RESEARCH REPORT

VTT-R-06563-13

FINIX - Fuel behavior model and interface for multiphysics applications - Code documentation for version 0.13.9

Authors: Timo Ikonen

Confidentiality: Public

Report's title	
FINIX - Fuel behavior model and interface for multiphysics applications - Code documentation for version 0.13.9	
Customer, contact person, address VYR	Order reference 25/2013SAF
Project name Extensive Fuel Modelling (Polttoaineen laaja-alainen mallinnus)	Project number/Short name 77462 / PALAMA
Author(s) Timo Ikonen	Pages 105
Keywords fuel behaviour modelling, FINIX	Report identification code VTT-R-06563-13

Summary

The FINIX fuel behavior code has been updated to version 0.13.9. The focus of the update has been to enhance FINIX's modeling capabilities at accumulated burnup and to improve numerical stability.

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 VTT's 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. FINIX has been integrated into VTT's Monte Carlo reactor physics code Serpent, and reactor dynamics codes TRAB and TRAB-3D.

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 validated against the FRAPTRAN fuel performance code in RIA scenarios, and 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 30.9.2013

Written by	Reviewed by	Accepted by
Timo Ikonen Research Scientist	Ville Tulkki Senior Scientist	Timo Vanttola Technology Manager

VTT's contact address
P.O.Box 1000, Tietotie 3, 02044 VTT, Espoo

Distribution (customer and VTT)
SAFIR2014/TR3
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.

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].

2 General model description

2.1 Version

2.1.1 Current version

This document describes version 0.13.9 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, 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 thin 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 be completely elastic (apart from thermal expansion). Plastic deformations due to extremely high stresses or creep are not modeled. As of version 0.13.9, plastic strains can be given as user input, but their values are not updated during the simulation. The model is therefore inadequate to model transient ruptures, ballooning of other strong deformations, and on the other hand, 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 0.13.9

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 [3] 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:

2013

- T. Ikonen, V. Tulkki, E. Syrjähti, V. Valtavirta and J. Leppänen, *FINIX – Fuel Behavior Model and Interface for Multiphysics Applications* [4]. Brief description of the FINIX code, its purpose as a universal fuel behavior module in multiphysics applications and first results. Concerns version 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),

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

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 [5]. For the pellet-plenum heat transfer, the coefficient is

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

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 $\frac{\partial}{\partial r}\left(\lambda_{\text{gap}}(T) r \frac{\partial T}{\partial r}\right) = 0$. Integrating once with respect to r and applying $\lambda_{\text{gap}} \frac{\partial T}{\partial r}\big|_{r=R_f} = h_{\text{cond}}(T_{ci} - T_f)$ gives

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

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 $\int_{T_f}^{T_{ci}} \lambda_{\text{gap}}(T) dT \equiv \bar{\lambda}(T_{ci}, T_f)(T_{ci} - T_f)$, one obtains

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

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:

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

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: $\bar{\lambda}(T_{ci}, T_f) \approx \lambda_{\text{gap}}(T_{\text{gap}})$. 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 [6]

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

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

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

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

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

where d_{eff} is given by Eq. (15) and λ_{eff} is calculated for the gas mixture of n species as [7]

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

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

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

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

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.

For the water vapour (steam), the heat conductivity is given by [7]

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

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

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

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 [8]. 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

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

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

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

and the gap volume

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

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 [5]). 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. **??**. 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 outer surface of the pellet for the thermal strain. This amounts to assuming that the dishing of the pellet ends is strong enough, so that the contact between consecutive pellets is close to the surface. 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 [5, 6], although some further simplifications have been made. Most notably, the cladding is assumed to behave elastically. Therefore no plastic deformations are modeled. This simplifies the model and makes computation of the mechanical equilibrium faster, but limits FINIX to situations where the stresses do not exceed the yield point of the cladding, and to

relatively short transients (since the time-dependent creep of the cladding is not included in the model).

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.

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

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

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

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

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

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

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

Here ϵ_i^{th} are the cladding thermal strains (see Sec. 5.2.3), and ϵ_i^{pl} the cladding plastic strains (not updated in FINIX 0.13.9), 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, $(R_{co} + R_{ci})/2 \equiv \bar{R} = (1 + \epsilon_\theta)\bar{R}_{\text{cold}}$ and the change in the cladding thickness, $R_{co} - R_{ci} = (1 + \epsilon_r)(R_{co,\text{cold}} - R_{ci,\text{cold}})$. The resulting expressions for R_{ci} and R_{co} are

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

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

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

In principle, the open gap model of Eqs. (33)–(40) should be 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}. However, since the whole open gap model is solved iteratively with the pressure P, it is sufficient to proceed through Eqs. (33)–(40) only once for each iteration. The solution then converges with respect to R_{ci} and R_{co}, as well as P.

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

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

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

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

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

Finally, the contact pressure is calculated from

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

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.

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,

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

using Newton-Raphson iteration [3]. The stresses and strains can then be solved from

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

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

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

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

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

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).

5 Material correlations

The material correlations used in FINIX have been mostly adopted from the MATPRO library [7] and the FRAPTRAN fuel performance code [5]. 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. [7, 9].

5.1 Fuel properties

5.1.1 Specific heat

The fuel specific heat correlation is the same as in FRAPTRAN and MATPRO [5, 7]. The specific heat c_m is given as

$$c_m = K_1 \frac{\Theta^2 \exp(\Theta/T)}{T^2 \left[\exp(\Theta/T) - 1\right]^2} + K_2 T + K_3 \frac{Y E_d}{2RT^2} \exp(-E_d/RT), \tag{55}$$

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. [7]) is used for the fuel thermal conductivity. The thermal conductivity λ is given by

$$\lambda = 1.0789 \frac{d}{1 + 0.5(1-d)} \lambda_{95}, \tag{56}$$

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

$$\lambda_{95} = \left[A + a \cdot gad + BT + f(Bu) + \left(1 - 0.9e^{-0.04Bu}\right) g(Bu) h(T)\right]^{-1} + \frac{E}{T^2} e^{-F/T}. \tag{57}$$

Here gad is the gadolinia weight fraction, Bu is the burnup (GWd/MTU) and T is the temperature (K). The functions introduced in Eq. (57) 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. (57) 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

$$\epsilon_{th} = K_1 T - K_2 + K_3 e^{-E_d/k_B T}, \tag{58}$$

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. [7] 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

$$\Delta G/G = \begin{cases} 0.3 + 0.1 f(Bu), & \text{for } LHR < 20 \text{ kW/m}, \\ 0.28 + g(LHR) + [0.12 + g(LHR)] f(Bu), & \text{for } 20 \text{ kW/m} < LHR < 40 \text{ kW/m}, \\ 0.32 + 0.18 f(Bu), & \text{for } LHR > 40 \text{ kW/m}, \end{cases} \tag{59}$$

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

Table 2. *The specific heat capacity of the cladding [7]. 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

The 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. [7]. 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.

5.2.2 Thermal conductivity

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

$$\lambda = 7.51 + 2.09 \cdot 10^{-2}T - 1.45 \cdot 10^{-5}T^2 + 7.67 \cdot 10^{-9}T^3, \tag{60}$$

where the temperature T is given in Kelvin and the conductivity λ in W/mK [7]. For temperatures higher than 2098 K, the conductivity is

$$\lambda = 36.0 \text{ (W/mK)}. \tag{61}$$

5.2.3 Thermal strain

The correlation for 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.

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

$$\epsilon^{\alpha}_{\text{axial}} = -1.1924085 \cdot 10^{-4} + (T - 273.15) \cdot 4.441 \cdot 10^{-6}, \tag{62}$$

$$\epsilon^{\alpha}_{\text{diametral}} = -1.80459 \cdot 10^{-4} + (T - 273.15) \cdot 6.721 \cdot 10^{-6}, \tag{63}$$

and for $T \geq 1273$ K

$$\epsilon^{\beta}_{\text{axial}} = -8.3942 \cdot 10^{-3} + (T - 273.15) \cdot 9.70 \cdot 10^{-6}, \tag{64}$$

$$\epsilon^{\beta}_{\text{diametral}} = -6.7432 \cdot 10^{-3} + (T - 273.15) \cdot 9.70 \cdot 10^{-6}. \tag{65}$$

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

$$\epsilon^{\alpha-\beta}_{\text{axial}} = \frac{1273\text{K} - T}{200\text{K}}\epsilon^{\alpha}_{\text{axial}} + \frac{T - 1073\text{K}}{200\text{K}}\epsilon^{\beta}_{\text{axial}}, \tag{66}$$

$$\epsilon^{\alpha-\beta}_{\text{diametral}} = \frac{1273\text{K} - T}{200\text{K}}\epsilon^{\alpha}_{\text{diametral}} + \frac{T - 1073\text{K}}{200\text{K}}\epsilon^{\beta}_{\text{diametral}}. \tag{67}$$

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 (note the error in sign of last term in [7]):

$$H_M = \exp\left[26.034 + T(-0.026394 + T(4.3502 \cdot 10^{-5} - 2.5621 \cdot 10^{-8}T))\right], \tag{68}$$

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

5.2.5 Young's modulus

The Young's modulus E is given separately for the α and β phases [7]. Below 1094 K, the correlation used is

$$E^\alpha = (1.088 \cdot 10^{11} - 5.475 \cdot 10^7 T + K_1 + K_2)/K_3, \tag{69}$$

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

$$K_1 = (6.61 \cdot 10^{11} + 5.912 \cdot 10^8 T)\Delta, \tag{70}$$

$$K_2 = -2.6 \cdot 10^{10} C, \tag{71}$$

$$K_3 = 0.88 + 0.12 e^{-\Phi/10^{25}}. \tag{72}$$

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

$$E^\beta = 9.21 \cdot 10^{10} - 4.05 \cdot 10^7 T. \tag{73}$$

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

$$E^{\alpha-\beta} = \frac{1239\text{K} - T}{(1239 - 1094)\text{K}} E^\alpha + \frac{T - 1094\text{K}}{(1239 - 1094)\text{K}} E^\beta. \tag{74}$$

5.2.6 Poisson's ratio

The correlation for the Poisson's ratio ν is taken from FRAPTRAN [7]:

$$\nu = 0.42628 - 5.556 \cdot 10^{-5} T. \tag{75}$$

The temperature T is given in Kelvin, and ν is dimensionless.

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 0.13.9 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. [10]. Compared to the FRAPTRAN correlations, the numerical values are very similar.

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

$$\mu = 3.674 \cdot 10^{-7} T^{0.7} \text{ (kg/ms)}, \tag{76}$$

$$\rho = 48.14 \cdot 10^{-5} \frac{P}{T} \left[1 + 0.4446 \cdot 10^{-5} \frac{P}{T^{1.2}} \right] \text{ (kg/m}^3\text{)}, \tag{77}$$

$$Pr = \frac{0.717}{1 + 1.123 \cdot 10^{-8} P} T^{-\left(0.01 - 1.42 \cdot 10^{-9} P\right)} \text{ (dimensionless)}. \tag{78}$$

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 [11], 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 [6]

$$\begin{aligned} h_{\mathrm{DB}} =& [(-5.1889 \cdot 10^{-5} + 6.5044 \cdot 10^{-8} T_w) T_w + (3.5796 \cdot 10^{-7} - 1.0337 \cdot 10^{-9} T_w) P_w \\ &+ 3.2377 \cdot 10^{-2}] \phi_w^{0.8} D_e^{-0.2}, \end{aligned} \tag{79}$$

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,

$$D_e = \begin{cases} 2\frac{d^2-\pi R_{co}^2}{\pi R_{co}}, & \text{for square lattice,} \\ \frac{\sqrt{3}d^2-2\pi R_{co}^2}{\pi R_{co}}, & \text{for hexagonal lattice,} \end{cases} \tag{80}$$

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 [5]

$$h_{\mathrm{Thom}} = \left(\frac{e^{P_w/1260}}{0.072}(T_{co} - T_{sat})\right)^2 / (T_{co} - T_w), \tag{81}$$

with T_{sat} given by the Frapcon correlation

$$T_{sat} = \left[((0.4616716 \cdot 10^{-8} P_w - 0.4424529 \cdot 10^{-4})P_w) + 0.19042968\right] P_w + 394.03519, \tag{82}$$

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 [11], 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

$$q_k > \frac{A - x_k}{C} \tag{83}$$

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),

$$x_k = \frac{h_k - h_{sat}}{h_{vap}} \tag{84}$$

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. (84) are given by

$$A = 0.5328(P_w/P_{crit})^{0.1212}\phi^{-0.3040-0.3285(P_w/P_{crit})}, \tag{85}$$

$$C = 1.6151(P_w/P_{crit})^{1.4066}\phi^{0.4843-2.0749(P_w/P_{crit})}, \tag{86}$$

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. [3].

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

$$C_V[T(r)]\frac{\partial T}{\partial t} - \frac{1}{r}\frac{\partial}{\partial r}\left[\lambda[T(r)]r\frac{\partial T}{\partial r}\right] - s(r) = 0, \tag{87}$$

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 (87) is discretized with the Finite Element Method (FEM) [12]. 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

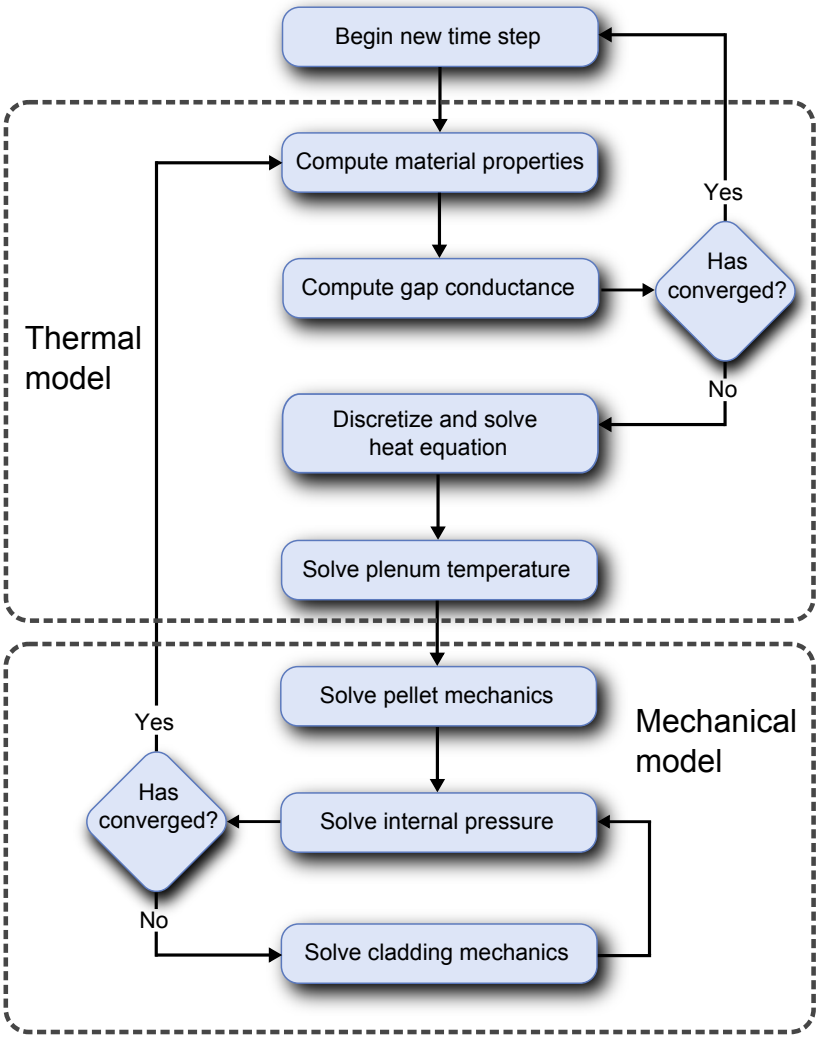


Figure 1. *Flowchart depicting the logic and execution order of the main FINIX modules. Computation of a new time step begins from the top and proceeds through the modules as indicated by the arrows. Convergence checks are made for the gap conductance in the thermal model, and for the internal pressure 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 [12]. 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

$$T(r) \approx [N_i(r)\ N_{i+1}(r)] \begin{bmatrix} T_i \\ T_{i+1} \end{bmatrix}, \tag{88}$$

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

$$N_i(r) = \frac{r_{i+1} - r}{r_{i+1} - r_i}, \tag{89}$$

$$N_{i+1}(r) = \frac{r - r_i}{r_{i+1} - r_i}. \tag{90}$$

For Eq. (87), the Galerkin method results in the matrix equation

$$\int \begin{bmatrix} N_i(r) \\ N_{i+1}(r) \end{bmatrix} C_V(r)\,[N_i(r)N_{i+1}(r)]\,dV \frac{\partial}{\partial t} \begin{bmatrix} T_i \\ T_{i+1} \end{bmatrix} - \begin{bmatrix} \lambda_{i,i} & \lambda_{i,i+1} \\ \lambda_{i+1,i} & \lambda_{i+1,i+1} \end{bmatrix} \begin{bmatrix} T_i \\ T_{i+1} \end{bmatrix} - \int \begin{bmatrix} N_i(r) \\ N_{i+1}(r) \end{bmatrix} s(r)dV = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \tag{91}$$

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

$$\lambda_{i,j} = -2\pi\Delta z \int_{r_i}^{r_j} r\lambda(r) \frac{\partial N_i(r)}{\partial r} \frac{\partial N_j(r)}{\partial r} dr + 2\pi\Delta z \Big|_{r_i}^{r_j} r\lambda(r) N_i(r) \frac{\partial N_j(r)}{\partial r}, \tag{92}$$

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

$$C_V(r) \approx C_i + (C_{i+1} - C_i) \frac{r - r_i}{r_{i+1} - r_i}, \tag{93}$$

$$\lambda(r) \approx \lambda_i + (\lambda_{i+1} - \lambda_i) \frac{r - r_i}{r_{i+1} - r_i}, \tag{94}$$

$$s(r) \approx s_i + (s_{i+1} - s_i) \frac{r - r_i}{r_{i+1} - r_i}, \tag{95}$$

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

Integration over the element gives the matrix equation

$$(\mathbf{K}_i + \mathbf{C}_i) \begin{bmatrix} T_i^{k+1} \\ T_{i+1}^{k+1} \end{bmatrix} = \mathbf{C}_i \begin{bmatrix} T_i^k \\ T_{i+1}^k \end{bmatrix} + \mathbf{f}_i, \tag{96}$$

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 [12, 13]. The matrices in Eq. (96) are defined as follows:

$$\mathbf{K}_i = \frac{\lambda_i(2r_i + r_{i+1}) + \lambda_{i+1}(r_i + 2r_{i+1})}{6(r_{i+1} + r_i)} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \tag{97}$$

$$\mathbf{C}_i = \frac{r_{i+1} - r_i}{60\delta t} \begin{bmatrix} c_i(12r_i + 3r_{i+1}) + c_{i+1}(3r_i + 2r_{i+1}) & c_i(3r_i + 2r_{i+1}) + c_{i+1}(2r_i + 3r_{i+1}) \\ c_i(3r_i + 2r_{i+1}) + c_{i+1}(2r_i + 3r_{i+1}) & c_i(2r_i + 3r_{i+1}) + c_{i+1}(3r_i + 12r_{i+1}) \end{bmatrix}, \tag{98}$$

$$\mathbf{f}_i = \frac{r_{i+1} - r_i}{12} \begin{bmatrix} s_i(3r_i + r_{i+1}) + s_{i+1}(r_i + r_{i+1}) \\ s_i(r_i + r_{i+1}) + s_{i+1}(r_i + 3r_{i+1}) \end{bmatrix} + \begin{bmatrix} -r_i q_i \\ r_{i+1} q_{i+1} \end{bmatrix}. \tag{99}$$

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

$$\mathbf{K}_{n_f} = \begin{bmatrix} hr_{n_f} & -hr_{n_f} \\ -hr_{n_f} & hr_{n_f} \end{bmatrix}. \tag{100}$$

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:

$$\mathbf{K}_i \equiv \begin{bmatrix} \mathrm{K}_i^{(11)} & \mathrm{K}_i^{(12)} \\ \mathrm{K}_i^{(21)} & \mathrm{K}_i^{(22)} \end{bmatrix}, \ \mathbf{C}_i \equiv \begin{bmatrix} \mathrm{C}_i^{(11)} & \mathrm{C}_i^{(12)} \\ \mathrm{C}_i^{(21)} & \mathrm{C}_i^{(22)} \end{bmatrix}. \tag{101}$$

Then, the global matrices are of the tridiagonal form

$$\mathbf{K} = \begin{bmatrix} \mathrm{D}_1 & \mathrm{U}_1 & & & & \mathbf{0} \\ \mathrm{L}_1 & \mathrm{D}_2 & \mathrm{U}_2 & & & \\ & \mathrm{L}_2 & \mathrm{D}_3 & \mathrm{U}_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \mathrm{L}_{n-2} & \mathrm{D}_{n-1} & \mathrm{U}_{n-1} \\ \mathbf{0} & & & & \mathrm{L}_{n-1} & \mathrm{D}_n \end{bmatrix}, \tag{102}$$

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

$$\mathrm{D}_i = \mathrm{K}_i^{(11)} + \mathrm{K}_{i-1}^{(22)} \text{ (for } 2 \le i \le n-1); \ \mathrm{D}_1 = \mathrm{K}_1^{(11)}; \ \mathrm{D}_n = \mathrm{K}_{n-1}^{(22)}, \tag{103}$$

$$\mathrm{U}_i = \mathrm{K}_i^{(12)} \text{ (for } 1 \le i \le n-1), \tag{104}$$

$$\mathrm{L}_i = \mathrm{K}_i^{(21)} \text{ (for } 1 \le i \le n-1). \tag{105}$$

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

$$\mathbf{f} = \begin{bmatrix} \mathrm{f}_1^{(1)} \\ \mathrm{f}_1^{(2)} + \mathrm{f}_2^{(1)} \\ \mathrm{f}_2^{(2)} + \mathrm{f}_3^{(1)} \\ \vdots \\ \mathrm{f}_{n-2}^{(2)} + \mathrm{f}_{n-1}^{(1)} \\ \mathrm{f}_{n-1}^{(2)} \end{bmatrix}, \tag{106}$$

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

$$\mathbf{f}_i \equiv \begin{bmatrix} \mathrm{f}_i^{(1)} \\ \mathrm{f}_i^{(2)} \end{bmatrix}. \tag{107}$$

Finally, the global matrix equation for the complete system is

$$(\mathbf{K} + \mathbf{C})\mathbf{T}^{k+1} = \mathbf{C}\mathbf{T}^k + \mathbf{f}, \tag{108}$$

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

$$\mathbf{T}^k \equiv \begin{bmatrix} T_1^k \\ T_2^k \\ \vdots \\ T_n^k \end{bmatrix}. \tag{109}$$

6.2.5 Time discretization

Implicit time discretization of the heat equation is implied in Eq. (108). The time discretization follows the standard implicit finite difference discretization (see, *e.g.*, Ref. [13]), 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. (108), 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. (99). For the cladding outer surface, several alternative boundary conditions can be used.

Dirichlet boundary condition For a fixed surface temperature, $T(R_{co}) = T_{\text{surf}}$, 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 $\mathrm{f}^{(2)}_{n-1} = T_{\text{surf}}$ in the load vector $\mathbf{f}$. This is equivalent with the the equation $T_n^{k+1} = T_{\text{surf}}$.

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 $q_n = h_{co}\left[T_n - T_{\text{coolant}}\right]$. In this case, the last element of $\mathbf{f}$ is $\mathrm{f}^{(2)}_{n-1} = r_n h_{co} T_{\text{coolant}}$, 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. (108), 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. [3].

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 [3]. 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.

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 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 is possible to initialize FINIX for accumulated burnup using FRAPCON-generated FRAPTRAN restart 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

For any physical quantity, 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 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. However, the burn-up (in MWd/kgU) should of course be based on the fission power.

The cold state in FINIX specifies 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, it is required to continue the FINIX simulation for several consecutive time steps. All these data have to be transferred between FINIX and the host code via functions calls. Therefore, the "secondary" data are packaged into multidimensional data arrays, and only the pointers to these arrays are passed as arguments of the FINIX functions.

The purposes of the most important data arrays are described below. **A more detailed code documentation is given in Appendix A.**

The arrays **r** and **r_cold** contain the spatial locations of the radial and axial nodes in the current state (**r**) and in the cold state of 300 Kelvin (**r_cold**).

The array **T** contains the temperature distribution evaluated at each axial and radial node.

The array **nnodes** contains the number of axial nodes, and the number of radial nodes for the pellet and cladding.

The array **options** contains the model options, such as the coolant boundary condition to be used, and toggles for switching off various sub-models.

The array **params** is an array of type double**, that contains miscellaneous scalar parameters. These include the rod dimensions, pellet and cladding properties, fill gas parameters and coolant conditions. The values are not updated by FINIX, but remain constant. A detailed list of the array

elements is given in Appendix A.

The array **sresults** is an array of type double*, that contains miscellaneous scalar results calculated by FINIX. The values are updated in the course of the simulation. A detailed list of the array elements is given in Appendix A.

The array **vresults** is an array of type double**, and contains vector quantities (each element of the vector corresponds to different axial node) given by the thermal and mechanical solution. A detailed list of the array elements is given in Appendix A.

The array **bcond**, of type double**, contains the numerical values of the boundary conditions for each axial node. The possible boundary conditions are the cladding outer surface temperature, bulk coolant temperature, heat flux from the cladding to the coolant, and the heat transfer coefficient between the cladding the coolant.

The array **linear_power**, of type double*, contains the user-given linear power for each axial node. Linear power in each axial node is conserved in FINIX and assumed constant over one time step.

The array **power_dist**, of type double**, contains the user-given radial power distribution for each node. The values of the power distribution are dimensionless and are normalized so that their area-weighted sum over the radial elements gives the cross-sectional area of the pellet. In FINIX, the linear power and the radial power distribution are used to calculate the power density for each radial and axial node so that the linear power is conserved.

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, the user must declare the necessary data structures in the host code, initialize the arrays using FINIX functions, set the simulation parameters using FINIX functions or by hand, setup the initial state of the simulation using FINIX functions (or by hand), and finally run the model for as many time steps as is needed. FINIX comes with an example host code file, *host.c*, which should be replaced in its entirety by the host code. However, the *host.c* file gives examples on how to setup the model. The minimal necessary steps are also described below.

The host code should declare the data arrays used by FINIX, so that the the necessary data can be retained. In addition, the host code needs to specify the power history, the type of rod to be modeled (to retrieve default parameters for the particular rod type later on), and the simulation time step. Currently, FINIX has no time-step control routines, so 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 definition of the variables can be done as follows:

```
char **err=NULL, **new_err=NULL;//error message strings

//power information, should be supplied by the host program
double plin=20000.0;

double dt=0.001; //time step

//defines the type of fuel rod / power plant. should be given by the host program
//this is used to get various default values for system parameters needed by FINIX.
//the supported types are:
// 0 = US PWR (TMI-1)
int rodtype=0;

//System parameters; pointers need to be declared, but not initialized
int *nnodes;

double **T;
double **r;
double **r_cold;
double **power_dist;
double *linear_power;
double **burnup;

double **params;
double **bcond;
double **vresults;
double *sresults
int *options;

//streams for writing data to disk
//these are standard, not needed to execute the FINIX model
```

```
//(if the output is not written to disk)
FILE *Tfile;
Tfile=fopen("T.txt","w");
```

After the variables have been declared, they need to be initialized. This is done in two steps, using FINIX functions. First, memory is allocated for the arrays with the *finix_initialize_arrays()* function, then the variables are given default values using the *finix_get_default_values()* function. Also note, how the FINIX error message is caught, checked and free'd for these function calls.

```
//parameter initialization and system setup

//nodalization; nnodes[0]=axial nodes, nnodes[1]=pellet radial nodes,
//nnodes[2]=cladding radial nodes
//get the default values, nnodes[]={1,20,5}.
nnodes = finix_get_default_nnodes();

//allocate memory to the data arrays
err=finix_initialize_arrays(&nnodes,&T,&r,&r_cold,&power_dist,&linear_power,
                &burnup,&params,&bcond,&sresults,&vresults);
if(err!=NULL){ finix_printf_err(err);}
finix_free_err(&err);

//fill the arrays with default values
err=finix_get_default_values(rodtype, &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);
```

The values of the parameters can also be changed by hand. To do this, the user may refer to the more detailed description of the data structures found in Appendix A. The user may also change the default boundary conditions, and the power density distribution:

```
//set boundary condition type for the cladding-coolant boundary:
// 0 = fixed & user-given cladding outer surface temperature at each axial node
// 1 = fixed & user-given heat flux from cladding to coolant at each axial node
// 2 = fixed & user-given heat transfer coefficient and bulk coolant temperature
// at each axial node
// 3 = fixed & user-given bulk coolant temperature at each axial node,
// heat transfer coefficient calculated by internal correlations
options[0]=0;

//change the cladding outer surface temperature to 500.0 K:
for(i=0;i<nnodes[0];i++){
        bcond[0][i]=500.0;
}

//set (spatially) constant power density (in units of W/m^3).
//default power distribution is uniform
//(1.0 in all pellet nodes, 0.0 in all cladding nodes)
//this can be changed by accessing the elements power_dist[i][j], if necessary
//below, set linear power of each axial node to plin:
for(i=0;i<nnodes[0];i++){
        linear_power[i]=plin;
}
```

After the system and simulation parameters have been set, the simulation can begin. However, in many cases the transient begins from a steady state, which of course has to be solved first. 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.

```
//solve initial steady state
err=finix_solve_initial_steady_state(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);
```

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, analyzed, etc. For example, to write the temperature distribution to a file, one can use the built-in function *finix_fprintf_array()*.

```
//solve transient with the given boundary conditions for several time steps
for(i=0;i<100;i++){
        for(j=0;j<nnodes[0];j++){
                linear_power[j]=plin*1.1;
        }

        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);

        finix_fprintf_array(Tfile,nnodes,(double)(i+1)*dt,T);
}
```

After the transient simulation has been done, and FINIX is no longer needed, the memory allocated for FINIX arrays has to be free'd. This can be done with the *finix_free_arrays()* function.

```
//free allocated memory to prevent memory leak.
//this should be done when FINIX is no longer needed
err=finix_free_arrays(&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);
```

8 Code assessment

8.1 General performance

The validation of FINIX-0.13.9 is presented in a separate validation report [1], where detailed results can be found. Only a brief summary is given here.

FINIX-0.13.9 has been validated 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 [14], 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 Known issues and possible caveats

The overall stability and performance of FINIX-0.13.9 has been improved since version 0.13.1. As shown in [1], the performance is very good for temperature calculations. However, a number of issues remains to solved. These are:

- The mechanical model has a discontinuous transition from the weak contact to the strong contact between the pellet and the cladding. This gives errors in the cladding stresses and strains. In some cases, oscillations of the mechanical solution can also develop. The source of these is presently not known. However, they seem to affect the overall solution in only a limited manner.

- The pressure calculated by FINIX typically differs somewhat from that calculated by FRAP-

TRAN. 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-0.13.9 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-0.13.9 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-0.13.9 has no criteria for rod failure.

- The cladding model has no plastic deformations. Because of this, cladding stresses can be overestimated, and strains underestimated in more extreme conditions.

- FINIX-0.13.9 has no models for fission gas release. In transients, this may lead to underestimation of the pressure.

- The burnup calculation of FINIX makes no difference 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 0.13.9. The new version improves the numerical stability of the solvers and implements new physical models such as the pellet relocation. Much of the changes concern adding quantities that allow initialization of FINIX for non-fresh fuel. Currently initialization can be done manually or from FRAPCON simulation. In future versions, parameterization of key burnup dependent quantities will be added, as well as capability to model steady state irradiation with FINIX.

Validation of the stand-alone FINIX 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 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, where FINIX will serve as the default fuel behavior module in the upcoming versions of Serpent 2. Initial results have been reported in Ref. [4]. In addition to Serpent, FINIX has been successfully integrated into VTT's reactor dynamics codes TRAB and TRAB-3D. Further results with TRAB, TRAB-3D and Serpent will be reported in the future.

Development of the FINIX code continues. Near-term goals are the development of a simple parameterization scheme that allows initialization of FINIX for accumulated burnup without explicit simulation of the steady state irradiation with FRAPCON and, on the other hand, improvement of FINIX's LOCA modeling capabilities. In addition, a model to handle fission gas release should be implemented. For future validation purposes, capability to simulate the burnup-dependent phenomena occurring in steady state irradiation would be beneficial. As before, user feedback and needs are seen as primary drivers of FINIX development.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

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

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

[14] 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 - Fuel behavior model and interface for multiphysics applications

Generated by Doxygen 1.8.2

Mon Sep 30 2013 10:48:30

Contents

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

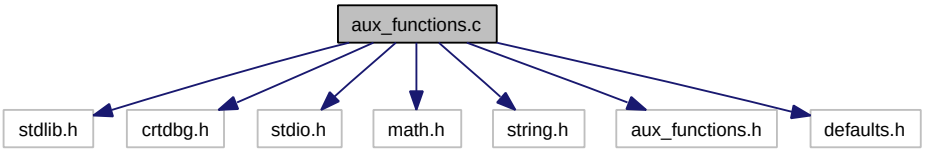
Chapter 2

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 "aux_functions.h"
#include "defaults.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_params (double ∗∗params)
- int finix_printf_array (int ∗nnodes, double ∗∗array)
- int finix_fprintf_array (FILE ∗stream, int ∗nnodes, 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, int ∗nnodes, double ∗∗r, double ∗∗power_den)

2.1.1 Macro Definition Documentation

2.1.1.1 #define _CRTDBG_MAP_ALLOC

Definition at line 13 of file aux_functions.c.

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

Definition at line 31 of file aux_functions.c.

2.1.2.2 double finix_calculate_linear_power (int *zind*, int * *nnodes*, double ** *r*, double ** *power_den*)

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

Parameters

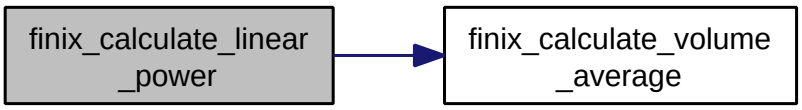
zind	axial node index
nnodes	number of nodes
r	positions of the nodes (m)
power_den	power density at each node (W/m^3)

Returns

linear power at axial node zind

Definition at line 351 of file aux_functions.c.

Here is the call graph for this function:



2.1.2.3 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

Definition at line 325 of file aux_functions.c.

2.1.2.4 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

Definition at line 89 of file aux_functions.c.

2.1.2.5 int finix_fprintf_array (FILE * *stream,* int * *nnodes,* 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
nnodes	number of nodes
time	simulation time (or other identifier of type double that distinguishes between function calls)
array	the array to be printed out

Returns

0

Definition at line 239 of file aux_functions.c.

2.1.2.6 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)

Definition at line 136 of file aux_functions.c.

2.1.2.7 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

Definition at line 270 of file aux_functions.c.

2.1.2.8 int finix_printf_array (int * *nnodes,* double ** *array*)

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

Parameters

nnodes	number of nodes
array	the array to be printed out

Returns

0

Definition at line 206 of file aux_functions.c.

2.1.2.9 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)

Definition at line 102 of file aux_functions.c.

2.1.2.10 int finix_printf_params (double ** *params*)

Prints the parameter values to stdout

Parameters

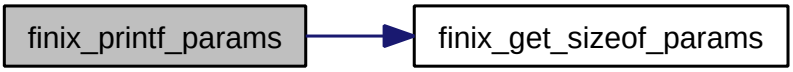
params	parameter array

Returns

0

Definition at line 171 of file aux_functions.c.

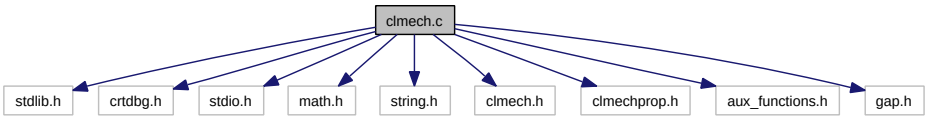
Here is the call graph for this function:



2.2 clmech.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "clmech.h"
#include "clmechprop.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, int *nnodes, double **T, double **r, double **r_cold, double **power_den, double **params, double *sresults, double **vresults, int *options)
- char ** finix_mech_solve_thin_cladding (int *nnodes, double **T, double **r, double **r_cold, double **power_den, double **params, double *sresults, double **vresults, int *options)

2.2.1 Macro Definition Documentation

2.2.1.1 #define _CRTDBG_MAP_ALLOC

Definition at line 12 of file clmech.c.

2.2.2 Function Documentation

2.2.2.1 char** finix_mech_solve_cladding_stress_strain (int *zind*, int * *nnodes*, double ** *T*, double ** *r*, double ** *r_cold*, double ** *power_den*, double ** *params*, double * *sresults*, double ** *vresults*, int * *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

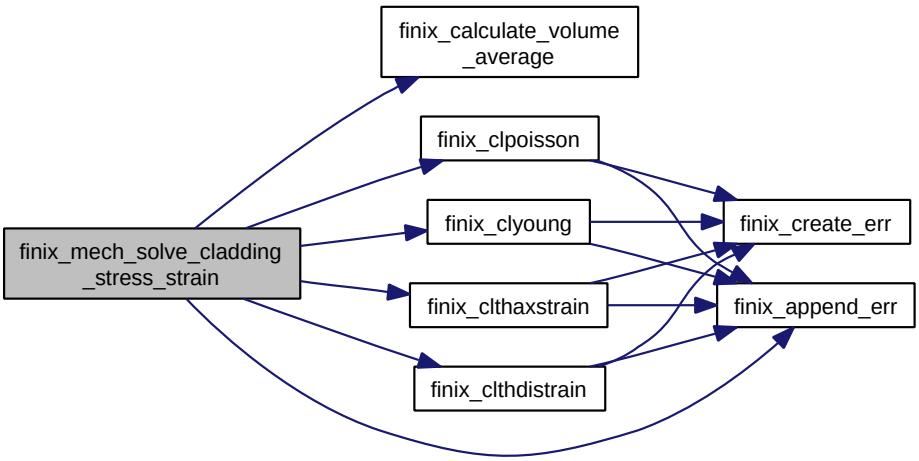
zind	axial node index
nnodes	number of nodes
T	temperature (K)
r	locations of nodes (m)
r_cold	locations of nodes in cold state (m)
power_den	power density (W/m^3)
params	system parameters (scalars)
sresults	results in scalar form
vresults	computed results (vectors)
options	model options

Returns

error string, NULL for no errors

Definition at line 43 of file clmech.c.

Here is the call graph for this function:



2.2.2.2 char** finix_mech_solve_thin_cladding (int * *nnodes*, double ** *T*, double ** *r*, double ** *r_cold*, double ** *power_den*, double ** *params*, double * *sresults*, double ** *vresults*, int * *options*)

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

Parameters

nnodes	number of nodes
T	temperature (K)
r	locations of nodes (m)
r_cold	locations of nodes in cold state (m)
power_den	power density (W/m^3)
params	system parameters (scalars)

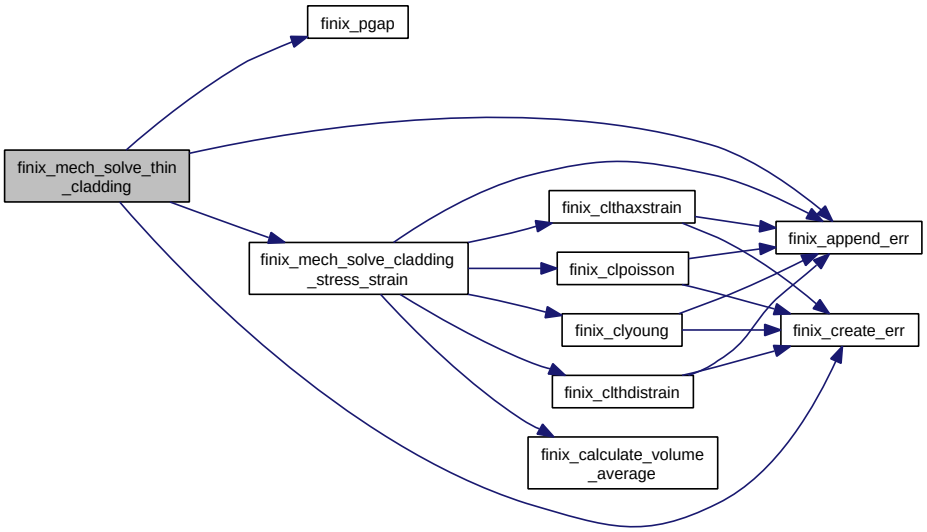
sresults	results in scalar form
vresults	computed results (vectors)
options	model options

Returns

error string, NULL for no errors

Definition at line 231 of file clmech.c.

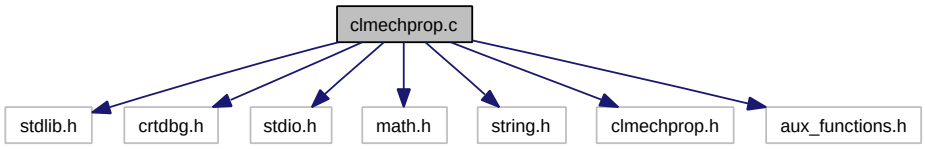
Here is the call graph for this function:



2.3 clmechprop.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.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)
- char ∗∗ finix_clpoisson (double T, double ∗nu)
- char ∗∗ finix_clyoung (double T, double oxygen_con, double coldwork, double fnf, double ∗E)
- char ∗∗ finix_clthaxstrain (double T, double ∗strain)
- char ∗∗ finix_clthdistrain (double T, double ∗strain)

2.3.1 Macro Definition Documentation

2.3.1.1 #define _CRTDBG_MAP_ALLOC

Definition at line 14 of file clmechprop.c.

2.3.2 Function Documentation

2.3.2.1 char∗∗ finix_clmeyer (double *T*, double ∗ *HM*)

Cladding Meyer's hardness

Calculates the Meyer's hardness of Zircaloy with the FRAPTRAN-1.4 correlation.

Parameters

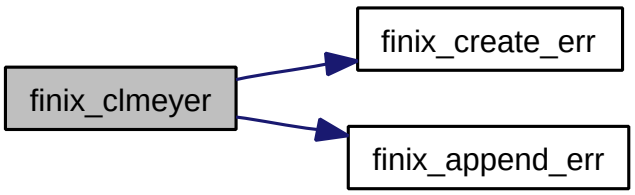
T	temperature (K)
HM	Meyer's hardness (N/m^2)

Returns

error string, NULL for no errors

Definition at line 36 of file clmechprop.c.

Here is the call graph for this function:



2.3.2.2 char∗∗ finix_clpoisson (double *T*, double ∗ *nu*)

Cladding Poisson's ratio

Calculates the Poisson's ratio of Zircaloy with the FRAPTRAN-1.4 correlation.

Parameters

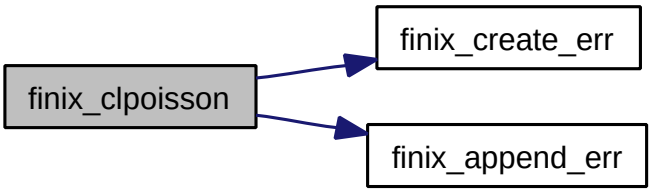
T	temperature (K)
nu	Poisson's ratio (dimensionless)

Returns

error string, NULL for no errors

Definition at line 70 of file clmechprop.c.

Here is the call graph for this function:



2.3.2.3 char** finix_clthaxstrain (double *T*, double * *strain*)

Cladding thermal axial strain

Calculates the axial thermal strain of Zircaloy with the FRAPTRAN-1.4 correlation.

Parameters

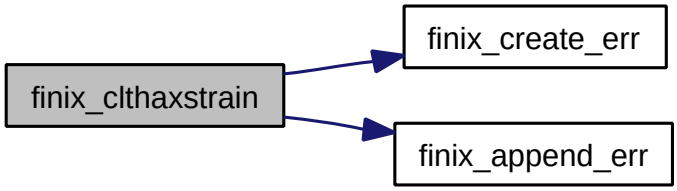
T	temperature (K)
strain	thermal axial strain

Returns

error string, NULL for no errors

Definition at line 144 of file clmechprop.c.

Here is the call graph for this function:



2.3.2.4 char** finix_clthdistrain (double *T*, double * *strain*)

Cladding thermal diametral strain

Calculates the diametral thermal strain of Zircaloy with the FRAPTRAN-1.4 correlation.

Parameters

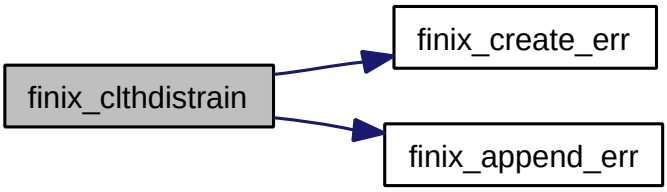
T	temperature (K)
strain	thermal axial strain

Returns

error string, NULL for no errors

Definition at line 180 of file clmechprop.c.

Here is the call graph for this function:



2.3.2.5 char** finix_clyoung (double *T*, double *oxygen_con*, double *coldwork*, double *fnf*, double * *E*)

Cladding Young's modulus

Calculates the Young's modulus of Zircaloy with the FRAPTRAN-1.4 correlation.

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)

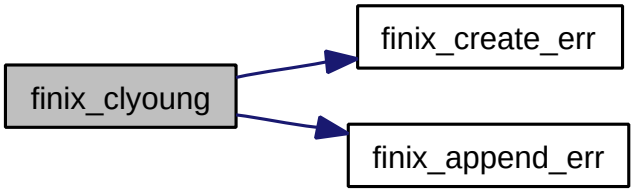
Returns

error string, NULL for no errors

assume as-received oxygen concentration of 0.0012.

Definition at line 101 of file clmechprop.c.

Here is the call graph for this function:

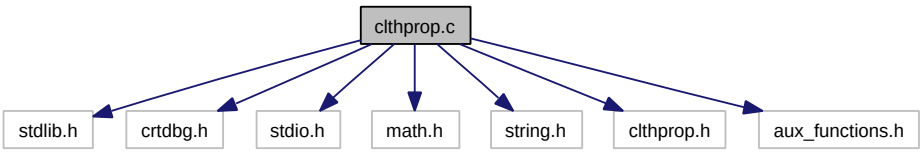


2.4 clthprop.c File Reference

```
#include <stdlib.h>
```

```
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.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)
- char ∗∗ finix_clcp (double T, double ∗cp)

2.4.1 Macro Definition Documentation

2.4.1.1 #define _CRTDBG_MAP_ALLOC

Definition at line 14 of file clthprop.c.

2.4.2 Function Documentation

2.4.2.1 char∗∗ finix_clcp (double *T*, double ∗ *cp*)

Cladding specific heat

Calculates the specific heat of Zircaloy with the FRAPTRAN-1.4 correlation.

Parameters

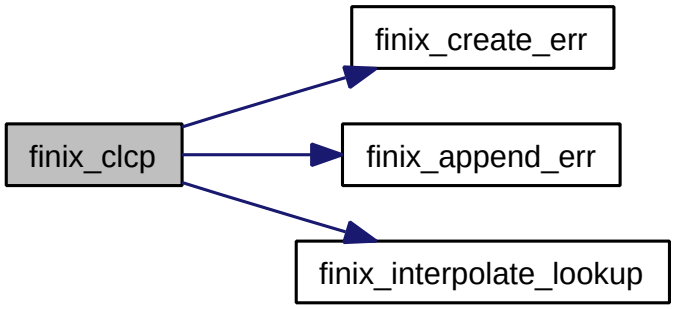
T	temperature (K)
cp	specific heat

Returns

error string, NULL for no errors

Definition at line 70 of file clthprop.c.

Here is the call graph for this function:



2.4.2.2 char** finix_clthcond (double *T*, double * *lambda*)

Cladding thermal conductivity

Calculates the thermal conductivity of Zircaloy with the FRAPTRAN-1.4 correlation.

Parameters

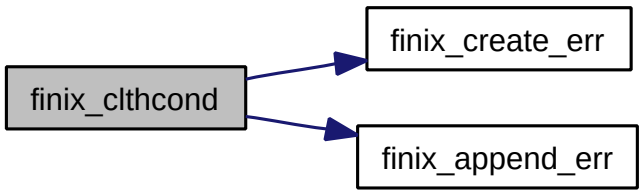
T	temperature (K)
lambda	thermal conductivity (W/mK)

Returns

error string, NULL for no errors

Definition at line 36 of file clthprop.c.

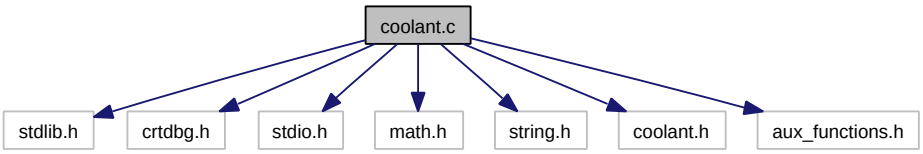
Here is the call graph for this function:



2.5 coolant.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.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, int ∗nnodes, double ∗∗r, double ∗∗bcond, double ∗∗params)
- char ∗∗ finix_update_bcond (int zind, int ∗nnodes, double ∗∗T, double ∗∗r, double ∗∗bcond, double ∗∗params, int ∗options)

2.5.1 Macro Definition Documentation

2.5.1.1 #define _CRTDBG_MAP_ALLOC

Definition at line 12 of file coolant.c.

2.5.2 Function Documentation

2.5.2.1 char∗∗ finix_hcoolant (int *zind*, int ∗ *nnodes*, double ∗∗ *r*, double ∗∗ *bcond*, double ∗∗ *params*)

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

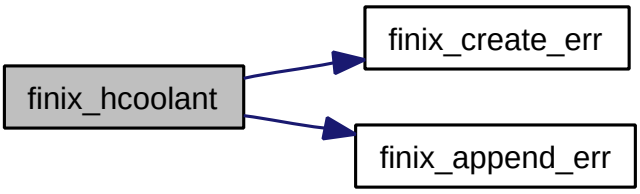
zind	index of the axial node
nnodes	numbers of nodes
r	node radial positions (m)
bcond	boundary conditions, including the temperatures of the cladding and the coolant (must be given), and the heat transfer coefficient (function output)
params	system parameters, including the coolant conditions and thermal hydraulic parameters at params[4]

Returns

error string, NULL for no errors

Definition at line 34 of file coolant.c.

Here is the call graph for this function:



2.5.2.2 char** finix_update_bcond (int *zind*, int * *nnodes*, double ** *T*, double ** *r*, double ** *bcond*, double ** *params*, int * *options*)

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
nnodes	numbers of nodes
T	temperature distribution (K)
r	node radial positions (m)
bcond	boundary conditions, including the temperatures of the cladding and the coolant (must be given), and the heat transfer coefficient (function output)
params	system parameters, including the coolant conditions and thermal hydraulic parameters at params[4]
options	model options. The boundary conditions are selected in options[0]

Returns

error string, NULL for no errors

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

If options[0]==1, use user-given heat flux between rod outer surface and coolant

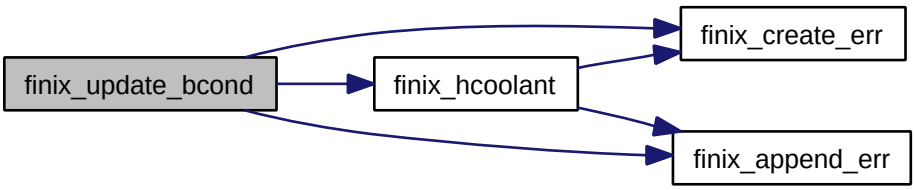
If options[0]==2, use user-given heat transfer coefficient and coolant bulk temperature as boundary condition

If options[0]==3, use user-given bulk temperature and calculate heat transfer coefficient from internal correlations (needs inlet mass flux)

If options[0]==4, use user-given inlet temperature and mass flux to calculate heat transfer coefficient and coolant bulk temperature downstream (NOT IMPLEMENTED)

Definition at line 178 of file coolant.c.

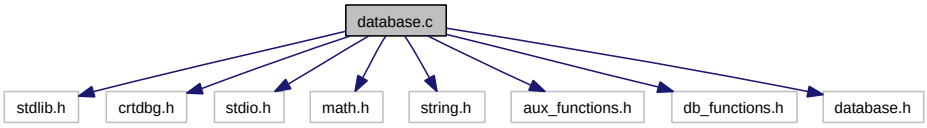
Here is the call graph for this function:



2.6 database.c File Reference

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

Include dependency graph for database.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- char ** finix_get_default_params (int rodtype, double **params, double *sresults)
- char ** finix_db_fetch_data (int rodtype, double **axialPdistr, double ***Phist, double ***Tcoolhist, int *hist_dims, double **params, double **axialTbcond, double **axialPlin, double *endtime)
- char ** finix_db_initialize_database (int rodtype, double **params, int *nnodes, int *options, double ***axialPdistr, double ****Phist, double ****Tcoolhist, int **hist_dims, double ***axialTbcond, double ***axialPlin, double **time_dt)
- char ** finix_db_free_data_arrays (double ***axialPdistr, double ****Phist, double ****Tcoolhist, int **hist_dims, double ***axialTbcond, double ***axialPlin, double **time_dt)
- char ** finix_db_timestep_division (int rodtype, double time, double *dt, int *pp)
- char ** finix_db_fetch_ss_data (int rodtype, double **params, double ***Phist, double ***Tcoolhist, double **axialPlin, double **axialTbcond, double **axialPdistr, double *time_dt, int *nnodes, int *hist_dims)

2.6.1 Macro Definition Documentation

2.6.1.1 #define _CRTDBG_MAP_ALLOC

Definition at line 15 of file database.c.

2.6.2 Function Documentation

2.6.2.1 char** finix_db_fetch_data (int *rodtype*, double ** *axialPdistr*, double *** *Phist*, double *** *Tcoolhist*, int * *hist_dims*, double ** *params*, double ** *axialTbcond*, double ** *axialPlin*, double * *endtime*)

Database for power histories, axial power distributions, coolant temperature histories.

The indexing of the data is the same as the rod data indexing in finix_get_default_params().

Parameters

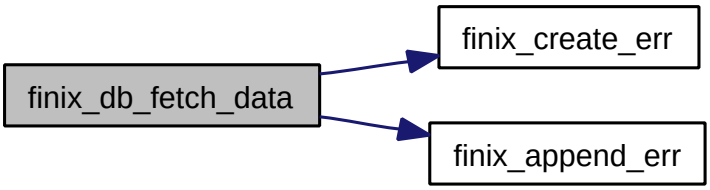
rodtype	rod type index
hist_dims	rod history dimensions
params	miscellaneous system parameters in scalar form
endtime	simulation end time
axialPdistr	axial power distribution, axialPdistr[0] = axial elevation, axialPdistr[1] = axial power factor
Phist	linear power history, Phist[i][0] = time, zone i, Phist[i][1] = linear power in W/m, zone i
Tcoolhist	coolant temperature history, Tcoolhist[i][0] = time, zone i, Tcoolhist[i][1] = coolant temperature, zone i
axialTbcond	axial coolant temperature: axialTbcond[0] = axial elevation, axialTbcond[1] = coolant temperature at this elevation
axialPlin	axial linear power

Returns

error string, NULL for no errors

Definition at line 1217 of file database.c.

Here is the call graph for this function:



2.6.2.2 char** finix_db_fetch_ss_data (int *rodtype*, double ** *params*, double *** *Phist*, double *** *Tcoolhist*, double ** *axialPlin*, double ** *axialTbcond*, double ** *axialPdistr*, double * *time_dt*, int * *nnodes*, int * *hist_dims*)

Database for steady state power histories and coolant temperature histories. Also contains elevation data for axial multi-zone coolant temperature and linear power histories.

The indexing of the data is the same as the rod data indexing in finix_get_default_params().

Parameters

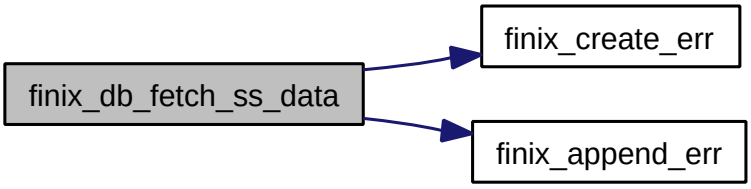
rodtype	rod type index
params	miscellaneous system parameters in scalar form
nnodes	numbers of nodes
hist_dims	rod history data dimensions
time_dt	time interval data
axialPdistr	axial power distribution, axialPdistr[0] = axial elevation, axialPdistr[1] = axial power factor
Phist	linear power history, Phist[0] = time, Phist[1] = linear power in W/m
Tcoolhist	coolant temperature history, Tcoolhist[i][0] = time, zone i, Tcoolhist[i][1] = coolant temperature, zone i
axialTbcond	axial coolant temperature: axialTbcond[0] = axial elevation, axialTbcond[1] = coolant temperature at this elevation
axialPlin	axial linear power

Returns

error string, NULL for no errors

Definition at line 3383 of file database.c.

Here is the call graph for this function:



2.6.2.3 char** finix_db_free_data_arrays (double *** *axialPdistr,* double **** *Phist,* double **** *Tcoolhist,* int ** *hist_dims,* double *** *axialTbcond,* double *** *axialPlin,* double ** *time_dt*)

Frees the memory allocated for data arrays and sets the pointers to NULL.

Parameters

time_dt	time interval data
axialPdistr	axial power distribution, axialPdistr[0] = axial elevation, axialPdistr[1] = axial power factor
Phist	linear power history, Phist[0] = time, Phist[1] = linear power in W/m
Tcoolhist	coolant temperature history, Tcoolhist[i][0] = time, zone i, Tcoolhist[i][1] = coolant temperature, zone i
hist_dims	history dimensions: hist_dims[0] = zones in coolant temperature history, hist_dims[1] = timesteps in coolant temperature history, hist_dims[2] = number of elevations of axial power factors, hist_dims[3] = timesteps in linear power history, hist_dims[4] = zones in linear power history
axialTbcond	axial coolant temperature: axialTbcond[0] = axial elevation, axialTbcond[1] = coolant temperature at this elevation
axialPlin	axial linear power

Definition at line 3032 of file database.c.

2.6.2.4 char** finix_db_initialize_database (int *rodtype,* double ** *params,* int * *nnodes,* int * *options,* double *** *axialPdistr,* double **** *Phist,* double **** *Tcoolhist,* int ** *hist_dims,* double *** *axialTbcond,* double *** *axialPlin,* double ** *time_dt*)

Allocates memory for rod history data arrays. Also nnodes[] is updated to match the restart file for each scenario.

Parameters

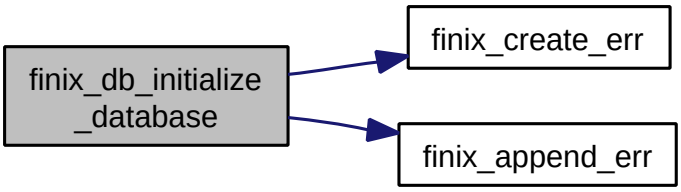
rodtype	rod type index
params	miscellaneous system parameters in scalar form
nnodes	numbers of nodes
options	simulation options
time_dt	time interval data
axialPdistr	axial power distribution, axialPdistr[0] = axial elevation, axialPdistr[1] = axial power factor
Phist	linear power history, Phist[0] = time, Phist[1] = linear power in W/m
Tcoolhist	coolant temperature history, Tcoolhist[i][0] = time, zone i, Tcoolhist[i][1] = coolant temperature, zone i
hist_dims	history dimensions: hist_dims[0] = zones in coolant temperature history, hist_dims[1] = timesteps in coolant temperature history, hist_dims[2] = number of elevations of axial power factors, hist_dims[3] = timesteps in linear power history, hist_dims[4] = zones in linear power history
axialTbcond	axial coolant temperature: axialTbcond[0] = axial elevation, axialTbcond[1] = coolant temperature at this elevation
axialPlin	axial linear power

Returns

error string, NULL for no errors

Definition at line 2627 of file database.c.

Here is the call graph for this function:



2.6.2.5 char** finix_db_timestep_division (int *rodtype*, double *time*, double * *dt*, int * *pp*)

Divides time steps for each rodtype. Rodtype indexing as in defaults.c.

Parameters

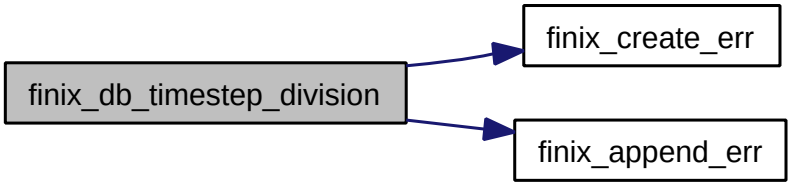
rodtype	rod type index
time	time step
dt,:	time step
pp,:	printing parameter

Returns

error string, NULL for no errors

Definition at line 3093 of file database.c.

Here is the call graph for this function:



2.6.2.6 char** finix_get_default_params (int *rodtype*, double ** *params*, double * *sresults*)

Gives default parametes by rodtype.

Parameters

rodtype	the type of the rod and plant. Available types are: 0 (TMI-1 PWR)
params	the parameter data array. The values of the array elements are changed to the default values for given rod type
sresults	scalar results, initialized from params

Returns

error string, NULL for no errors

Parameter explanations:

params[0]: rod dimensions

params[0][0]: pellet inner radius, as-fabricated (m)

params[0][1]: pellet outer radius, as-fabricated (m)

params[0][2]: cladding inner radius, as-fabricated (m)

params[0][3]: cladding outer radius, as-fabricated (m)

params[0][4]: total active axial length of the fuel in cold state (m)

params[0][5]: total active axial length of the cladding in cold state (m); should the the same as params[0][4]

params[0][6]: plenum length in cold state (m)

params[1]: pellet properties

params[1][0]: surface roughness (m)

params[1][1]: density of the pellet as fraction from theoretical (dimensionless)

params[1][2]: gadolinia weight fraction (dimensionless)

params[2]: cladding properties

params[2][0]: surface roughness (m)

params[2][1]: cold work (dimensionless)

params[2][2]: oxygen concentration (dimensionless)

params[2][3]: fast neutron fluence (n/m^2)

params[3]: gas properties

params[3][0]: fill pressure at cold state (at 300 K) (Pa)

params[3][1]: initial plenum temperature (K)

params[3][2]: helium fraction at cold state (dimensionless).

params[3][3]: argon fraction at cold state (dimensionless).

params[3][4]: krypton fraction at cold state (dimensionless).

params[3][5]: xenon fraction at cold state (dimensionless).

params[3][6]: hydrogen fraction at cold state (dimensionless).

params[3][7]: nitrogen fraction at cold state (dimensionless).

params[3][8]: water vapor fraction at cold state (dimensionless).

params[4]: coolant and thermal hydraulic properties

params[4][0]: coolant pressure (Pa)

params[4][1]: coolant inlet temperature (K)

params[4][2]: coolant mass flux (kg/sm^2)

params[4][3]: rod pitch (m)

params[4][4]: number of rods in one unit cell (1.0 for square lattice, 0.5 for triangular lattice)

sresults[0]: total active axial length of the fuel (m)

sresults[1]: total active axial length of the cladding (m)

sresults[2]: plenum length (m)

sresults[3]: plenum temperature (K)

sresults[4]: gas pressure (Pa)

sresults[5]: gas amount (moles)

sresults[6]: He fraction of the fill gas (dimensionless)

sresults[7]: Ar fraction of the fill gas (dimensionless)

sresults[8]: Kr fraction of the fill gas (dimensionless)

sresults[9]: Xe fraction of the fill gas (dimensionless)

sresults[10]: H2 fraction of the fill gas (dimensionless)

sresults[11]: N2 fraction of the fill gas (dimensionless)

sresults[12]: H2O fraction of the fill gas (dimensionless)

Supported rod types:

0: TMI-1 PWR

11: IFA-429 rod BC

12: IFA-429 rod AB

13: IFA-429 rod AD

14: IFA-429 rod AH

101: CABRI REP-Na1 rod

102: CABRI REP-Na3 rod

103: CABRI REP-Na4 rod

104: CABRI REP-Na8 rod

105: NSRR FK-1 rod

106: NSRR HBO-1 rod

107: NSRR HBO-5 rod

108: NSRR HBO-6 rod

109: NSRR TS-5 rod

110: NSRR VA-1 rod

111: NSRR VA-3 rod

112: BIGR RT-4 rod

113: BIGR RT-8 rod

114: BIGR RT-10 rod

115: BIGR RT-12 rod

120: Halden IFA-432 experiment rod 1

121: Halden IFA-432 experiment rod 2

122: Halden IFA-432 experiment rod 3

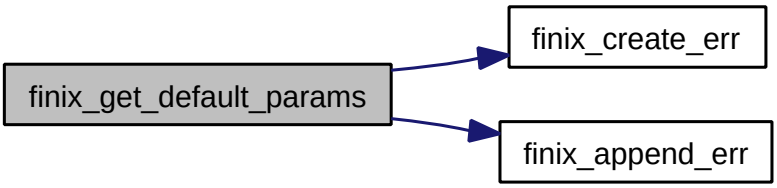
123: Halden IFA-432 experiment rod 5

124: Halden IFA-432 experiment rod 6

125: Halden IFA-507 experiment rod

Definition at line 39 of file database.c.

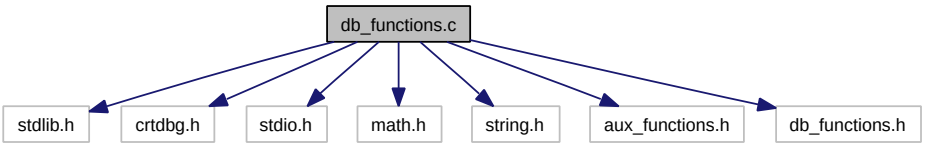
Here is the call graph for this function:



2.7 db_functions.c File Reference

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

Include dependency graph for db_functions.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- char ** finix_db_set_T_bconds (double time, int *nnodes, double **bcond, double **params, double ***Tcoolhist, int *hist_dims, double **axialTbcond, int *options)
- char ** finix_db_calculate_linear_power (double time, double **axialPdistr, double **axialPlin, double ***Phist, int *nnodes, double **params, double **r, double **burnup, double *linear_power, double **power_dist, int *hist_dims)
- char ** finix_db_calculate_axialPlin (double time, double ***Phist, double **axialPlin, int *hist_dims)
- char ** finix_db_calculate_axialTbcond (double time, double ***Tcoolhist, double **axialTbcond, int *hist_dims)
- char ** finix_db_fprintf_stripfile (int printoption, FILE *writefile, double time, double avp, double bu, double *sresults, double **params, int *nnodes, double **vresults, double **r, double **T, double **bcond, double *linear_power)

2.7.1 Macro Definition Documentation

2.7.1.1 #define _CRTDBG_MAP_ALLOC

Definition at line 13 of file db_functions.c.

2.7.2 Function Documentation

2.7.2.1 char** finix_db_calculate_axialPlin (double *time*, double *** *Phist*, double ** *axialPlin*, int * *hist_dims*)

Interpolates linear power from multiple axial power history zones for the given time step. Outputs the linear power values to axialPlin to corresponding elevations.

Parameters

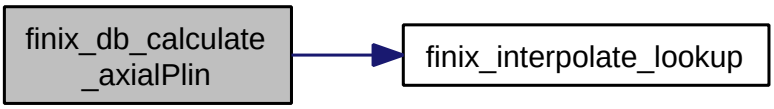
time	time step
Phist	linear power history
axialPlin	axial linear power distribution array
hist_dims	rod history dimensions

Returns

error string, NULL for no errors

Definition at line 171 of file db_functions.c.

Here is the call graph for this function:



2.7.2.2 char** finix_db_calculate_axialTbcond (double *time*, double *** *Tcoolhist*, double ** *axialTbcond*, int * *hist_dims*)

Interpolates coolant temperature from multiple axial coolant temperature zones for the given time step. Outputs the linear power values to axialPlin to corresponding elevations.

Parameters

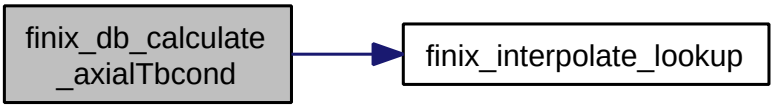
time	time step
Tcoolhist	coolant temperature history
axialTbcond	axial temperature boundary condition array
hist_dims	rod history dimensions

Returns

error string, NULL for no errors

Definition at line 222 of file db_functions.c.

Here is the call graph for this function:



2.7.2.3 char** finix_db_calculate_linear_power (double *time*, double ** *axialPdistr*, double ** *axialPlin*, double *** *Phist*, int * *nnodes*, double ** *params*, double ** *r*, double ** *burnup*, double * *linear_power*, double ** *power_dist*, int * *hist_dims*)

Calculates linear power for each axial node from one or multiple axial linear power history zones. Interpolates between zone center elevations, assumes constant linear power for rod ends from upmost zone center or other given axial elevation in axialPlin.

Parameters

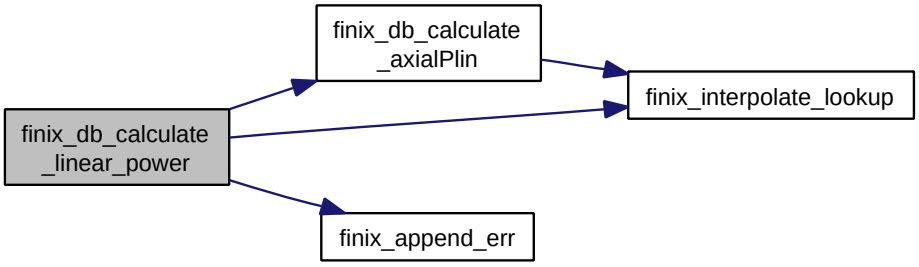
time	time step
axialPdistr	axial power distribution (dimensionless)
axialPlin	axial linear power distribution (kW/m)
Phist	linear power history
nnodes	numbers of nodes
params	miscellaneous system parameters
r	rod radii
burnup	burnup array
linear_power	linear power array
power_dist	radial power distribution (dimensionless)
hist_dims	rod history dimensions

Returns

error string, NULL for no errors

Definition at line 116 of file db_functions.c.

Here is the call graph for this function:



2.7.2.4 char** finix_db_fprintf_stripfile (int *printoption*, FILE * *writefile*, double *time*, double *avp*, double *bu*, double * *sresults*, double ** *params*, int * *nnodes*, double ** *vresults*, double ** *r*, double ** *T*, double ** *bcond*, double * *linear_power*)

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
avp	rod-average linear power
bu	burnup at the current time step
sresults	results calculated by FINIX in scalar form
params	miscellaneous system parameters
nnodes	numbers of nodes

vresults	results calculated by FINIX in vector form
r	rod radii
T	rod temperatures
bcond	boundary conditions
linear_power	linear power array

Returns

always returns NULL

Definition at line 262 of file db_functions.c.

2.7.2.5 char** finix_db_set_T_bconds (double *time*, int * *nnodes*, double ** *bcond*, double ** *params*, double *** *Tcoolhist*, int * *hist_dims*, double ** *axialTbcond*, int * *options*)

Calculates coolant temperature boundary conditions for each axial node at the given time step. Interpolates between zone center elevations, assumes constant temperature for rod ends from upmost zone center or other given axial elevation in axialTbcond.

Parameters

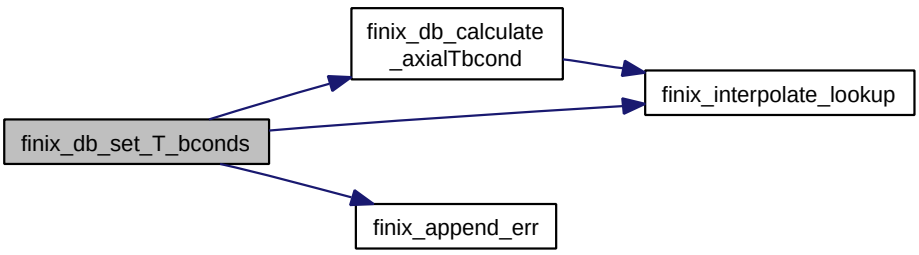
time	time step
nnodes	numbers of nodes
bcond	boundary conditions
params	miscellaneous system parameters
Tcoolhist	coolant temperature history
hist_dims	rod history dimensions
axialTbcond	axial temperature boundary condition array
options	system options

Returns

error string, NULL for no errors

Definition at line 40 of file db_functions.c.

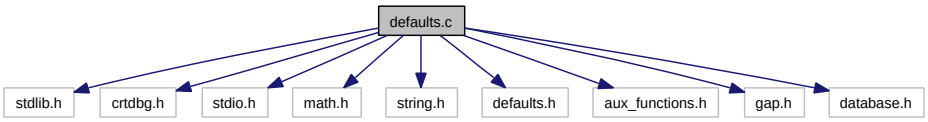
Here is the call graph for this function:



2.8 defaults.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "defaults.h"
#include "aux_functions.h"
#include "gap.h"
#include "database.h"
```

Include dependency graph for defaults.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- int * finix_get_default_nnodes ()
- int * finix_get_max_nnodes ()
- int * finix_get_sizeof_params ()
- int finix_get_sizeof_options ()
- int finix_get_sizeof_sresults ()
- int finix_get_sizeof_vresults ()
- char ** finix_initialize_arrays (int **nnodes, double ***T, double ***r, double ***r_cold, double ***power_dist, double **linear_power, double ***burnup, double ***params, double ***bcond, double **sresults, double ***vresults)
- char ** finix_get_default_values (int rodtype, int **nnodes, double **T, double **r, double **r_cold, double **power_dist, double *linear_power, double **burnup, double **params, double **bcond, double *sresults, double **vresults, int **options)
- char ** finix_get_default_positions (int **nnodes, double **r, double **r_cold, double **params)
- char ** finix_get_default_cold_state (int **nnodes, double **T)
- char ** finix_get_default_burnup (int **nnodes, double **burnup)
- char ** finix_get_default_power (int **nnodes, double **power_dist, double *linear_power)
- char ** finix_get_default_bcond (int **nnodes, double **params, double **bcond)
- int * finix_get_default_options ()
- char ** finix_set_constant_power_dist (int **nnodes, double **power_dist)
- char ** finix_free_arrays (int **nnodes, double ***T, double ***r, double ***r_cold, double ***power_dist, double **linear_power, double ***burnup, double ***params, double ***bcond, double **sresults, double ***vresults, int **options)

2.8.1 Macro Definition Documentation

2.8.1.1 #define _CRTDBG_MAP_ALLOC

Definition at line 15 of file defaults.c.

2.8.2 Function Documentation

2.8.2.1 char** finix_free_arrays (int ** *nnodes*, double *** *T*, double *** *r*, double *** *r_cold*, double *** *power_dist*, double ** *linear_power*, double *** *burnup*, double *** *params*, double *** *bcond*, double ** *sresults*, double *** *vresults*, int ** *options*)

Frees the allocated memory of the data arrays used by FINIX, including the nnodes and options arrays.

Parameters

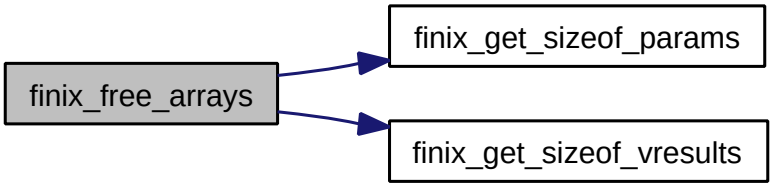
nnodes	array storing numbers of nodes; nnodes[0] = axial, nnodes[1] = pellet radial, nnodes[2] = cladding radial nodes.
T	temperature at each node. The nodes are indexed as T[i][j], where i is the axial index and j is the radial index.
r	node positions
r_cold	node positions at cold state (zero strain)
power_dist	power distribution at each node
linear_power	linear power at each axial node
burnup	burn-up at each node
params	miscellaneous parameters:
bcond	boundary conditions:
sresults	scalar results
vresults	various data computed by the model
options	

Returns

error string, NULL for no errors

Definition at line 718 of file defaults.c.

Here is the call graph for this function:



2.8.2.2 char** finix_get_default_bcond (int ** *nnodes*, double ** *params*, double ** *bcond*)

Gives default boundary conditions (rod outer surface T equal to coolant T given in params).

Parameters

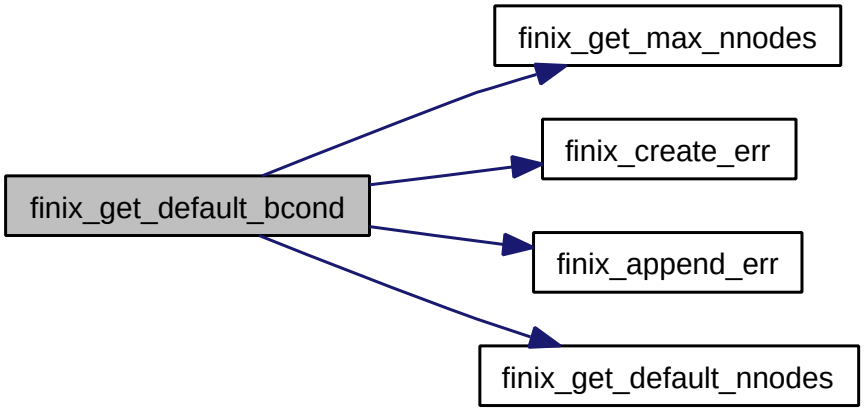
nnodes	numbers of nodes
params	system parameters
bcond	boundary conditions array.

Returns

error string, NULL for no errors

Definition at line 587 of file defaults.c.

Here is the call graph for this function:



2.8.2.3 char** finix_get_default_burnup (int ** *nnodes*, double ** *burnup*)

Gives the fresh fuel burn-up of zero in all nodes.

Parameters

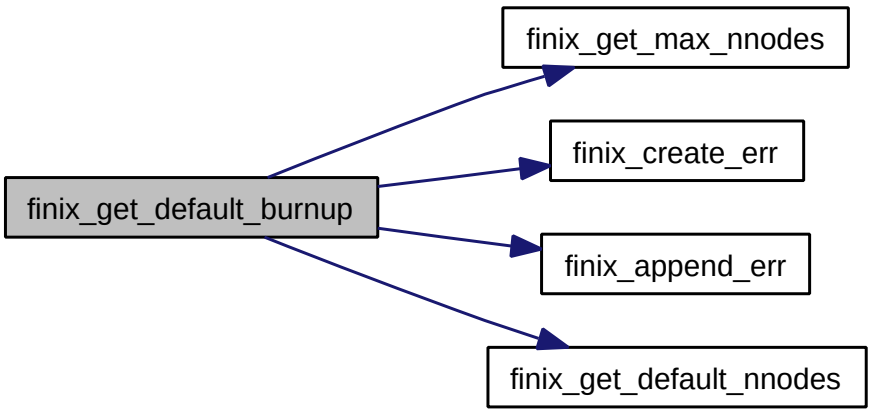
nnodes	numbers of nodes
burnup	burn-up distribution

Returns

error string, NULL for no errors

Definition at line 479 of file defaults.c.

Here is the call graph for this function:



2.8.2.4 char** finix_get_default_cold_state (int ** *nnodes*, double ** *T*)

Gives the cold state temperature T = 300 K in all nodes.

Parameters

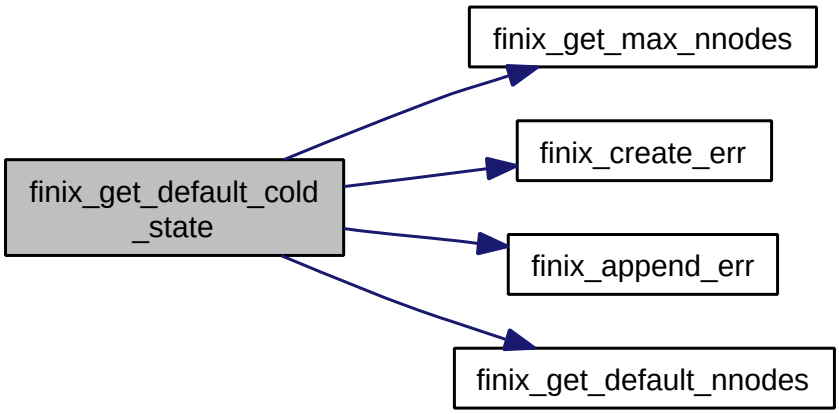
nnodes	numbers of nodes
T	temperature distribution

Returns

error string, NULL for no errors

Definition at line 429 of file defaults.c.

Here is the call graph for this function:



2.8.2.5 int* finix_get_default_nnodes ( )

Returns the default nodalization of the rod.

Returns

array storing numbers of nodes; nnodes[0] = axial, nnodes[1] = pellet radial, nnodes[2] = cladding radial nodes.

Definition at line 33 of file defaults.c.

2.8.2.6 int* finix_get_default_options ( )

Gives the default options.

Returns

default options: fixed cladding outer surface temperature as boundary conditions, temperature iteration ON, gap contact heat transfer model ON, cladding elasticity model ON plenum model ON, FGR model OFF, FRAPCON pellet relocation model

Definition at line 643 of file defaults.c.

Here is the call graph for this function:

2.8.2.7 char** finix_get_default_positions (int ** *nnodes*, double ** *r*, double ** *r_cold*, double ** *params*)

Gives the default discretization for the position vectors r and r_cold, with the given number of nodes.

Parameters

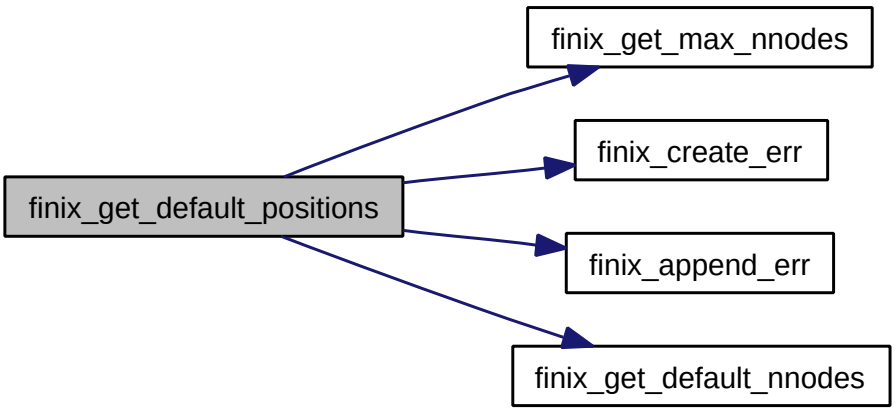
nnodes	array containing the number of radial and axial nodes; must be given as input
r	the node radial coordinates; the array must be initialized
r_cold	the node radial coordinates in the cold state; must be initialized
params	the system parameters, including the rod dimensions at params[0]; must be given as input

Returns

error string, NULL for no errors

Definition at line 333 of file defaults.c.

Here is the call graph for this function:



2.8.2.8 char** finix_get_default_power (int ** *nnodes*, double ** *power_dist*, double * *linear_power*)

Gives power density of zero in all nodes.

Parameters

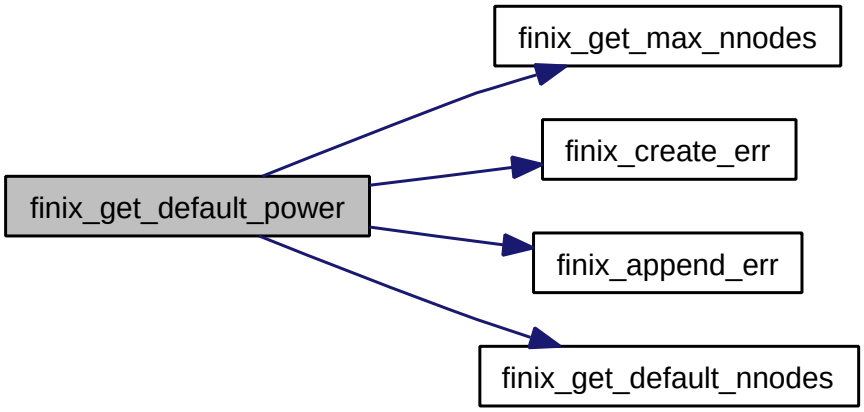
nnodes	numbers of nodes
power_dist	power density distribution
linear_power	linear power at each axial node

Returns

error string, NULL for no errors

Definition at line 530 of file defaults.c.

Here is the call graph for this function:



2.8.2.9 char** finix_get_default_values (int *rodtype*, int ** *nnodes*, double ** *T*, double ** *r*, double ** *r_cold*, double ** *power_dist*, double * *linear_power*, double ** *burnup*, double ** *params*, double ** *bcond*, double * *sresults*, double ** *vresults*, int ** *options*)

Gives default values for all FINIX data arrays. Calls the other finix_get_default_ functions. See the description of the respective functions for details.

Parameters

rodtype	the type of the rod and plant.
nnodes	array storing numbers of nodes; nnodes[0] = axial, nnodes[1] = pellet radial, nnodes[2] = cladding radial nodes.
T	temperature at each node. The nodes are indexed as T[i][j], where i is the axial index and j is the radial index.
r	node positions
r_cold	node positions at cold state (zero strain)
power_dist	power distribution at each node
linear_power	linear power at each axial node
burnup	burn-up at each node
params	miscellaneous parameters: params[0] = rod dimensions, params[1] = pellet properties, params[2] = cladding properties, params[3] = gas gap properties (including current pressure at sresults[4]), params[4] = coolant properties
bcond	boundary conditions: bcond[0] = cladding outer T, bcond[1] = coolant T, bcond[2] = heat transfer coeffienct from cladding, bcond[3] = heat flux from coolant; the used boundary condition is defined in options (see below)
sresults	scalar results
vresults	various data computed by the model, each for every axial node: vresults[0] = gap heat transfer coefficient, vresults[1] = cladding radial strain, vresults[2] = cladding hoop strain, vresults[3] = cladding axial strain, vresults[4] = pellet axial strain, vresults[5] = pellet-cladding contact pressure, vresults[6] = cladding hoop stress, vresults[7] = cladding axial stress, vresults[8] = pellet axial strain with previous open gap, vresults[9] = cladding axial strain with previous open gap vresults[10] = fuel swelling strain, vresults[11] = fuel densification strain, vresults[12] = fuel hard relocation strain, vresults[13] = fuel soft relocation strain, vresults[14] = clad plastic hoop strain, vresults[15] = clad plastic axial strain, vresults[13] = clad plastic radial strain

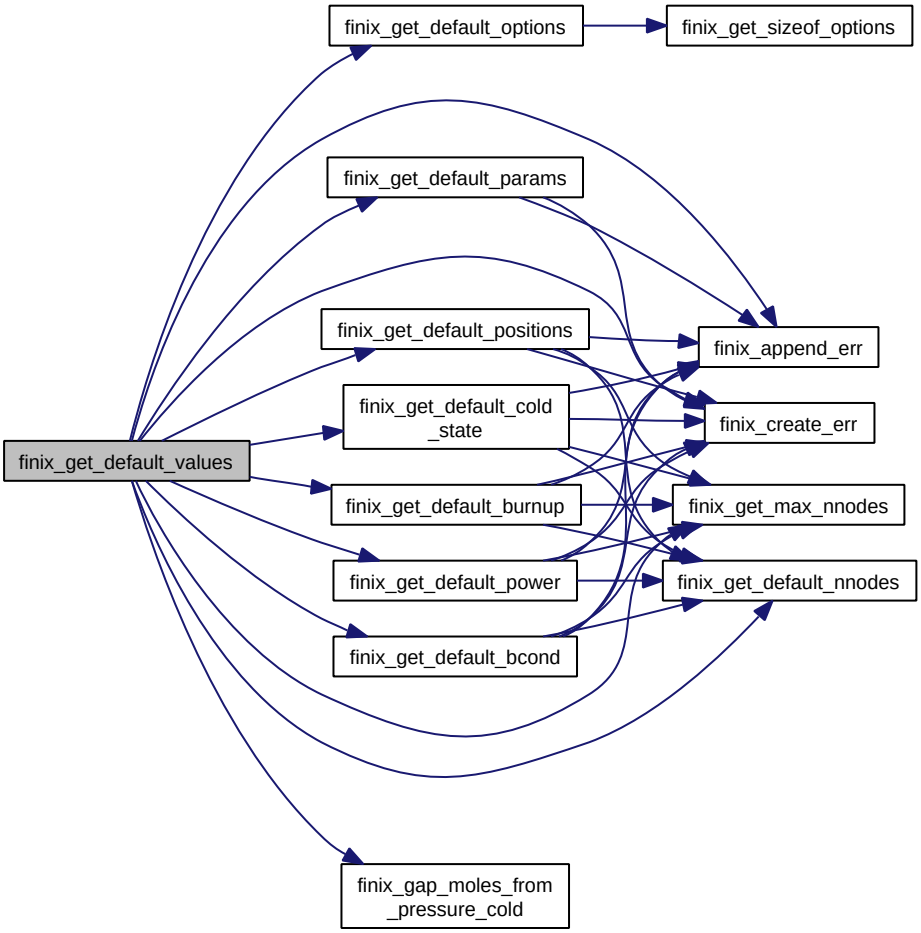
options	various model options: options[0] = boundary conditions, options[1] = temperature iteration, options[2] = gap conductance model: 1 for FRAPCON (default) 2 for FRAPTRAN, negative value switches off contact conductance; options[3] = cladding elasticity model, options[4] = plenum model, options[5] = FGR model; options[0] can take the values 0 (use fixed cladding outer T as boundary condition), 1 (use fixed heat flux), 2 (use fixed heat transfer coefficient and temperatures), 3 (use Dittus-Boelter model and coolant parameters). options[1-5] can be used to switch the default model off (0) or on (1). More models can be added in later versions.

Returns

error string, NULL for no errors

Definition at line 258 of file defaults.c.

Here is the call graph for this function:



2.8.2.10 int* finix_get_max_nnodes ( )

Returns the maximum nodalization of the rod. Limits are enforced to guard against uninitialized values, and in that sense are arbitrary.

Returns

array storing numbers of nodes; nnodes[0] = axial, nnodes[1] = pellet radial, nnodes[2] = cladding radial nodes.

Definition at line 47 of file defaults.c.

2.8.2.11 int finix_get_sizeof_options ( )

Gives the number of options

Returns

the number of elements that should be allocated in the options array

Definition at line 79 of file defaults.c.

2.8.2.12 int* finix_get_sizeof_params ( )

Gives the number of elements in the params array that should be allocated in the data structures.

Returns

the size of the params array. First number is the number of lists, the following numbers are the lengths of each list.

Definition at line 60 of file defaults.c.

2.8.2.13 int finix_get_sizeof_sresults ( )

Gives the number of different observables stored in the sresults array

Returns

the number of pointers that should be allocated in the sresults array. Each pointer then points to the array of size nz (number of axial nodes).

Definition at line 89 of file defaults.c.

2.8.2.14 int finix_get_sizeof_vresults ( )

Gives the number of different observables stored in the vresults array

Returns

the number of pointers that should be allocated in the vresults array. Each pointer then points to the array of size nz (number of axial nodes).

Definition at line 98 of file defaults.c.

2.8.2.15 char** finix_initialize_arrays (int ** *nnodes,* double *** *T,* double *** *r,* double *** *r_cold,* double *** *power_dist,* double ** *linear_power,* double *** *burnup,* double *** *params,* double *** *bcond,* double ** *sresults,* double *** *vresults*)

Initializes data arrays used by FINIX. Takes nnodes as input and uses its values to initialize (allocate memory) the rest of the arrays. Does NOT free previously allocated memory.

Parameters

nnodes	array storing numbers of nodes; nnodes[0] = axial, nnodes[1] = pellet radial, nnodes[2] = cladding radial nodes.
T	temperature at each node. The nodes are indexed as T[i][j], where i is the axial index and j is the radial index.

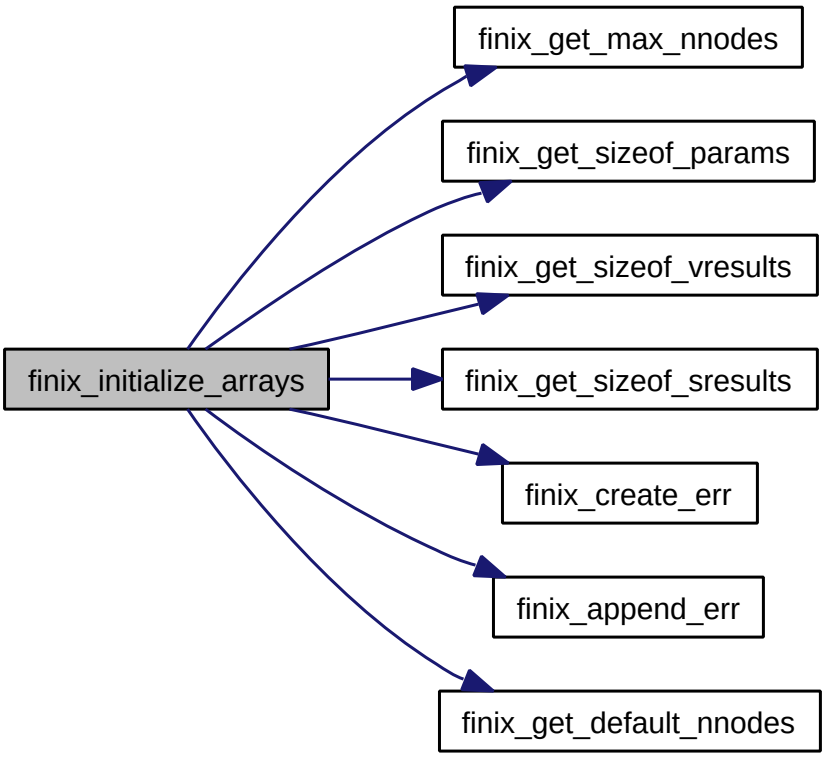
r	node positions
r_cold	node positions at cold state (zero strain)
power_dist	power distribution at each node
linear_power	linear power at each axial node
burnup	burn-up at each node
params	miscellaneous parameters: params[0] = rod dimensions, params[1] = pellet properties, params[2] = cladding properties, params[3] = gas gap properties (including current pressure at sresults[4]), params[4] = coolant properties
bcond	boundary conditions: bcond[0] = cladding outer T, bcond[1] = coolant T, bcond[2] = heat transfer coeffienct from cladding, bcond[3] = heat flux from coolant; the used boundary condition is defined in options (see below)
sresults	scalar results
vresults	various data computed by the model, each for every axial node: vresults[0] = gap heat transfer coefficient, vresults[1] = cladding radial strain, vresults[2] = cladding hoop strain, vresults[3] = cladding axial strain, vresults[4] = pellet axial strain, vresults[5] = pellet-cladding contact pressure, vresults[6] = cladding hoop stress, vresults[7] = cladding axial stress, vresults[8] = pellet axial strain with previous open gap, vresults[9] = cladding axial strain with previous open gap, vresults[10] = fuel swelling strain, vresults[11] = fuel densification strain, vresults[12] = fuel hard relocation strain, vresults[13] = fuel soft relocation strain, vresults[14] = clad plastic hoop strain, vresults[15] = clad plastic axial strain, vresults[16] = clad plastic radial strain, vresults[17] = clad effective plastic strain

Returns

error string, NULL for no errors

Definition at line 129 of file defaults.c.

Here is the call graph for this function:



2.8.2.16 char** finix_set_constant_power_dist (int ** *nnodes*, double ** *power_dist*)

Sets uniform power distribution in all nodes.

Parameters

nnodes	numbers of nodes
power_dist	power distribution

Returns

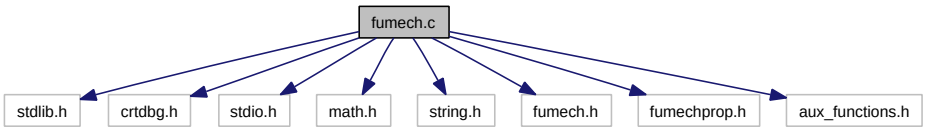
error string, NULL for no errors

Definition at line 668 of file defaults.c.

2.9 fumech.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.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, int *nnodes, double **T, double **r, double **r_cold, double **power_den, double **burnup, double **params, double **vresults, int *options)
- char ** finix_mech_solve_rigid_pellet (int *nnodes, double **T, double **r, double **r_cold, double **power_den, double **burnup, double **params, double *sresults, double **vresults, int *options)

2.9.1 Macro Definition Documentation

2.9.1.1 #define _CRTDBG_MAP_ALLOC

Definition at line 13 of file fumech.c.

2.9.2 Function Documentation

2.9.2.1 char** finix_mech_solve_pellet_strain (int *zind*, int * *nnodes*, double ** *T*, double ** *r*, double ** *r_cold*, double ** *power_den*, double ** *burnup*, double ** *params*, double ** *vresults*, int * *options*)

Solves the strains for one axial node with the rigid pellet appoximation.

Parameters

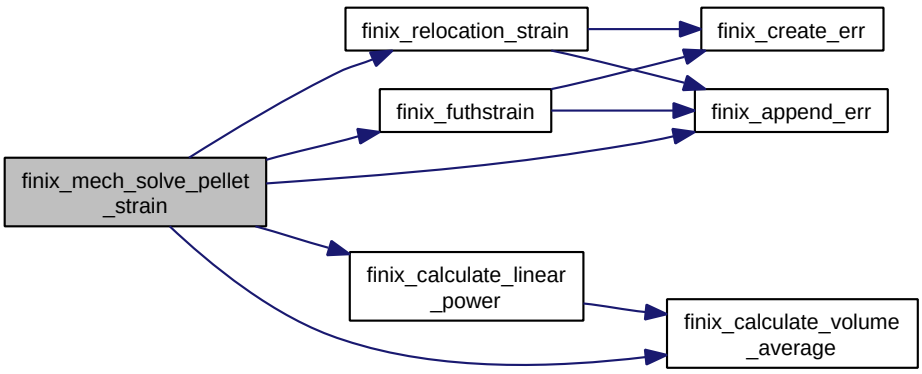
zind	index of the axial slice
nnodes	number of nodes
T	temperature (K)
r	locations of nodes (m)
r_cold	locations of nodes in cold state (m)
power_den	power density (W/m^3)
burnup	burnup at each node (MWd/kgU)
params	system parameters (scalars)
vresults	computed results (vectors)
options	model options

Returns

error string, NULL for no errors

Definition at line 41 of file fumech.c.

Here is the call graph for this function:



2.9.2.2 char** finix_mech_solve_rigid_pellet (int * *nnodes*, double ** *T*, double ** *r*, double ** *r_cold*, double ** *power_den*, double ** *burnup*, double ** *params*, double * *sresults*, double ** *vresults*, int * *options*)

Solves the pellet mechanical deformations with the rigid pellet appoximation.

Parameters

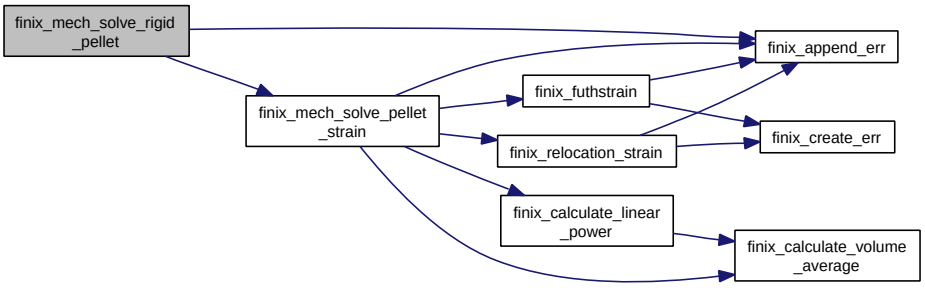
nnodes	number of nodes
T	temperature (K)
r	locations of nodes (m)
r_cold	locations of nodes in cold state (m)
power_den	power density (W/m^3)
burnup	burnup at each node (MWd/kgU)
params	system parameters (scalars)
sresults	results in scalar form
vresults	computed results (vectors)
options	model options

Returns

error string, NULL for no errors

Definition at line 140 of file fumech.c.

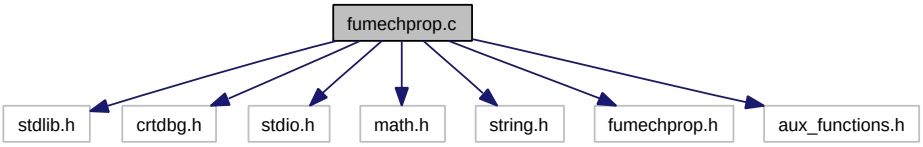
Here is the call graph for this function:



2.10 fumechprop.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.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, int *options, double *strain)
- char ** finix_calculate_density (int *nnodes, double **r, double **r_cold, double **params, double **vresults, double **den)

2.10.1 Macro Definition Documentation

2.10.1.1 #define _CRTDBG_MAP_ALLOC

Definition at line 13 of file fumechprop.c.

2.10.2 Function Documentation

2.10.2.1 char** finix_calculate_density (int * *nnodes,* double ** *r,* double ** *r_cold,* double ** *params,* double ** *vresults,* 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

nnodes	number of nodes
r	node positions (m)
r_cold	node positions in the cold state (m)
params	model parameters
vresults	results per axial node
den	density at to be calculated for each node (kg/m^3)

Returns

error string, NULL for no errors

Definition at line 143 of file fumechprop.c.

2.10.2.2 char** finix_futhstrain (double *T,* double * *strain*)

Fuel thermal strain

Calculates the thermal strain of UO2 with the FRAPTRAN-1.4 correlation.

Parameters

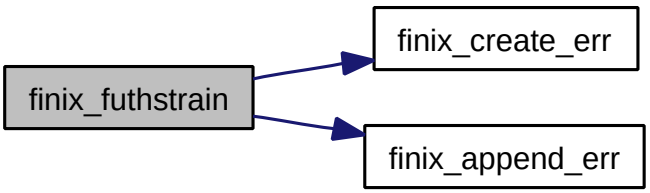
T	temperature (K)
strain	thermal strain

Returns

error string, NULL for no errors

Definition at line 33 of file fumechprop.c.

Here is the call graph for this function:



2.10.2.3 char** finix_relocation_strain (double *lhr,* double *bu,* double *rfo,* double *rci,* int * *options,* double * *strain*)

Fuel relocation strain. Added 30 May 2013 by Timo Ikonen.

Calculates the radial pellet relocation.

Parameters

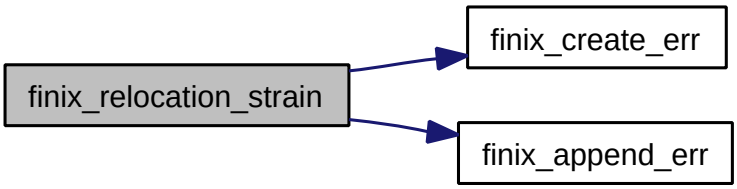
lhr	pellet average linear heat rate (W/m)
bu	pellet average burnup (MWd/kgU)
rfo	fuel outer radius in the cold state (m)
rci	cladding inner radius in the cold state (m)
options	model options
strain	relocation strain (output)

Returns

error string, NULL for no errors

Definition at line 70 of file fumechprop.c.

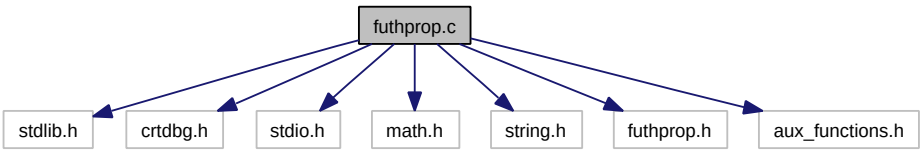
Here is the call graph for this function:



2.11 futhprop.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.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)

2.11.1 Macro Definition Documentation

2.11.1.1 #define _CRTDBG_MAP_ALLOC

Definition at line 12 of file futhprop.c.

2.11.2 Function Documentation

2.11.2.1 char** finix_fucp (double *T*, double * *cp*)

Fuel specific heat

Calculates the specific heat of UO2 with the FRAPTRAN-1.4 correlation.

Parameters

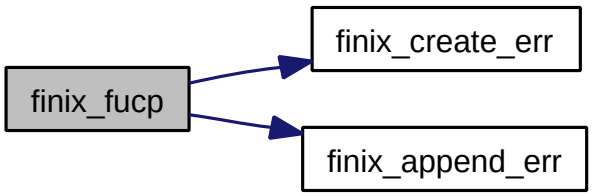
T	temperature (K)
cp	specific heat

Returns

error string, NULL for no errors

Definition at line 73 of file futhprop.c.

Here is the call graph for this function:



2.11.2.2 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

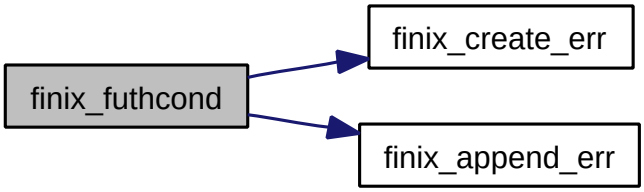
T	temperature (K)
den	as-fabricated density as a fraction from the theoretical value
gad	gadolinia weight fraction
Bu	burn-up (GWd/MTU)
lambda	thermal conductivity (W/mK)

Returns

error string, NULL for no errors

Definition at line 36 of file futhprop.c.

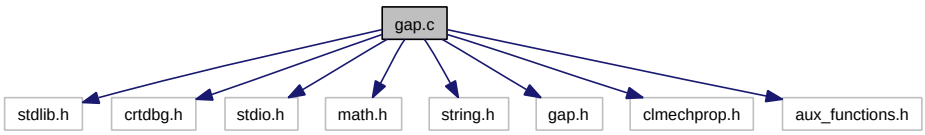
Here is the call graph for this function:



2.12 gap.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.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, int *nnodes, double **T, double **r, double **r_cold, double **lambda, double **params, double *sresults, double **vresults, int *options)
- char ** finix_Tplenum (int *nnodes, double **T, double **r, double **lambda, double **params, double *sresults, double **bcond)
- char ** finix_pgap (int *nnodes, double **T, double **r, double **r_cold, double **params, double *sresults, double **vresults)
- char ** finix_gap_moles_from_pressure (int *nnodes, double **T, double **r, double **r_cold, double **params, double *sresults, double **vresults)
- char ** finix_gap_moles_from_pressure_cold (int *nnodes, double **T, double **r, double **r_cold, double **params, double *sresults, double **vresults)

2.12.1 Macro Definition Documentation

2.12.1.1 #define _CRTDBG_MAP_ALLOC

Definition at line 14 of file gap.c.

2.12.2 Function Documentation

2.12.2.1 char** finix_gap_moles_from_pressure (int * *nnodes,* double ** *T,* double ** *r,* double ** *r_cold,* double ** *params,* double * *sresults,* double ** *vresults*)

Calculates the amount of gas (moles) from the gas pressure, temperature and volume.

Parameters

nnodes	number of nodes
T	temperature (K)
r	locations of nodes (m)
r_cold	cold state locations of nodes (m)
params	system parameters
sresults	results in scalar form (including the gas amount to be calculated in sresults[5])
vresults	results array, including the fuel pellet axial strains in vresults[4]

Returns

error string, NULL for no errors

Definition at line 496 of file gap.c.

2.12.2.2 char** finix_gap_moles_from_pressure_cold (int * *nnodes,* double ** *T,* double ** *r,* double ** *r_cold,* double ** *params,* double * *sresults,* double ** *vresults*)

Calculates the amount of gas (moles) from the gas pressure, temperature and volume using cold state (300 K) values.

Parameters

nnodes	number of nodes
T	temperature (K)
r	locations of nodes (m)
r_cold	cold state locations of nodes (m)
params	system parameters
sresults	results in scalar form (including the gas amount to be calculated in sresults[5])
vresults	results array, including the fuel pellet axial strains in vresults[4]

Returns

error string, NULL for no errors

Definition at line 594 of file gap.c.

2.12.2.3 char** finix_hgap (int *zind,* int * *nnodes,* double ** *T,* double ** *r,* double ** *r_cold,* double ** *lambda,* double ** *params,* double * *sresults,* double ** *vresults,* int * *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
nnodes	number of nodes
T	temperature (K)
r	locations of nodes (m)
r_cold	locations of nodes in cold state (m)
lambda	conductivity at each node (W/mK)
params	system parameters (scalars)
sresults	results in scalar form
vresults	computed results (vectors)
options	model options

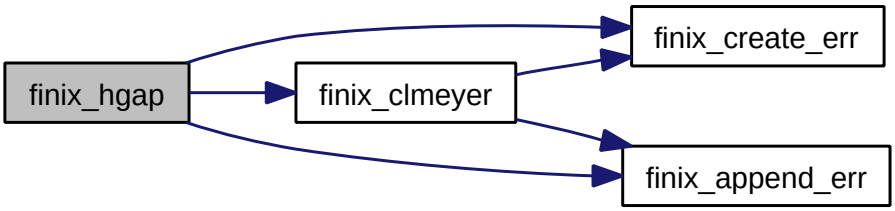
Returns

error string, NULL for no errors

currently pure helium is assumed; should add A and B for other gases' conductivities

Definition at line 46 of file gap.c.

Here is the call graph for this function:



2.12.2.4 char** finix_pgap (int * *nnodes*, double ** *T*, double ** *r*, double ** *r_cold*, double ** *params*, double * *sresults*, double ** *vresults*)

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

nnodes	number of nodes
T	temperature (K)
r	locations of nodes (m)
r_cold	cold state locations of nodes (m)
params	system parameters
sresults	results in scalar form (including the pressure to be calculated in sresults[4])
vresults	results array, including the fuel pellet axial strains in vresults[4]

Returns

error string, NULL for no errors

Definition at line 397 of file gap.c.

2.12.2.5 char** finix_Tplenum (int * *nnodes*, double ** *T*, double ** *r*, double ** *lambda*, double ** *params*, double * *sresults*, double ** *bcond*)

Plenum gas temperature

Calculates the plenum gas temperature. Assumes helium for gas properties.

Parameters

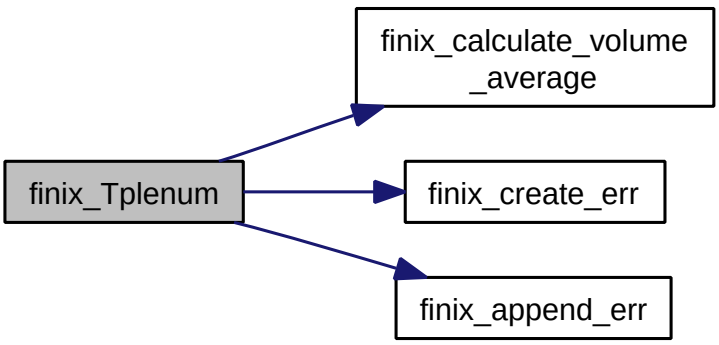
nnodes	number of nodes
T	temperature (K)
r	locations of nodes (m)
lambda	heat conductivity at each node
params	system parameters
sresults	results in scalar form (including plenum temperature at sresults[3] and pressure at sresults[4])
bcond	boundary conditions (gives, e.g., coolant temperature)

Returns

error string, NULL for no errors

Definition at line 240 of file gap.c.

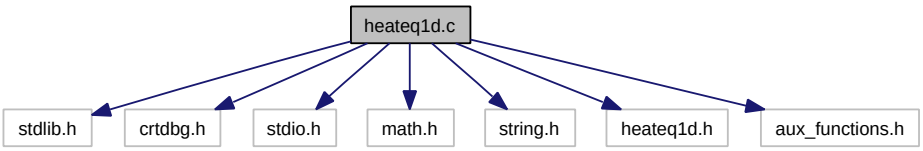
Here is the call graph for this function:



2.13 heateq1d.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.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, int *nnodes, double **T, double **r, double **power_den, double **lambda, double **cv, double **params, double **vresults, double **bcond, int *options, double **diag, double **subdiag, double **superdiag, double **loadvec)
- char ** finix_FEM_solve_tridiagonal (int zind, int *nnodes, double **T, double **diag, double **subdiag, double **superdiag, double **loadvec)

2.13.1 Macro Definition Documentation

2.13.1.1 #define _CRTDBG_MAP_ALLOC

Definition at line 12 of file heateq1d.c.

2.13.2 Function Documentation

2.13.2.1 char** finix_FEM_discretize_HE_1D (int *zind*, double *dt*, int * *nnodes*, double ** *T*, double ** *r*, double ** *power_den*, double ** *lambda*, double ** *cv*, double ** *params*, double ** *vresults*, double ** *bcond*, int * *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
nnodes	number of nodes
T	temperature (K)
r	locations of nodes (m)
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)
params	system parameter array
vresults	results vector (including gap conductance)
bcond	boundary conditions
options	model options (needed to select boundary condition)
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[0]==0, use user-given rod outer surface temperature as boundary condition

If options[0]==1, use user-given heat flux between rod outer surface and coolant

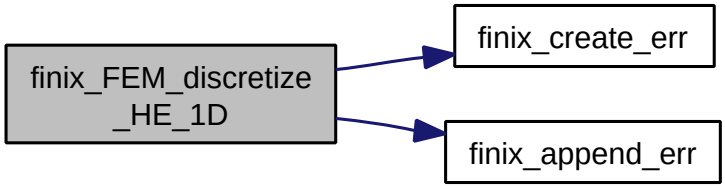
If options[0]==2, use user-given heat transfer coefficient and coolant bulk temperature as boundary condition

If options[0]==3, use user-given bulk temperature and internally calculated heat transfer coefficient

If options[0]==4, use user-given inlet temperature and mass flux to calculate heat transfer coefficient and coolant bulk temperature downstream (NOT IMPLEMENTED)

Definition at line 47 of file heateq1d.c.

Here is the call graph for this function:



2.13.2.2 char** finix_FEM_solve_tridiagonal (int *zind*, int * *nnodes*, double ** *T*, 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
nnodes	number of nodes
T	temperature (K) (function output to be solved; must be initialized beforehand)
diag	matrix diagonal
subdiag	matrix sub-diagonal
superdiag	matrix super-diagonal
loadvec	load vector

Returns

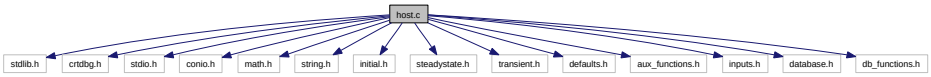
error string, NULL for no errors

Definition at line 258 of file heateq1d.c.

2.14 host.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <string.h>
#include "initial.h"
#include "steadystate.h"
#include "transient.h"
#include "defaults.h"
#include "aux_functions.h"
#include "inputs.h"
#include "database.h"
#include "db_functions.h"
```

Include dependency graph for host.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- int main ()

2.14.1 Macro Definition Documentation

2.14.1.1 #define _CRTDBG_MAP_ALLOC

Definition at line 14 of file host.c.

2.14.2 Function Documentation

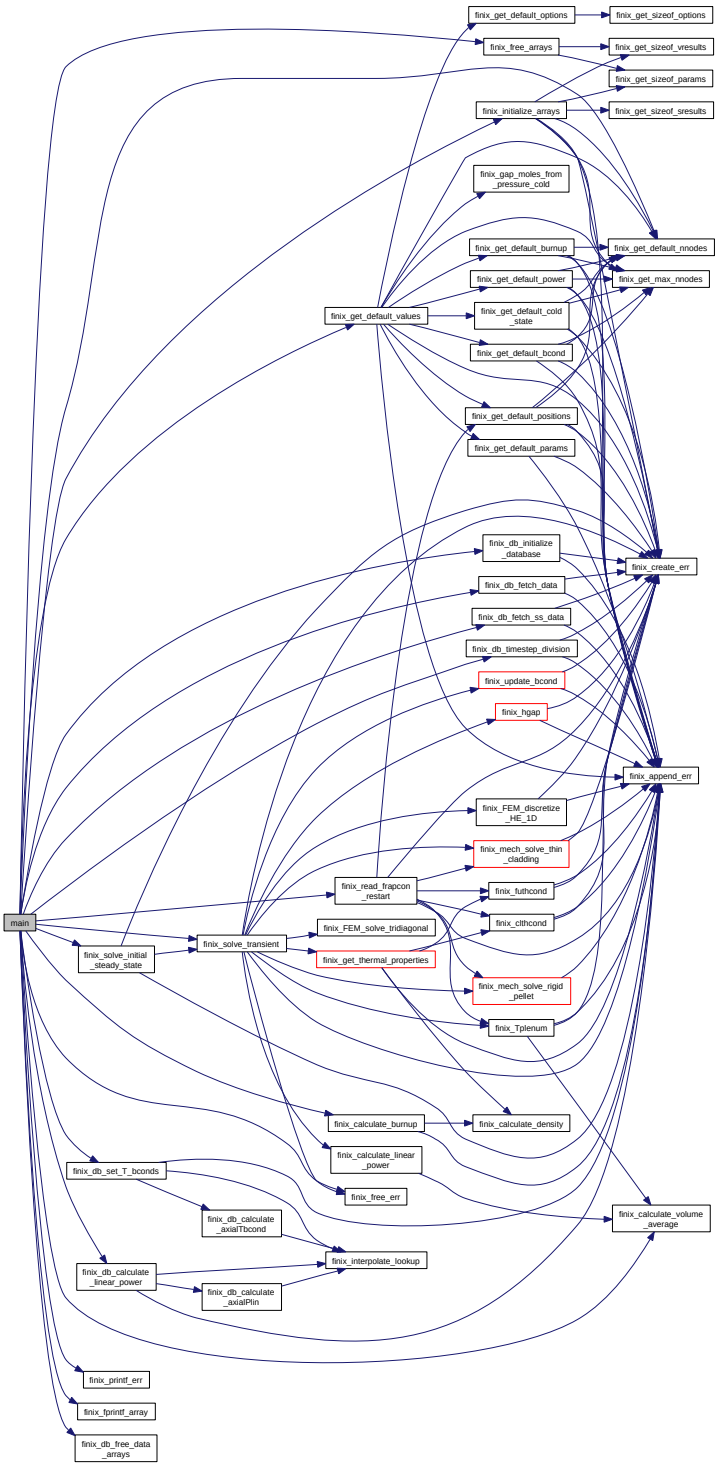
2.14.2.1 int main ( )

The host code file.

The host code that runs the FINIX subprogram. This file should be replaced in its entirety with the host code.

Definition at line 41 of file host.c.

Here is the call graph for this function:

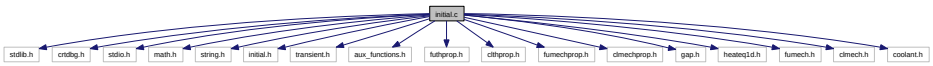


2.15 initial.c File Reference

```
#include <stdlib.h>
```

```
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.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 (int *nnodes, double **T, double **r, double **r_cold, double **power_dist, double *linear_power, double **burnup, double **params, double **bcond, double *sresults, double **vresults, int *options)

2.15.1 Macro Definition Documentation

2.15.1.1 #define _CRTDBG_MAP_ALLOC

Definition at line 12 of file initial.c.

2.15.2 Function Documentation

2.15.2.1 char** finix_solve_initial_steady_state (int * *nnodes*, double ** *T*, double ** *r*, double ** *r_cold*, double ** *power_dist*, double * *linear_power*, double ** *burnup*, double ** *params*, double ** *bcond*, double * *sresults*, double ** *vresults*, int * *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

nnodes	numbers of nodes
T	the temperature distribution (K)
r	the radial node positions of the pellet and the cladding (m)
r_cold	the radial node positions of the pellet and the cladding in the cold state (m)
power_dist	power distribution at each node (unitless)

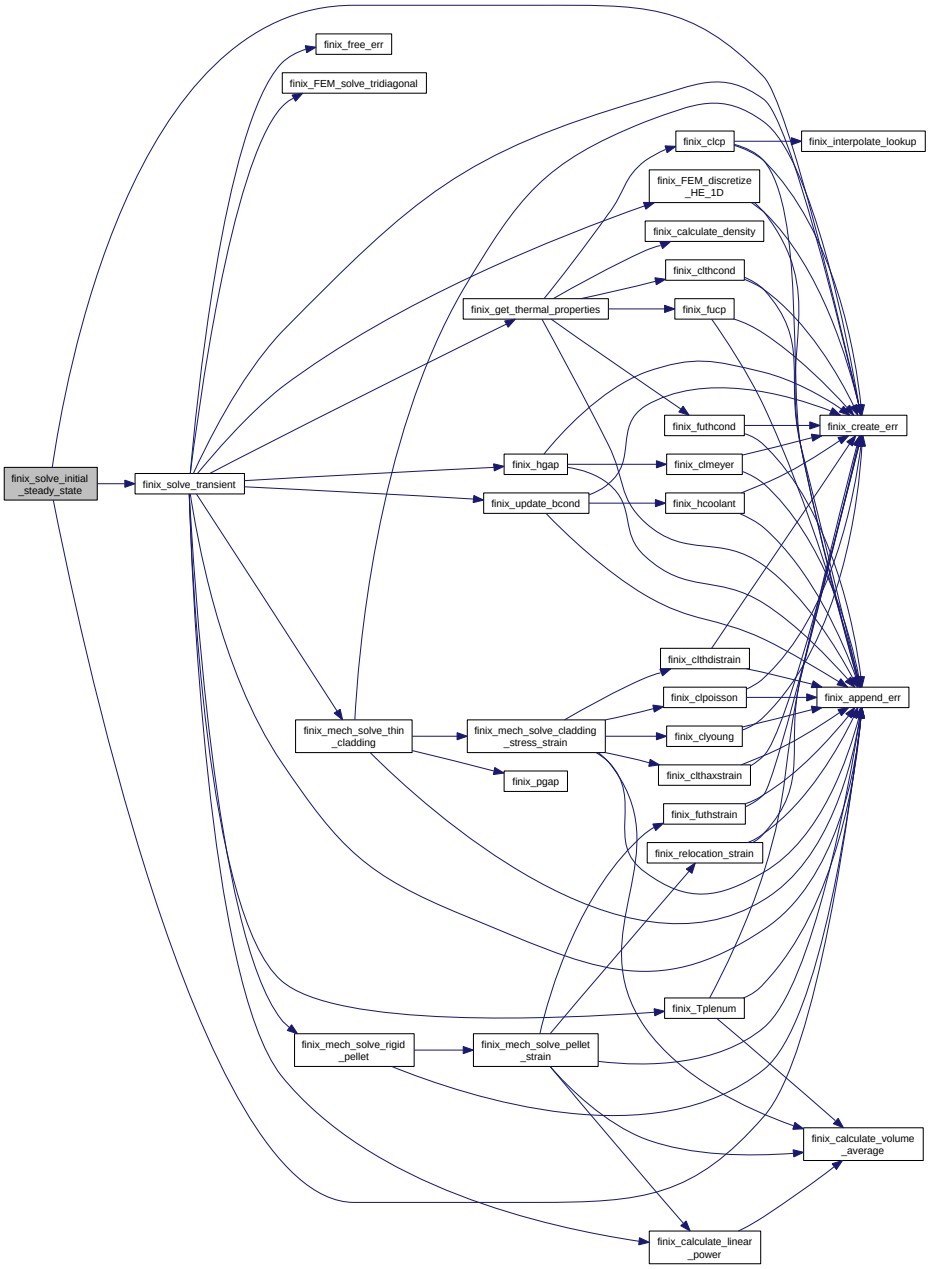
linear_power	linear power at each axial node (W/m)
burnup	burn-up at each node (GWd/MTU); defined also for cladding nodes, although not used
params	miscellaneous system parameters in scalar form
bcond	boundary conditions in vector form (for each axial node)
sresults	results expressed in scalar form
vresults	miscellanoues computed values in vector form (for each axial node)
options	model options

Returns

error string, NULL for no errors

Definition at line 58 of file initial.c.

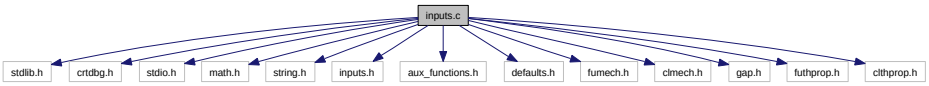
Here is the call graph for this function:



2.16 inputs.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "inputs.h"
#include "aux_functions.h"
#include "defaults.h"
#include "fumech.h"
#include "clmech.h"
#include "gap.h"
#include "futhprop.h"
#include "clthprop.h"
```

Include dependency graph for inputs.c:



Macros

- #define _CRTDBG_MAP_ALLOC

Functions

- char ∗∗ finix_read_frapcon_restart (int ∗nnodes, double ∗∗T, double ∗∗r, double ∗∗r_cold, double ∗∗power_dist, double ∗∗burnup, double ∗∗params, double ∗∗bcond, double ∗sresults, double ∗∗vresults, int ∗options, char ∗restartfilename)

2.16.1 Macro Definition Documentation

2.16.1.1 #define _CRTDBG_MAP_ALLOC

Definition at line 12 of file inputs.c.

2.16.2 Function Documentation

2.16.2.1 char∗∗ finix_read_frapcon_restart (int ∗ *nnodes,* double ∗∗ *T,* double ∗∗ *r,* double ∗∗ *r_cold,* double ∗∗ *power_dist,* double ∗∗ *burnup,* double ∗∗ *params,* double ∗∗ *bcond,* double ∗ *sresults,* double ∗∗ *vresults,* int ∗ *options,* char ∗ *restartfilename*)

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

Parameters

nnodes	numbers of nodes
T	the temperature distribution (K)
r	the radial node positions of the pellet and the cladding (m)
r_cold	the radial node positions of the pellet and the cladding in the cold state (m)
power_dist	(radial) power distribution at each axial node (dimensionless, normalized with cross-sectional area)
burnup	burn-up at each node (GWd/MTU); defined also for cladding nodes, although not used

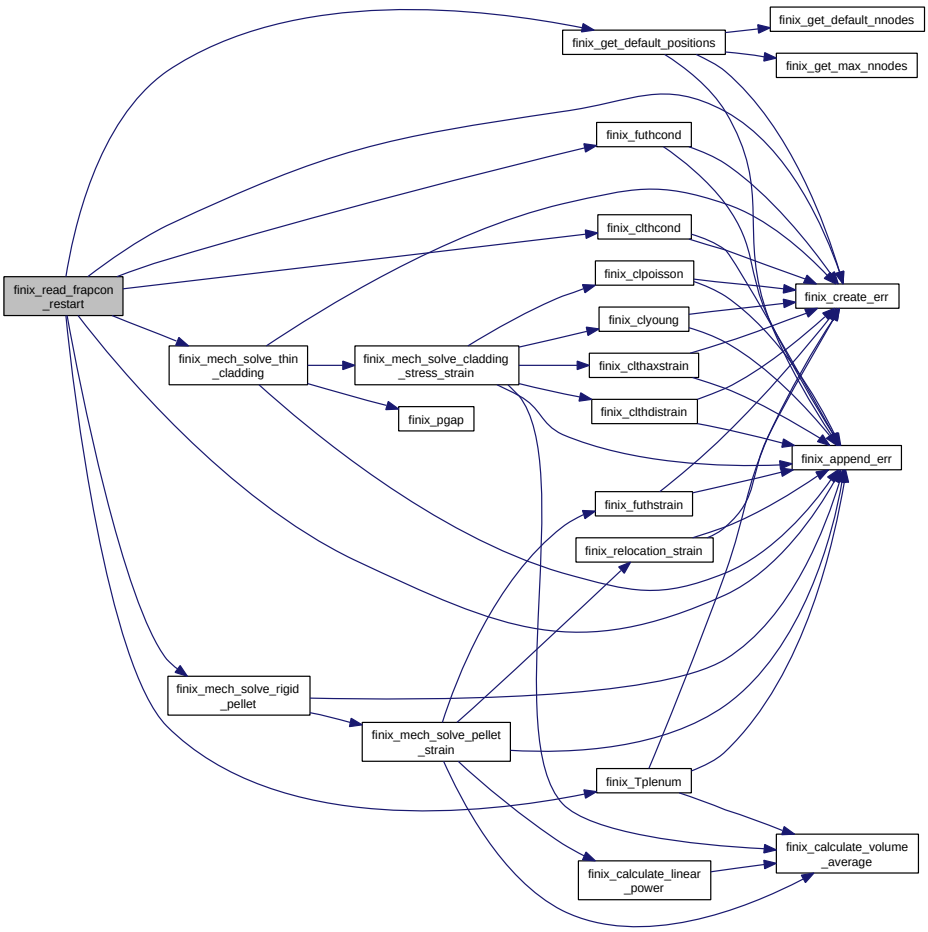
params	miscellaneous system parameters in scalar form
bcond	coolant boundary conditions
sresults	scalar results
vresults	miscellanoues computed values in vector form (for each axial node)
options	model options
restartfilename	name of the restart file to be read

Returns

error string, NULL for no errors

Definition at line 47 of file inputs.c.

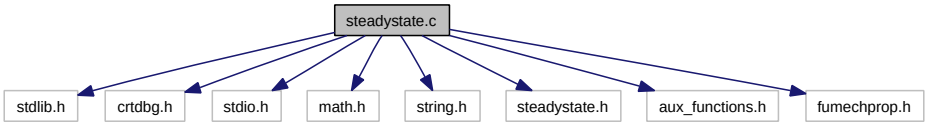
Here is the call graph for this function:



2.17 steadystate.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "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, int ∗nnodes, double ∗∗r, double ∗∗r_cold, double ∗∗power_dist, double ∗linear_power, double ∗∗burnup, double ∗∗params, double ∗∗vresults)

2.17.1 Macro Definition Documentation

2.17.1.1 #define _CRTDBG_MAP_ALLOC

Definition at line 13 of file steadystate.c.

2.17.2 Function Documentation

2.17.2.1 char∗∗ finix_calculate_burnup (double *time*, int ∗ *nnodes*, double ∗∗ *r*, double ∗∗ *r_cold*, double ∗∗ *power_dist*, double ∗ *linear_power*, double ∗∗ *burnup*, double ∗∗ *params*, double ∗∗ *vresults*)

Calculates the accumulation of burnup during one constant-power time step.

Parameters

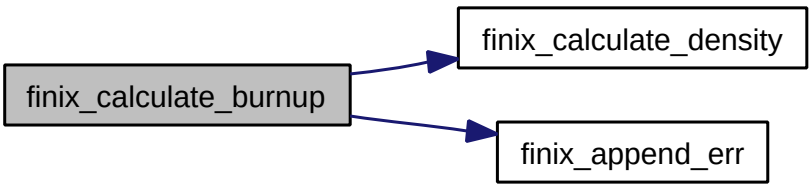
time	duration (time step) of burnup accumulation
nnodes	numbers of nodes
r	the radial node positions of the pellet and the cladding (m)
r_cold	the radial node positions of the pellet and the cladding in the cold state (m)
power_dist	radial power distribution
linear_power	linear power at each axial node (W/m)
burnup	burn-up at each node (MWd/kgU); defined also for cladding nodes, although not used
params	miscellaneous system parameters in scalar form
vresults	miscellanoues computed values in vector form (for each axial node)

Returns

error string, NULL for no errors

Definition at line 40 of file steadystate.c.

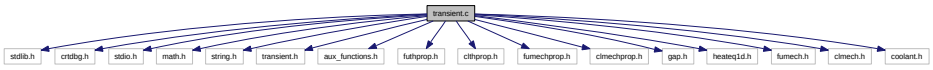
Here is the call graph for this function:



2.18 transient.c File Reference

```
#include <stdlib.h>
#include <crtdbg.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "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, int *nnodes, double **T, double **r, double **r_cold, double **power_dist, double *linear_power, double **burnup, double **params, double **bcond, double *sresults, double **vresults, int *options)
- char ** finix_get_thermal_properties (int *nnodes, double **T, double **r, double **r_cold, double **power_den, double **burnup, double **params, double **vresults, int *options, double **lambda, double **cp, double **den, double **cv)

2.18.1 Macro Definition Documentation

2.18.1.1 #define _CRTDBG_MAP_ALLOC

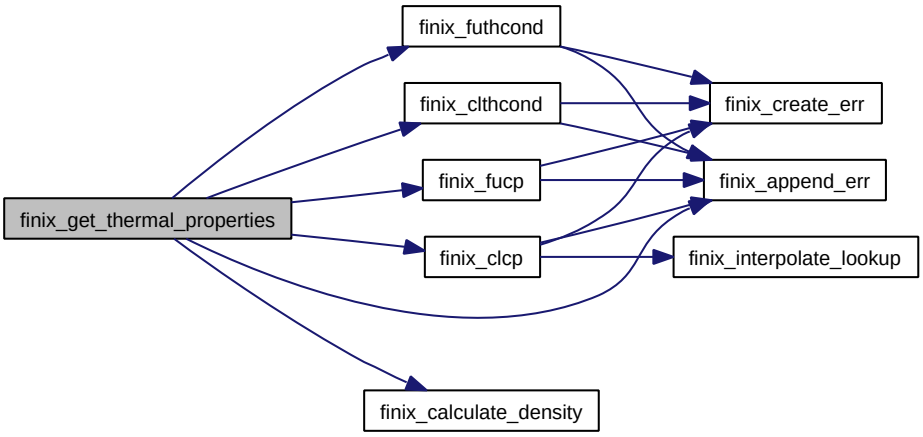
Definition at line 17 of file transient.c.

2.18.2 Function Documentation

2.18.2.1 char** finix_get_thermal_properties (int * *nnodes,* double ** *T,* double ** *r,* double ** *r_cold,* double ** *power_den,* double ** *burnup,* double ** *params,* double ** *vresults,* int * *options,* double ** *lambda,* double ** *cp,* double ** *den,* double ** *cv*)

Definition at line 462 of file transient.c.

Here is the call graph for this function:



2.18.2.2 char** finix_solve_transient (double *dt,* int * *nnodes,* double ** *T,* double ** *r,* double ** *r_cold,* double ** *power_dist,* double * *linear_power,* double ** *burnup,* double ** *params,* double ** *bcond,* double * *sresults,* double ** *vresults,* int * *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. Other output can be extracted from the *params pointer.

Parameters

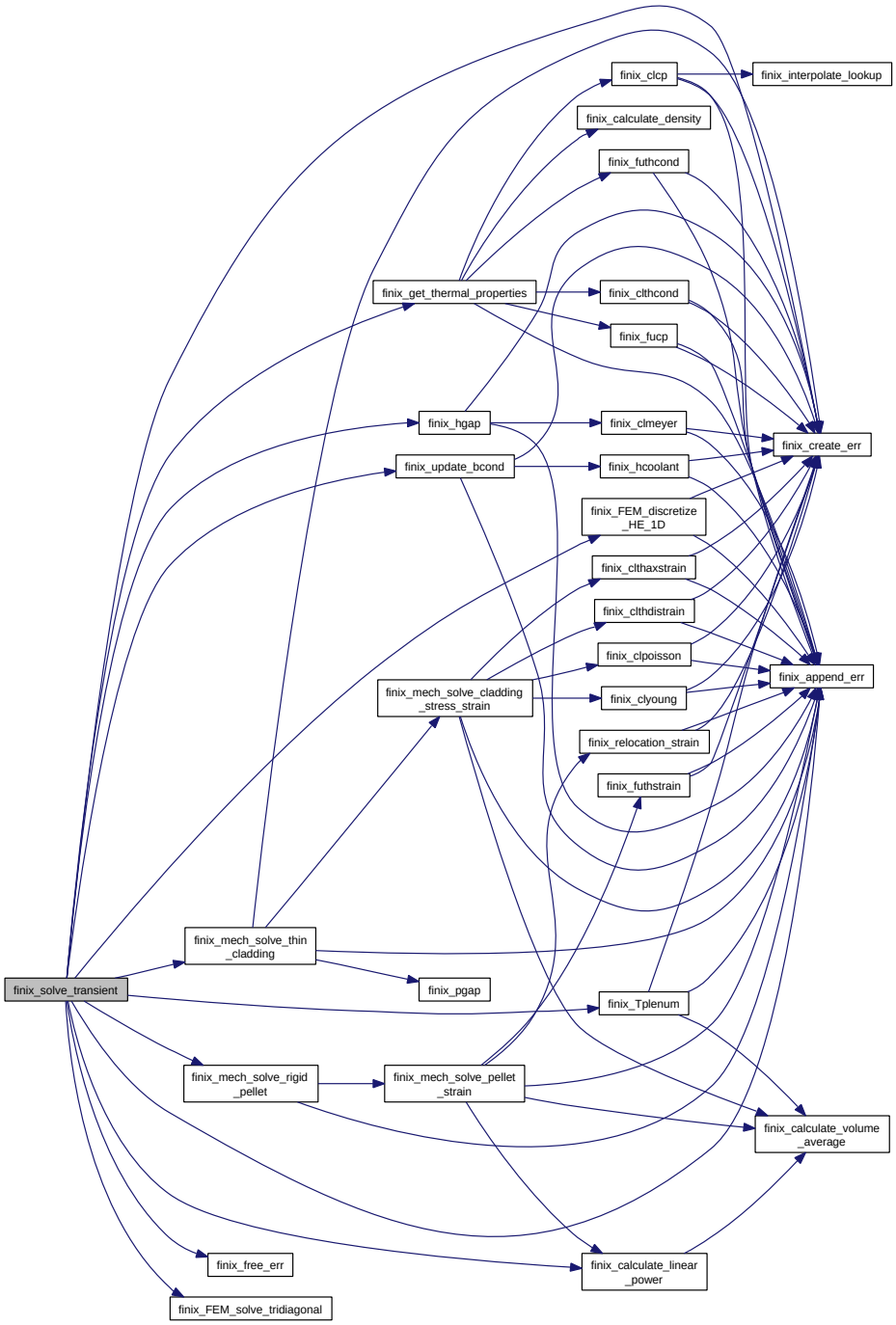
dt	time step (seconds); the function solves the thermal and mechanical time evolution of the fuel rod and propagates solution by dy in time
nnodes	numbers of nodes
T	the temperature distribution (K)
r	the radial node positions of the pellet and the cladding (m)
r_cold	the radial node positions of the pellet and the cladding in the cold state (m)
power_dist	radial power distribution at each node (dimensionless, normilzed by cross-sectional area)
linear_power	linear power at each axial node (W/m)
burnup	burn-up at each node (GWd/MTU); defined also for cladding nodes, although not used
params	miscellaneous system parameters in scalar form
bcond	boundary conditions in vector form (for each axial node)
sresults	results expressed in scalar form
vresults	miscellanoues computed values in vector form (for each axial node)
options	model options

Returns

error string, NULL for no errors

Definition at line 62 of file transient.c.

Here is the call graph for this function:



Index