (Options *options, Rod *rod, Results *results)
- char ** finix_gap_moles_from_pressure (Options *options, Rod *rod, Results *results)
- char ** finix_gap_moles_from_pressure_cold (Options *options, Rod *rod, Results *results)

2.13.1 Macro Definition Documentation

2.13.1.1 #define _CRTDBG_MAP_ALLOC

2.13.2 Function Documentation

2.13.2.1 char** finix_gap_moles_from_pressure (Options * *options,* Rod * *rod,* Results * *results*)

Calculates the amount of gas (moles) from the gas pressure, temperature and volume.

Parameters

options	FINIX calculation options
rod	data structure containing fuel rod properties
results	results struct, including the fuel pellet axial strains in results->pellet_strain.axial and the gas amount to be calculated in results->gas_mole_amount

Returns

error string, NULL for no errors

2.13.2.2 char** finix_gap_moles_from_pressure_cold (Options * *options,* Rod * *rod,* Results * *results*)

Calculates the amount of gas (moles) from the gas pressure, temperature and volume using cold state (300 K) values.

Parameters

options	FINIX calculation options
rod	data structure containing fuel rod properties
results	results struct, including the fuel pellet axial strains in results->pellet_strain.axial and the gas amount to be calculated in results->gas_mole_amount

Returns

error string, NULL for no errors

2.13.2.3 char** finix_hgap (int *zind,* double ** *lambda,* Rod * *rod,* Results * *results,* Options * *options*)

Gap heat transfer coefficient

Calculates the gap heat transfer coefficient h, taking into account conductive, radiative and contact heat transfer.

23 July 2013: Corrected effetive gap width correlation

Parameters

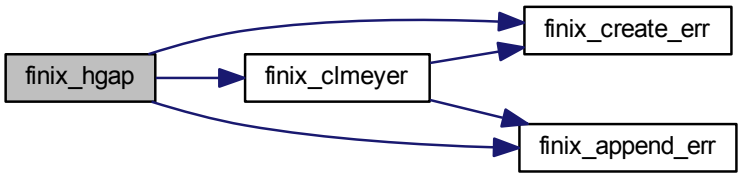
zind	index of the axial slice
lambda	conductivity at each node (W/mK)
rod	data structure containing fuel rod properties
results	computed results
options	FINIX calculation options

Returns

error string, NULL for no errors

currently pure helium is assumed; should add A and B for other gases' conductivities

Here is the call graph for this function:



2.13.2.4 char** finix_pgap (Options * *options,* Rod * *rod,* Results * *results*)

Calculates the rod internal gas pressure with given temperatures and dimensions. Modified 5 June 2013 by Timo Ikonen to calculate current pressure from amount of gas, instead of fill pressure. Allows nonconservation of gas.

Parameters

options	FINIX calculation options
rod	data structure containing fuel rod properties
results	results struct, including the fuel pellet axial strains in results->pellet_strain.axial and the pressure to be calculated in results->fill_gas_pressure

Returns

error string, NULL for no errors

2.13.2.5 char** finix_Tplenum (Options * *options,* double ** *lambda,* Results * *results,* Boundary_conditions * *bc*)

Plenum gas temperature

Calculates the plenum gas temperature. Assumes helium for gas properties.

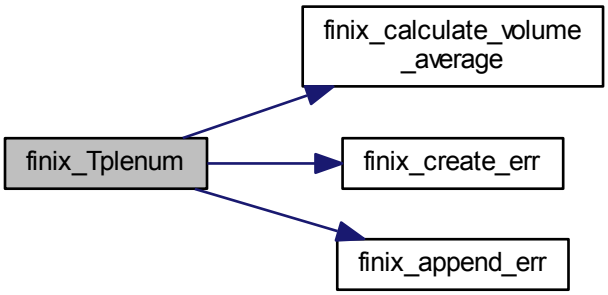
Parameters

options	FINIX calculation options
lambda	heat conductivity at each node
results	calculated results (including plenum temperature at results->fill_gas_temperature and pressure at results->fill_gas_pressure)
bc	boundary conditions (gives, e.g., coolant temperature)

Returns

error string, NULL for no errors

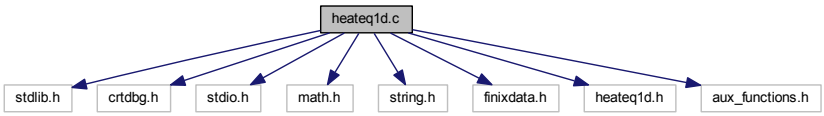
Here is the call graph for this function:



2.14 heateq1d.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "finixdata.h"
#include "heateq1d.h"
#include "aux_functions.h"
```

Include dependency graph for heateq1d.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- char ** finix_FEM_discretize_HE_1D (int zind, double dt, double **power_den, double **lambda, double **cv, Results *results, Boundary_conditions *bc, Options *options, double **diag, double **subdiag, double **superdiag, double **loadvec)
- char ** finix_FEM_solve_tridiagonal (int zind, Options *options, Results *results, double **diag, double **subdiag, double **superdiag, double **loadvec)

2.14.1 Macro Definition Documentation

2.14.1.1 #define _CRTDBG_MAP_ALLOC

2.14.2 Function Documentation

2.14.2.1 char** finix_FEM_discretize_HE_1D (int *zind,* double *dt,* double ** *power_den,* double ** *lambda,* double ** *cv,* Results * *results,* Boundary_conditions * *bc,* Options * *options,* double ** *diag,* double ** *subdiag,* double ** *superdiag,* double ** *loadvec*)

The FEM discretization of the 1D heat equation

Assembles the discretization matrix and load vector of the heat equation in the pellet, gap and cladding, using the given nodalization, material parameters and boundary conditions.

Parameters

zind	index of the axial slice
dt	discretization time step
options	FINIX calculation options
power_den	power_den density at each node (W/m^2)
lambda	conductivity at each node (W/mK)
cv	volumetric heat capacity (J/m^3K)
results	computed results (including gap conductance)
bc	boundary conditions
diag	matrix diagonal (function output; must be initialized beforehand)
subdiag	matrix sub-diagonal (function output; must be initialized beforehand)
superdiag	matrix super-diagonal (function output; must be initialized beforehand)
loadvec	load vector (function output; must be initialized beforehand)

Returns

error string, NULL for no errors

If options->boundary_option==0, use user-given rod outer surface temperature as boundary condition

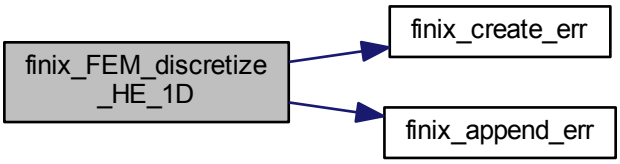
If options->boundary_option==1, use user-given heat flux between rod outer surface and coolant

If options->boundary_option==2, use user-given heat transfer coefficient and coolant bulk temperature as boundary condition

If options->boundary_option==3, use user-given bulk temperature and internally calculated heat transfer coefficient

If options->boundary_option==4, use user-given inlet temperature and mass flux to calculate heat transfer coefficient and coolant bulk temperature downstream (NOT IMPLEMENTED)

Here is the call graph for this function:



2.14.2.2 char** finix_FEM_solve_tridiagonal (int *zind,* Options * *options,* Results * *results,* double ** *diag,* double ** *subdiag,* double ** *superdiag,* double ** *loadvec*)

Solves the 1D FEM discretized heat equation using the tridiagonal matrix (Thomas) algorithm. Does NOT preserve the LHS matrix or the load vector.

Parameters

zind	index of the axial slice
options	FINIX calculation options
results	data structure including parameter temperature (K) (which is the function output to be solved; must be initialized beforehand)
diag	matrix diagonal
subdiag	matrix sub-diagonal
superdiag	matrix super-diagonal
loadvec	load vector

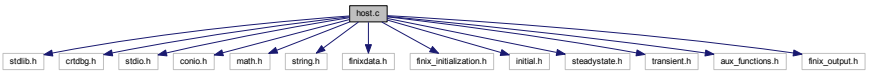
Returns

error string, NULL for no errors

2.15 host.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <string.h>
#include "finixdata.h"
#include "finix_initialization.h"
#include "initial.h"
#include "steadystate.h"
#include "transient.h"
#include "aux_functions.h"
#include "finix_output.h"
```

Include dependency graph for host.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- int main ()

2.15.1 Macro Definition Documentation

2.15.1.1 #define _CRTDBG_MAP_ALLOC

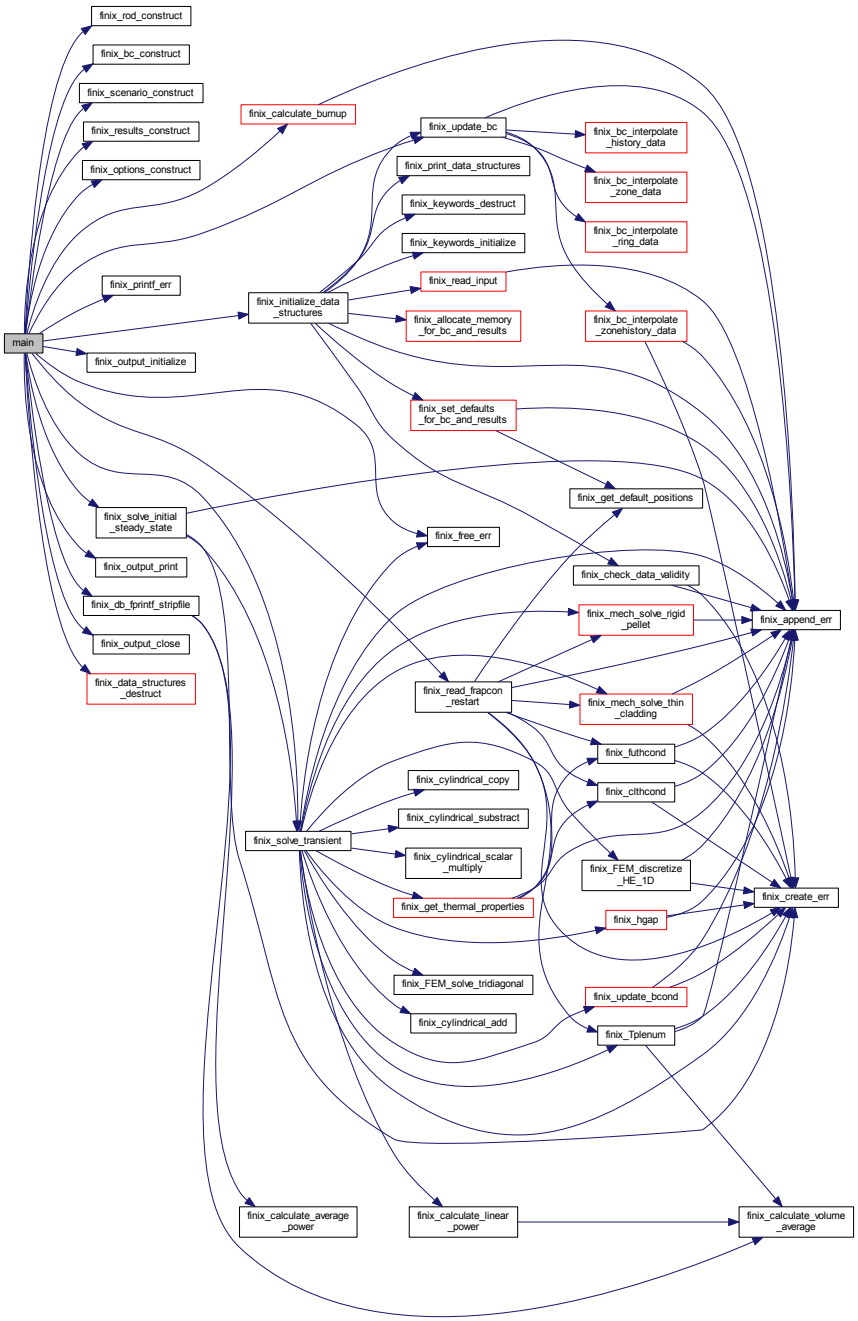
2.15.2 Function Documentation

2.15.2.1 int main ( )

The host code file.

The host code that runs the FINIX subprogram. This file should be replaced in its entirety with the host code.

Here is the call graph for this function:

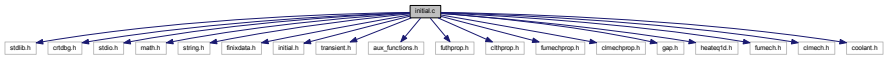


2.16 initial.c File Reference

```
#include <stdlib.h>
```

```
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "finixdata.h"
#include "initial.h"
#include "transient.h"
#include "aux_functions.h"
#include "futhprop.h"
#include "clthprop.h"
#include "fumechprop.h"
#include "clmechprop.h"
#include "gap.h"
#include "heateq1d.h"
#include "fumech.h"
#include "clmech.h"
#include "coolant.h"
```

Include dependency graph for initial.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- char ** finix_solve_initial_steady_state (Rod *rod, Boundary_conditions *bc, Results *results, Options *options)

2.16.1 Macro Definition Documentation

2.16.1.1 #define _CRTDBG_MAP_ALLOC

2.16.2 Function Documentation

2.16.2.1 char** finix_solve_initial_steady_state (Rod * *rod,* Boundary_conditions * *bc,* Results * *results,* Options * *options*)

The initial state function.

This function is used to solve the initial steady state of the fuel rod, with given boundary conditions and power density.

Parameters

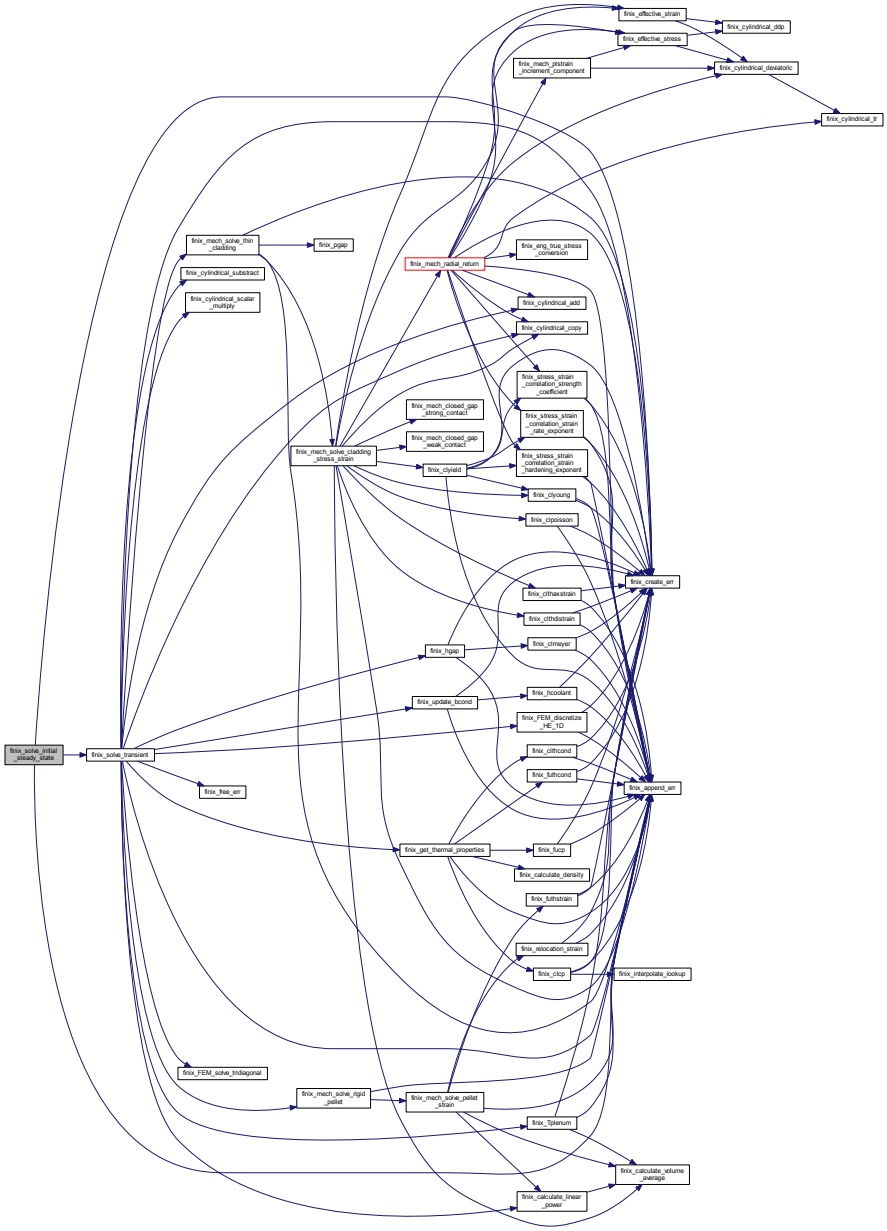
rod	data structure containing fuel rod properties
bc	boundary conditions
results	miscellaneous computed values

options	FINIX calculation options

Returns

error string, NULL for no errors

Here is the call graph for this function:

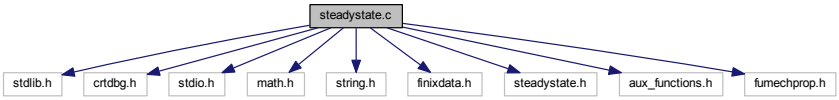


2.17 steadystate.c File Reference

```
#include <stdlib.h>
```

```
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "finixdata.h"
#include "steadystate.h"
#include "aux_functions.h"
#include "fumechprop.h"
```

Include dependency graph for steadystate.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- char ** finix_calculate_burnup (double time, Options *options, Rod *rod, Results *results, Boundary_conditions *bc)

2.17.1 Macro Definition Documentation

2.17.1.1 #define _CRTDBG_MAP_ALLOC

2.17.2 Function Documentation

2.17.2.1 char** finix_calculate_burnup (double *time*, Options * *options*, Rod * *rod*, Results * *results*, Boundary_conditions * *bc*)

Calculates the accumulation of burnup during one constant-power time step.

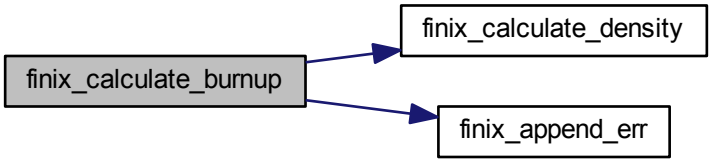
Parameters

time	duration (time step) of burnup accumulation
options	FINIX calculation options
rod	data structure containing fuel rod properties
results	miscellaneous computed values
bc	boundary condition data structure

Returns

error string, NULL for no errors

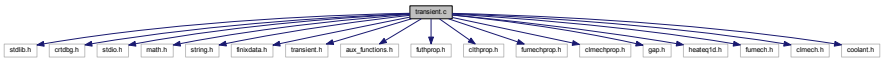
Here is the call graph for this function:



2.18 transient.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "finixdata.h"
#include "transient.h"
#include "aux_functions.h"
#include "futhprop.h"
#include "clthprop.h"
#include "fumechprop.h"
#include "clmechprop.h"
#include "gap.h"
#include "heateq1d.h"
#include "fumech.h"
#include "clmech.h"
#include "coolant.h"
```

Include dependency graph for transient.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- char ** finix_solve_transient (double dt, Rod *rod, Boundary_conditions *bc, Results *results, Options *options)
- char ** finix_get_thermal_properties (double **power_den, Rod *rod, Results *results, Options *options, double **lambda, double **cp, double **den, double **cv)

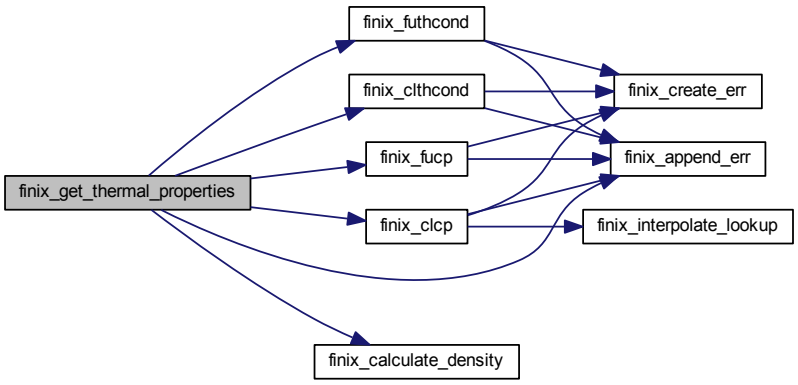
2.18.1 Macro Definition Documentation

2.18.1.1 #define _CRTDBG_MAP_ALLOC

2.18.2 Function Documentation

2.18.2.1 char** finix_get_thermal_properties (double ** *power_den*, Rod * *rod*, Results * *results*, Options * *options*, double ** *lambda*, double ** *cp*, double ** *den*, double ** *cv*)

Here is the call graph for this function:



2.18.2.2 char** finix_solve_transient (double *dt*, Rod * *rod*, Boundary_conditions * *bc*, Results * *results*, Options * *options*)

The main transient behavior function.

This function is used to solve the transient behavior of the fuel rod. The function gives the temperature distribution and the dimensions of the pellet and the cladding at a specified time.

Parameters

dt	time step (seconds); the function solves the thermal and mechanical time evolution of the fuel rod and propagates solution by dy in time
rod	data structure containing fuel rod properties
bc	boundary conditions
results	miscellaneous computed values
options	FINIX calculation options

Returns

error string, NULL for no errors

Here is the call graph for this function: