TECHNICAL REPORT
IGE–344

A USER GUIDE FOR DONJON VERSION5

A. Hébert, D. Sekki, and R. Chambon

Institut de génie nucléaire
Département de génie mécanique
École Polytechnique de Montréal
May 4, 2016

Contents

List of Tables

List of Figures

1 INTRODUCTION

DONJON is a full-core modelization code designed around solution techniques of the neutron diffusion or simplified P_n equation.[?] The current DONJON package is an evolution version, released as an attempt to introduce the innovative capabilities for the full-core modeling and simulations of different types of nuclear reactors sush as Pressurized Water Reactors (PWRs), legacy CANDU reactors, and Advanced CANDU Reactors (ACRs). The computer code DONJON (Release 4.0) is part of Version4 distribution[?], built around the GAN generalized driver[?]. The current DONJON package (DONJON Version5) is a rewrite of the code around the GANLIB5 kernel[?], intended to be 64-bit clean.

DONJON execution depends on other computer codes, components of Version4, namely: GANLIB, UTILIB, DRAGON[?], and TRIVAC[?] codes. The DRAGON modules are used with DONJON code to define the reactor geometry, to provide the macroscopic cross-section libraries and to perform micro-depletion calculations. The TRIVAC solver modules are used to perform a spatial discretization of the reactor geometry and to provide the numerical solution according to the user-selected numerical procedure[?,?,?,?,?,?]. The UTILIB library provides the utility and linear algebra libraries. Finally, the GANLIB computer code provides CLE-2000 capabilities to control data flows and to implement *computational schemes*. GANLIB also provide LCM data structures to exchange information between modules.

The DONJON code is divided into several modules, each module is designed to perform some particular tasks. The transfer of information between the modules is achieved by means of well defined data structure. Several design features, data structure and computing algorithms were recovered, revised and adapted from the previous DONJON version[?,?]. One of the main concerns of the DONJON developers is to ensure the code reliability and extensibility.

The DONJON modules are first designed for the reactor full-core modeling in *3-D* Cartesian geometry. These modules are built around the reactor fuel lattice specification corresponding to the common design features of CANDU reactors. The modules related to the modeling of reactivity mechanisms, which are normally presented in the reactor core, also constitute an important part of code. The DONJON code can perform several full-core calculations and can be used to determine some important core characteristics, such as the power and normalized flux distributions over the reactor core. All full-core calculations using current version of DONJON correspond to the reactor static conditions.

The modeling of the reactor fuel lattice using DONJON is made in considering that the fuel lattice is composed of a well defined number of fuel channels and bundles. All reactor channels contain the same number of fuel bundles and are identified by their specific names. The fuel bundles have a distinct set of properties that are recovered and interpolated according to the specified global and local parameters. The interpolation of fuel properties with respect to burnup distribution can be performed according to the time-average or instantaneous models[?]. The time-average calculation is performed in considering the bidirectional refuelling scheme of reactor channels and assuming that all channels have the same bundle-shift.

The modeling of the reactivity mechanisms is based on their specified parameters, which include the devices position, rods insertion level, water filling level, direction of movement, etc. The rod-devices insertion level can be set according to their nominal positions or they can be displaced in and out of core. The devices can also be divided into several groups so that they can be manipulated, displaced or moved simultaneously. The time-dependent behaviour of the moving devices can be modeled and used for the transient simulations or reactor control studies. The reactivity worth of devices can also be studied and predicted using DONJON.

The reactor material properties are essentially recovered from the reactor database, obtained from the lattice calculations using DRAGON code. The two distinct macroscopic cross-section libraries can be constructed using DONJON. The first MACROLIB is constructed only for the material properties which are evolution-independent, such as reflector and devices properties. The second MACROLIB is constructed only for the fuel properties, defined per each fuel bundle over the fuel lattice. The two libraries are next combined and updated, according to the devices insertion level. The produced extended MACROLIB is subsequently used to obtain the numerical solution, using TRIVAC modules.

Finally, it should be noted that the DONJON code development is permanently in progress. The future updates will provide several extended capabilities for the reactor design and calculations; they will be gradually added to the subsequent DONJON versions.

2 GENERAL SPECIFICATION OF DONJON

2.1 Modules

Reactor calculations using DONJON are performed by means of sequential execution of several user-selected modules, according to the user-defined computing scheme. Each module is designed to perform some particular tasks. The detailed description of DONJON modules is given in Section **??** to Section **??**. In order to perform the reactor calculations, it is also required to use some DRAGON and TRIVAC modules. For more details on DRAGON modules specification, refer to its user guide[?]; for more details on TRIVAC modules specification, refer to its user guide[?]. Because the code execution is controlled by the GAN generalized driver, it is also possible to use its utility modules[?, ?]. A brief description of each module that can be executed using DONJON is given below. A short description of each data structure that can be used in DONJON is given in Section **??**.

- The following DRAGON modules can be executed using DONJON:

`GEO:`	module used to create or modify a reactor geometry. The spatial locations of the reactor material mixtures must also be defined using the `GEO:` module. Only *3-D* Cartesian reactor geometries are allowed with DONJON.
`MAC:`	module used to create or modify a MACROLIB containing the material properties, by directly specifying the group-ordered macroscopic cross-sections for each selected material mixture.

- The following TRIVAC modules can be executed using DONJON:

`TRIVAT:`	module used to perform a *3-D* numerical discretization or “tracking” of the reactor geometry.
`TRIVAA:`	module used to compute the set of system matrices with respect to the previously obtained ”tracking” information.
`FLUD:`	module used to compute the numerical solution to an eigenvalue problem, corresponding to a previously obtained set of system matrices.

- The following are short descriptions of utility modules that can be executed using DONJON:

`UTL:`	module used to perform several utility actions on a data structure.
`DELETE:`	module used to delete one or many data structures.
`GREP:`	module used to extract a single value from a data structure.
`END:`	module used to delete all the local linked lists, to close all the remaining local files and to return from a procedure; or to terminate the overall DONJON execution controlled by the GAN generalized driver.

- The following are short descriptions of DONJON modules:

`CRE:`	module used to create a MACROLIB containing the material properties, by interpolating the nuclear properties from a mono-parameter database, previously generated in the lattice code.
`NCR:`	module used to create a MICROLIB or a MACROLIB containing the material properties, by interpolating the nuclear properties from a multi-parameter database, previously generated in the lattice code.
`AFM:`	module used to create a MACROLIB containing the material properties, by interpolating the nuclear properties from a multi-parameter feedback model database, previously generated in the lattice code.
`USPLIT:`	module used to create an extended reactor material index over the whole mesh-splitted reactor geometry.
`RESINI:`	module used to define the fuel lattice, to create the fuel-map geometry and to specify the global and local parameters.
`MACINI:`	module used to create an extended MACROLIB, in which the properties are stored per each material region, over the whole mesh-splitted reactor geometry.
`DEVINI:`	module used for *3-D* modeling of rod-type devices in the reactor core.
`DETINI:`	module used to read and store detector information.
`LZC:`	module used for *3-D* modeling of liquid zone controllers in the reactor core.
`DSET:`	module used to set the new devices parameters, that can be used for the reactivity worth studies.
`MCC:`	module used to modify the fuel map in order to compute a Doppler or general reactivity coefficient.
`MOVDEV:`	module used to compute the time-dependent positions of the moving rod-type devices.
`NEWMAC:`	module used to create an extended MACROLIB, that will contain the updated material properties, computed with respect to the actual devices positions.
`FLPOW:`	module used to compute and print powers and normalized fluxes over the reactor core.
`TAVG:`	module used to perform burnups calculation according to the time-average model, compute burnups integration limits, core-average exit burnup, axial power-shapes and channel refuelling rates.
`TINST:`	module used to perform burnups calculation according to the time-linear model and compute instantaneous burnups values. This module is specific to Candu reactor refuelling.
`SIM:`	module used to perform burnups calculation according to the time-linear model and compute instantaneous burnups values. This module is specific to PWR reactor refuelling.
`DETECT:`	module used to compute the mean flux at each detector site and the response of each detector according to different types of interpolation.
`CVR:`	module used for the core-voiding simulations.
`HST:`	module used to manage a full reactor execution in DONJON using explicit DRAGON calculations for each cell (see Section **??**).[?]
`NAP:`	pin power reconstruction module.[?, ?]

2.2 Data structures

The transfer of information between the modules is performed by means of well defined data structures, also called objects. The objects can be defined in either create, read-only or modification mode. Each object has its own specific signature that can be easily recognized by a module. A detailed description of DONJON data structures is given in Section **??**. For more details on DRAGON and TRIVAC data structures, refer to their guide[?]. A brief description of all data structures that can be used in DONJON is given below.

GEOMETRY	data structure containing the geometry information. This object has a signature `L_GEOM`; it is created using DRAGON module `GEO:`.
MACROLIB	data structure containing the multigroup macroscopic properties; it has a signature `L_MACROLIB`. This object can be created in several modules, namely: using DRAGON modules `MAC:` and `NCR:`; or using DONJON modules `CRE:`, `MACINI:`, and `NEWMAC:`.
COMPO	data structure containing the mono-parameter database, generated by the lattice code. This object has a signature `L_COMPO`; it is created using DRAGON module `CPO:`.
MULTICOMPO	data structure containing the multi-parameter database, generated by the lattice code. This object has a signature `L_MULTICOMPO`; it is created using DRAGON module `COMPO:`.
SAPHYB	data structure containing the multi-parameter database, generated by the lattice code. This object has a signature `L_SAPHYB`; it is created using the APOLLO2 lattice code or the DRAGON module `SAP:`.
FMAP	data structure containing the fuel-lattice specification. This object has a signature `L_MAP`; it is created using DONJON module `RESINI:`.
MATEX	data structure containing the extended reactor material index. This object has a signature `L_MATEX`; it is created using DONJON module `USPLIT:`.
DEVICE	data structure containing the devices specification. This object has a signature `L_DEVICE`; it is created using DONJON module `DEVINI:`.
DETECT	data structure containing detector positions and responses. This object has a signature `L_DETECT`; it is created using DONJON module `DETINI:`, and can be modified by the modules `DETINI:` and `DETECT:` .
TRACK	data structure containing a "tracking" information of the reactor geometry. This object has a signature `L_TRACK`; it is created using TRIVAC module `TRIVAT:`.
SYSTEM	data structure containing a set of system matrices. This object has a signature `L_SYSTEM`; it is created using TRIVAC module `TRIVAA:`.
FLUX	data structure containing the numerical solution to an eigenvalue problem. This object has a signature `L_FLUX`; it is created using TRIVAC module `FLUD:`.
POWER	data structure containing the powers and normalized fluxes over the reactor core. This object has a signature `L_POWER`; it is created using DONJON module `FLPOW:`.
HISTORY	This data structure contains the information required to ensure a smooth coupling of DRAGON with DONJON when an history based full reactor calculation is to be performed. It is used only by the `HST:` module.

2.3 Syntactic rules for input specification

The input data to any module is read in free format using the subroutine `REDGET`. CLE-2000 variables[?, ?] are also allowed. The user guide for DONJON is written using the following convention:

- the parameters surrounded by single square brackets '[ ]' denote an optional input;
- the parameters surrounded by double square brackets '[[ ]]' denote an input which may be repeated as many times as needed;
- the parameters in braces separated by vertical bars '{ | | }' denote a choice where one and *only* one input is mandatory;
- the parameters in **bold face** and in brackets '( )' denote an input structure;
- the parameters in italics and in brackets with an index '(*data*(i) , i = 1, n)' denote a set of n inputs;
- the words using the typewriter font `KEYWORD` are character constants used as keywords;
- the words in italics denote the user-defined variables: they are lower-case and of integer type (when starting from *i* to *n*), or of real type (when starting from *a* to *h* or from *o* to *z*); or they are upper-case and of character type *CHARACTER*.

2.4 General input structure

DONJON is built around the GAN generalized driver[?, ?]. Accordingly, all the modules that will be used during the current execution must be first identified. It is also necessary to define the format of each object (data structure) that will be processed by these modules. Then, the modules required for the specific DONJON calculation are called successively, information being transferred from one module to the next via the objects. Finally, the execution of DONJON is terminated when it encounters the `END:` module, even if it is followed by additional data records in the input data stream. The general input data structure therefore follows the calling specifications given below:

Table 1: Structure **(DONJON)**

[`MODULE` [[*MODNAME*]] `;`]
[`LINKED_LIST` [[*STRNAME*]] `;`]
[`XSM_FILE` [[*STRNAME*]] `;`]
[`SEQ_BINARY` [[*STRNAME*]] `;`]
[`SEQ_ASCII` [[*STRNAME*]] `;`]
[[**(module)** `;`]]
`END: ;`

where

`MODULE`	keyword used to specify the names of all modules that will be used in the current DONJON execution.
MODNAME	`character*12` name of a DONJON, or DRAGON, or TRIVAC, or utility module. The list of modules that can be executed using DONJON code is provided in Section **??**.

`LINKED_LIST`	keyword used to specify the names of data structure that will be stored as linked lists.
`XSM_FILE`	keyword used to specify the names of all data structure that will be stored on XSM format files.
`SEQ_BINARY`	keyword used to specify the names of all data structure that will be stored on sequential binary files.
`SEQ_ASCII`	keyword used to specify the names of all data structures that will be stored on sequential ASCII files.
STRNAME	`character*12` name of a data structure. The list of data structure that can be used in DONJON is presented in Section **??**.
(module)	input specification for a module that will be executed. For DONJON specific modules, these input structures are described in Section **??** to Section **??**.
`END:`	keyword to call the normal end-of-execution utility module.
`;`	keyword to specify the end of record. This keyword is used to delimit the part of the input data stream associated with each module.

Generally, the user has the choice to declare the most of data structure in the format of a linked list to reduce CPU times or as a XSM file to reduce memory resources. In general, the data structure are stored on the sequential ASCII files only for the backup purposes.

The input data normally ends with a call to the `END:` module. However, the GAN driver will insert automatically the `END:` module, even if it was not provided, upon reaching an end-of-file in the input stream.

Each **(module)** calling specification contains a module execution description and its associated input structure. All these modules, except the `END:` module may be called more than once.

3 GENERAL CORE-DESCRIPTION MODULES

3.1 The `RESINI:` module

The `RESINI:` module is used for modeling of the reactor fuel lattice in *3-D* Cartesian geometry or *3-D* Hexagonal geometry. This modeling is based on the following considerations:

- For *3-D* Cartesian geometry, the reactor fuel lattice is composed of a well defined number of fuel channels. Each channel is composed of a well defined number of fuel bundles or assembly subdivisions. All channels contain the same number of fuel bundles or assembly subdivisions. Each reactor channel is identified by its specific name which corresponds to its position in the fuel lattice.

  In a Candu reactor, the channels are refuelled according to the bidirectional refuelling scheme. The refuelling scheme of a channel corresponds to the number of displaced fuel bundles (bundle-shift) during each channel refuelling. The direction of refuelling corresponds to the direction of coolant flow along the channel.

  In a PWR, a basic assembly layout can be projected over the fuel map using a naval-coordinate position system. Assembly refuelling and shuffling will be possible using the ad hoc module `SIM:` (see Section **??**).

- For *3-D* Hexagonal geometry, the reactor fuel lattice is composed of a well defined number of fuel channels and each channel is composed of a well defined number of fuel bundle. All fuel bundles have the same volume. All channels contain the same number of fuel bundles. Refuelling is not available during the calculation. The lattice indexation is kept to identify the hexagons.

- The fuel regions generally have a different set of global and local parameters. For example, the fuel bundles have a different evolution of the fuel properties according to the given burnup distribution, which is a global parameter. Consequently, the homogenized cell properties will differ from one fuel region to another, i.e., they are not uniform over the fuel lattice. Thus, the realistic modeling of a reactor core requires the fuel properties to be interpolated with respect to global and local parameters, which must be specified in the fuel map.

Note that the above considerations correspond to the typical core modeling of CANDU or PWR reactors. The `RESINI:` module will create a new `FMAP` object that will store the information related to the fuel lattice specification and properties (see Section **??**).
In PWR cases, each channel correspond to an assembly. Using heterogeneous mixtures in one assembly increases the complexity of the geometry. However, two levels geometries (embedded geometry) are not possible in the DONJON code. The general idea is then to define one channel per mixture for all assemblies. All these channels have then to be regrouped by assembly to impose the same burnup. This process could be done manually, but if the heterogeneity of the cross-section is large (ex. one mixture per pin within a complete core), the geometry definition may be too complex. This task can be performed automatically by the module `NAP:`.
The `RESINI:` module specifications are:

Table 2: Structure `RESINI:`

{ *FLMAP MATEX* `:=` `RESINI:` *MATEX* [*COMPO*] `::` **(descresini1)** |
FLMAP `:=` `RESINI:` *FLMAP* [*FLMAP2*] `::` **(descresini2)** }

where

FLMAP	`character*12` name of the RESINI object that will contain the fuel-lattice information. If *FLMAP* appears on both LHS and RHS, it will be updated; otherwise, it is created.
MATEX	`character*12` name of the MATEX object specified in the modification mode. *MATEX* is required only when *FLMAP* is created.
COMPO	`character*12` name of the MULTICOMPO data structure (`L_COMPO` signature) where the detailed subregion geometry at assembly level is stored.
FLMAP2	`character*12` name of the RESINI object that contains the fuel-lattice information to recover from.
(descresini1)	structure describing the main input data to the `RESINI:` module. Note that this input data is mandatory and must be specified only when *FLMAP* is created.
(descresini2)	structure describing the input data for global and local parameters. This data is permitted to be modified in the subsequent calls to the `RESINI:` module.

3.1.1 Main input data to the `RESINI:` module

Note that the input order must be respected.

Table 3: Structure **(descresini1)**

```
[ EDIT iprint ]
 ::: [ SPLIT-NAP: ] GEO: (descgeo)
 [ ::: NAP: (descnap) ]
 [ ASSEMBLY na nax nay
   A-ZONE { (iza(i) , i = 1, nch) A-NX (nbax(i) , i = 1, nay) A-IBX (ibax(i) , i = 1, nay) | ASBLY }
   AXNAME (XNAMEA(i) , i = 1, nax)
   AYNAME (YNAMEA(i) , i = 1, nay) ]
 { NXNAME (XNAME(i) , i = 1, nx) NYNAME (YNAME(i) , i = 1, ny)
           | NHNAME (HNAME(i) , i = 1, nh) }
 NCOMB { ncomb  B-ZONE (icz(i) , i = 1, nch) | ALL | ASBLY }
 [ SIM lx ly (naval(i) , i = 1, nch) ]
 (descresini2)
```

where

`EDIT`	keyword used to set *iprint*.
iprint	integer index used to control the printing on screen: = 0 for no print; = 1 for minimum printing (default value); larger values produce increasing amounts of output.
`:::`	keyword used to indicate the call to an embedded module.
`SPLIT-NAP:`	keyword to specify that the embedded geometry will be split by the embedded `NAP:` module .

`GEO:`	keyword used to call the `GEO:` module. The fuel-map geometry differs from the complete reactor geometry in the sense that it must be defined as a coarse geometry, i.e. without mesh-splitting over the fuel bundles. Consequently, the mesh-spacings over the fuel regions must correspond to the bundle dimensions (e.g. h_x=*width*; h_y=*height*; h_z=*length* or in *3-D* Hexagonal geometry h_x=*side*; h_z=*height*). Note that the total number of non-virtual regions in the embedded geometry must equal to the number of fuel channels *times* the number of fuel bundles per channel. This means that only the fuel-type mixture indices are to be provided in the data input to the `GEO:` module for `MIX` record. Other material regions (e.g. reflector) must be declared as virtual, i.e. with the mixtures indices set to 0.
(descgeo)	structure describing the input data to the `GEO:` module (see the user guide[?]). Only *3-D* Cartesian or *3-D* Hexagonal fuel-map geometry is allowed.
`NAP:`	keyword used to call the `NAP:` module. The heterogeneous assembly geometry definition will be called using the geometry defined previously with the embedded module `GEO:` and the *COMPO* data structure. See section **??** for important note on the coarse geometry requirement.
`ASSEMBLY`	keyword to specify that assembly related information are provided.
na	number of assemblies.
nax	number of assemblies along x-direction.
nay	number of assemblies along y-direction.
`A-ZONE`	keyword to specify the assembly number *iza* of each channels.
iza	assembly belonging number.
`A-NX`	keyword to specify the number of assembly *nbax* per row.
nbax	number of assembly for each row.
`A-IBX`	keyword to specify the column for the first assembly *ibax* on each row.
ibax	column number for the first assembly on each row.
`ASBLY`	(after `A-ZONE`) keyword to automatically compute the assembly number of each channel. A call to the embedded module `NAP:` is required previously.
`AXNAME`	keyword to specify the assembly position names along x-direction *XNAMEA*.
XNAMEA	`character*2` array of horizontal channel names. A horizontal channel name is identified by the channel column using numerical characters '1', '2', '3', and so on. Note that the total number of X-names must equal to *nxa*.
`AYNAME`	keyword to specify the assembly position names along y-direction *YNAMEA*.
YNAMEA	`character*2` array of horizontal channel names. A horizontal channel name is identified by the channel column using numerical characters 'A', 'B', 'C', and so on. Note that the total number of Y-names must equal to *nya*.
`NXNAME`	keyword used to specify *XNAME* for *3-D* Cartesian geometry case.
XNAME	`character*2` array of horizontal channel names. A horizontal channel name is identified by the channel column using numerical characters '1', '2', '3', and so on. Note that the total number of X-names must equal to the total number of subdivisions along the X-direction in the fuel-map geometry. All non-fuel regions are to be assigned a single character '-'. This option is not available for *3-D* Hexagonal geometry. When assembly are defined and split, several names can be the same.

nx	integer total number of subdivisions along the X-direction in the fuel-map geometry. Not used for *3-D* hexagonal geometry.
`NYNAME`	keyword used to specify *YNAME* for *3-D* Cartesian geometry case.
YNAME	`character*2` array of vertical channel names. A vertical channel name is identified by the channel row using alphabetical letters 'A' (from the top), 'B', 'C', and so on. The total number of Y-names must equal to the total number of subdivisions along the Y-direction in the fuel-map geometry. All non-fuel regions are to be assigned a single character '-'. This option is not available for *3-D* Hexagonal geometry. When assembly are defined and split, several names can be the same.
ny	integer total number of subdivisions along the Y-direction in the fuel-map geometry. Not used for *3-D* hexagonal geometry.
`NHNAME`	keyword used to specify *XHAME* for *3-D* hexagonal geometry case.
HNAME	`character*8` array of horizontal channel names. A radial channel name can be identified by the following scheme: The core is divided into 6 60-degree sectors; the sectors are labeled "A", "B", "C", "D", "E", and "F" • Rings of channels are numbered starting at ring 0 for the central channel • Each channel is now identified as a string 'RRSAA', with RR the ring number, S the sector, and AA the assembly number in the ring. For example: **Ring 0:** `C00A01` **Ring 1:** `C01A01`, `C01B01`, `C01C01`, `C01D01`, `C01E01`, `C01F01` **Ring 2:** `C02A01`, `C02A02`, `C02B01`, `C02B02`, `C02C01`, `C02C02`, `C02D01`, `C02D02`, `C02E01`, `C02E02`, `C02F01`, `C02F02`. All non-fuel regions are to be assigned a single character '`-`'. This option is not available for *3-D* Cartesian geometry.
nh	integer total number of subdivisions along the hexagonal plane in the fuel-map geometry. Not used for *3-D* Cartesian geometry.
`NCOMB`	keyword used to specify the number of combustion zones.
ncomb	integer total number of combustion zones. This value must be greater than (or equal to) 1 and less than (or equal to) the total number of reactor channels.
`B-ZONE`	keyword used to specify *icz*.
icz	integer array of combustion-zone indices, specified for every channel. A reactor channel can belong to only one combustion zone, however a combustion zone can be specified for several channels.
`ALL`	keyword used to indicate that the total number of combustion zones equals to the number of reactor channels. In this particular case, each channel will have a unique combustion-zone number. Hence, an explicit specification of the combustion-zone indices can be omitted.
nch	$N_{\rm ch}$: number of fuel channels in the radial plane.
nb	$N_{\rm b}$: number of fuel bundles or assembly subdivisions in the axial plane.

ASBLY	(after NCOMB) keyword to specify that one combustion zone per assembly is to be defined.
SIM	keyword used to specify a basic assembly layout for the SIM: PWR refuelling module (see Section **??**).
lx	number of assemblies along the X axis. Typical values are 15 or 17.
ly	number of assemblies along the Y axis.
naval	character*3 identification name corresponding to the basic naval-coordinate position of an assembly. *naval*(i) is the concatenation of a letter (generally chosen between A and T) and of an integer (generally chosen between 01 and 17). An assembly may occupies four positions in the fuel map in order to be represented by four radial burnups. In this case, the same naval-coordinate value will appear at four different (i) indices.

3.1.2 Input of global and local parameters

The information with respect to the fuel burnup is required for the fuel-map MACROLIB construction, using either the CRE:, NCR: or AFM: module. The fuel-region properties related to other local or global parameters can be interpolated only using the NCR: module.

Table 4: Structure **(descresini2)**

```
[ EDIT iprint ]
[ BTYPE { TIMAV-BURN | INST-BURN } ]
[ TIMAV-BVAL (bvalue(i) , i = 1, ncomb ) ]
[ INST-BVAL { SAME bvalue | CHAN (bvalue(i) , i = 1, nch ) | BUND (bvalue(i) , i = 1, nch·nb ) |
  SMOOTH } | ASBLY (bvalue(i) , i = 1, na ) | OLDMAP ]
[ BUNDLE-POW { SAME pwvalue | CHAN (pwvalue(i) , i = 1, nch ) | BUND (pwvalue(i) , i = 1, nch·nb ) } ]
[ REF-SHIFT { ishift | COMB (ishift(i) , i = 1, ncomb ) } ]
[[ ADD-PARAM  PNAME PNAME  PARKEY PARKEY { GLOBAL | LOCAL } ]]
[[ SET-PARAM PNAME { pvalue | OLDMAP | { [ TIMES PNAMEREF ] SAME pvalue |
   CHAN (pvalue(i) , i = 1, nch ) | BUND (pvalue(i) , i = 1, nch·nb ) } } ]]
[[ FUEL { WEIGHT | ENRICH | POISON } (fvalue(i) , i = 1, nfuel ) ]]
[ CELL (ialch(i) , i = 1, nch ) ]
;
```

where

EDIT	keyword used to set *iprint*.
iprint	integer index used to control the printing on screen: = 0 for no print; = 1 for minimum printing (default value); = 2 to print the channels refuelling schemes (if they are new or modified); = 3 initial burnup limits per each channel are also printed (if the axial power-shape has been reinitialized).
BTYPE	keyword used to specify the type of interpolation with respect to burnup data. This information will be used during the execution of CRE:, NCR: or AFM: module.

`TIMAV-BURN`	keyword used to indicate the burnups interpolation according to the time-average model. This option is not available in *3-D* Hexagonal geometry.
`INST-BURN`	keyword used to indicate the burnups interpolation according to the instantaneous model.
`TIMAV-BVAL`	keyword used to indicate the input of average exit burnup values per each combustion zone. Note that the axial power-shape and the first burnup limits will be reinitialized each time the average exit burnups are modified by the user. These data are required for the time-average calculation (see Section **??**). This option is not available with *3-D* Hexagonal geometry.
`INST-BVAL`	keyword used to specify the instantaneous burnup values for each fuel bundle.
`SMOOTH`	keyword used to level fuel mixtures burnup. If the burnup is supposed to be the same at each occurence of every fuel mixture (for symetry reasons), `SMOOTH` will make sure they share the exact same value (the first one in the burnup map). Purpose is only to correct calculation noise in historic calculation.
`ASBLY`	keyword to specify that one burnup value per assembly is to be defined.
`OLDMAP`	keyword to specify that the burnup value is recovered from *FLMAP2*. The recovered burnup distribution is either from a previous calculation: • with the same geometry but different initialization values. Example: homogeneous calculation followed by a pin power reconstruction where assemblies were not defined in the first place. • with a different geometry. In this case, the assembly geometry of the new *FLMAP* and the geometry of the *FLMAP2* must match. Example: homogeneous calculation followed by a heterogeneous calculation or pin power reconstruction
`BUNDLE-POW`	keyword used to specify the power values for each fuel bundle. This option is not available in *3-D* Hexagonal geometry.
bvalue	real array containing the burnups values, given in *MW·day per tonne*/MW of initial heavy elements. The fuel burnup is considered as a global parameter.
pwvalue	real array containing the powers values, given in kW.
`REF-SHIFT`	keyword used to specify *ishift*. Note that the axial power-shape and the first burnup limits will be reinitialized each time the channel refuelling schemes are modified by the user. This option is not avaialble in *3-D* Hexagonal geometry.
`COMB`	keyword used to indicate the input of bundle-shift numbers per combustion zone.
ishift	integer array (or single value) of the bundle-shift numbers. A single *ishift* value means that the same bundle-shift will be applied for all combustion zones. Note that the bundle-shift value must be positive, it corresponds to the number of displaced fuel bundles during each channel refuelling.
`ADD-PARAM`	keyword used to indicate the input of information for a new global or local parameter. For more information about the parameter data organization on `FMAP` data structure see Section **??**.
`PNAME`	keyword used to specify *PNAME*.

PNAME `character*12` identification name of a given parameter. This name is user-defined so that it is arbitrary, however it must be unique so that it can be used for the search of parameter information and interpolation purpose. Moreover, it is recommended to use the following pre-defined values:

`C-BORE`	Boron concentration
`T-FUEL`	Averaged fuel temperature
`T-SURF`	Surfacic fuel temperature
`T-COOL`	Averaged coolant temperature
`D-COOL`	Averaged coolant density
CANDU-only parameters:	
`T-MODE`	Averaged moderator temperature
`D-MODE`	Averaged moderator density

`PARKEY` keyword used to specify *PARKEY*.

PARKEY `character*12` corresponding name of a given parameter as it is recorded in the particular multi-parameter compo file. The *PARKEY* name of a parameter may not be same as its *PNAME* and can also differ from one multi-compo file to another.

`GLOBAL` keyword used to indicate that a given parameter is global, which will have a single and constant parameter's value.

`LOCAL` keyword used to indicate that a given parameter is local. In this case, the total number of recorded parameter's values will be set to $N_{\rm ch} \times N_{\rm b}$.

`SET-PARAM` keyword used to indicate the input (or modification) of the actual values for a parameter specified using its *PNAME*.

`SAME` keyword used to indicate that a core-average value of a local parameter will be provided. If the keyword `SAME` is specified, then this average value will be set for all fuel bundles for every reactor channel.

`CHAN` keyword used to indicate that the values of a local parameter will be provided per each reactor channel. If the keyword `CHAN` is specified, then the channel-averaged parameter's value will be set for all fuel bundles containing in the same reactor channel.

`BUND` keyword used to indicate that the values of a local parameter will be specified per each fuel bundle for every channel.

`TIMES` keyword used to indicate that the values of the local parameter *PNAME* is a translation of the local parameter *PNAMEREF* via a multiplication of the constant indicated by `SAME`.

PNAMEREF `character*12` identification name of a given parameter.

pvalue real array (or a single value) containing the actual parameter's values. Note that these values will not be checked for consistency by the module. It is the user responsibility to provide the valid parameter's values which should be consistent with those recorded in the multicompo database.

`OLDMAP` keyword to specify that the *pvalue* value(s) is (are) recovered from *FLMAP2*.

`FUEL` keyword used to indicate the input of data which will be specified for each fuel type.

`WEIGHT` keyword used to indicate the input of fuel weight in a bundle, given in *kg*.

`ENRICH` keyword used to indicate the input of fuel enrichment values, given in *wt*%.

`POISON` keyword used to indicate the input of poison load in a fuel.

fvalue	real value of the fuel-type parameter, specified for each fuel type in the same order as the fuel mixture indices have been recorded in the MATEX object (see Section **??**).
nfuel	integer total number of the fuel types, as been defined in the `USPLIT:` module.
`CELL`	keyword used to specify that a patterned age distribution will be input and used to compute instantaneous bundle burnup.
ialch	real array containing the refueling sequence numbers. This channel is refueled the *ialch(i)*th one. The channels are ordering from the top left to the bottom right of the core. The expression of the resulting bundle burnups are given in Ref. **?**.

3.2 The `USPLIT:` module

The `USPLIT:` module is used to create a `MATEX` object that will provide a link between the reactor geometry and material index. The *3-D* Cartesian or *3-D* Hexagonal reactor geometry, which is previously produced in the `GEO:` module, is analyzed and the material mixture indices are recomputed in order to provide a unique mixture number for each material sub-volume. Such renumbering permits a complex reactor core modeling. A `MATEX` object is also used to store some additional information that will be required and updated by other DONJON modules (see Section **??**).

The `USPLIT:` module specification is:

Table 5: Structure `USPLIT:`

GEOM MATEX `:=` `USPLIT:` { *GEOM* | *GEOMOLD* } `::` **(desclink)**

where

GEOM	`character*12` name of a GEOMETRY object. This object is defined in creation (appears only on LHS) or modification (appears on both LHS and RHS) mode. An existing geometry previously created in the `GEO:` module is modified. Only *3-D* Cartesian or *3-D* Hexagonal reactor geometries are allowed.
MATEX	`character*12` name of a MATEX object to be created by the module.
GEOMOLD	`character*12` name of a GEOMETRY object previously created in the `GEO:` module. This object must be specified in read-only mode (appears only on RHS). It is copied into *GEOM* at the beginning of `USPLIT:` module. Only *3-D* Cartesian or *3-D* Hexagonal reactor geometries are allowed.
(desclink)	structure describing the input data to the `USPLIT:` module.

3.2.1 Input data to the `USPLIT:` module

Note that the fuel-type and reflector-type mixture indices are need to be specified explicitly and the input order must be respected.

Table 6: Structure **(desclink)**

[`EDIT` *iprint*]
`NGRP` *ngrp*
`MAXR` *maxreg*
`NMIX` *nmixt*
[`NREFL` *nrefl* `RMIX` (*mixr*(i) , i = 1, *nrefl*)]
[`NFUEL` { *nfuel* `FMIX` (*mixf*(i) , i = 1, *nfuel*) | `ASBLY` }]
;

where

`EDIT`	keyword used to set *iprint*.
iprint	integer index used to control the printing on screen: = 0 for no print; = 1 for minimum printing (default value); larger values produce increasing amounts of output.
`NGRP`	keyword used to specify *ngrp*.
ngrp	integer total number of energy groups. This value must be greater than 0.
`MAXR`	keyword used to specify *maxreg*.
maxreg	integer maximum number of mesh-splitted regions in the reactor geometry. In *3-D* Hezagonal geometry, it corresponds to the total number of prismatic blocks l_h*l_z.
`NMIX`	keyword used to extend number of material mixtures in case new fuels are going to be inserted in the fuel map in upcoming fuel cycles. By default, *nmixt* is set to the maximum mixture index in RHS geometry *GEOM* or *GEOMOLD*.
nmixt	the maximum fuel mixture index in the complete life of the reactor. This number must be greater than the maximum mixture index in RHS geometry *GEOM* or *GEOMOLD*.
`NREFL`	keyword used to specify *nrefl*.
nrefl	integer total number of reflector types. A reactor should have at least one reflector material.
`RMIX`	keyword used to specify *mixr*.
mixr	integer array of the reflector-type mixture indices. Each reflector type is assigned a distinct mixture number as previously defined in the GEOMETRY object.
`NFUEL`	keyword used to specify *nfuel*.
nfuel	integer total number of fuel types. A reactor should have at least one fuel type.
`FMIX`	keyword used to specify *mixf*.
mixf	integer array of the fuel-type mixture indices. Each fuel type is assigned a distinct mixture number as previously defined in the GEOMETRY object.
`ASBLY`	keyword used to compute automatically *nfuel* and *mixf*(i). This option is only available when the geometry has been split by the `NAP:` module.

3.3 The `MACINI:` module

The `MACINI:` module is used to construct an extended MACROLIB, in which the properties are stored per each material region over the whole mesh-splitted reactor geometry. This MACROLIB is obtained by combining the material properties which are contained in the two distinct MACROLIB objects:

- The first MACROLIB contains the material properties which are evolution-independent, such as reflector and device properties. It is created using either `MAC:`, `CRE:`, `NCR:` or `AFM:` module.
- The second is a fuel-map MACROLIB created using either `CRE:`, `NCR:` or `AFM:` module. It must contain the interpolated fuel properties per each fuel bundle.

The resulting MACROLIB will contain the properties that are stored for each reactor material and per each mesh-splitted volume. When the devices are not present in the reactor core, then the resulting MACROLIB can be considered as a complete reactor MACROLIB and it can be directly used for the numerical solving. However, when the devices are inserted into the reactor core, the resulting MACROLIB is not yet complete; it must be subsequently updated with respect to the device properties, using the `NEWMAC:` module (see Section **??**).

The `MACINI:` module specification is:

Table 7: Structure `MACINI:`

MACRO2 MATEX `:=` `MACINI:` *MATEX MACRO* [*MACFL*] `::` [`EDIT` *iprint*] ;

where

MACRO2	`character*12` name of the extended MACROLIB to be created by the module.
MATEX	`character*12` name of the MATEX object containing an extended material index over the reactor geometry. *MATEX* must be specified in the modification mode; it will store the recovered h-factors per each fuel region.
MACRO	`character*12` name of a MACROLIB, created using either `MAC:`, `CRE:`, `NCR:` or `AFM:` module, for the evolution-independent material properties (see structure **(desccre1)** or refer to the user guide[?]).
MACFL	`character*12` name of a fuel-map MACROLIB, created using either `CRE:`, `NCR:` or `AFM:` module, for the interpolated fuel properties (see structure **(desccre2)** or refer to the user guide[?]).
`EDIT`	keyword used to set *iprint*.
iprint	integer index used to control the printing on screen: = 0 for no print; = 1 for minimum printing; larger values produce increasing amounts of output. The default value is *iprint* = 1.

3.4 The `DEVINI:` module

The `DEVINI:` module is used for the modeling of reactivity mecanisms, based on the devices specifications which are read from the input data file. The module will create a new DEVICE object that will store the devices specifications and parameters (see Section **??**). Note that only the rod-type (i.e. solid) devices are considered using the `DEVINI:` module; the liquid zone controllers can be added subsequently, using the `LZC:` module (see Section **??**). A rod-type device is a reactivity controller rod (or plate), such as: a zone control rod (ZCR), a shutoff rod (SOR), etc. Several devices parameters can be modified using the `DSET:` module (see Section **??**).

A device specification includes several controller rod parameters, such as: a rod position, rod insertion level, direction of movement, etc. The devices positions can not overlap in the reactor core; they are referred using *3-D*-Cartesian coordinates. The insertion level of rods can be set according to their nominal positions or they can be displaced *in* or *out* of core. The rods can also be divided into the several user-defined groups so that they can be manipulated, displaced or moved simultaneously.

The `DEVINI:` module specification is:

Table 8: Structure `DEVINI:`

DEVICE MATEX `:=` `DEVINI:` *MATEX* `::` **(descdev)**

where

DEVICE	`character*12` name of the DEVICE object that will be created by the module; it will contain the devices information.
MATEX	`character*12` name of the MATEX object that will be updated by the module. The rod-devices material mixtures are appended to the previous material index and the rod-devices indices are also modified, accordingly.
(descdev)	structure describing the input data to the `DEVINI:` module.

3.4.1 Input data to the `DEVINI:` module

The `DEVINI:` module allows the definition of rod-type devices made of one or many (up to 10) parts, as depicted in Fig. **??**.

Table 9: Structure **(descdev)**

[`EDIT` *iprint*]
`NUM-ROD` *nrod* [{ `FADE` | `MOVE` }]
(**(dev-rod)**, i = 1, *nrod*)
[`CREATE` `ROD-GR` *ngrp* (**(rod-group)**, i = 1, *ngrp*)]
;

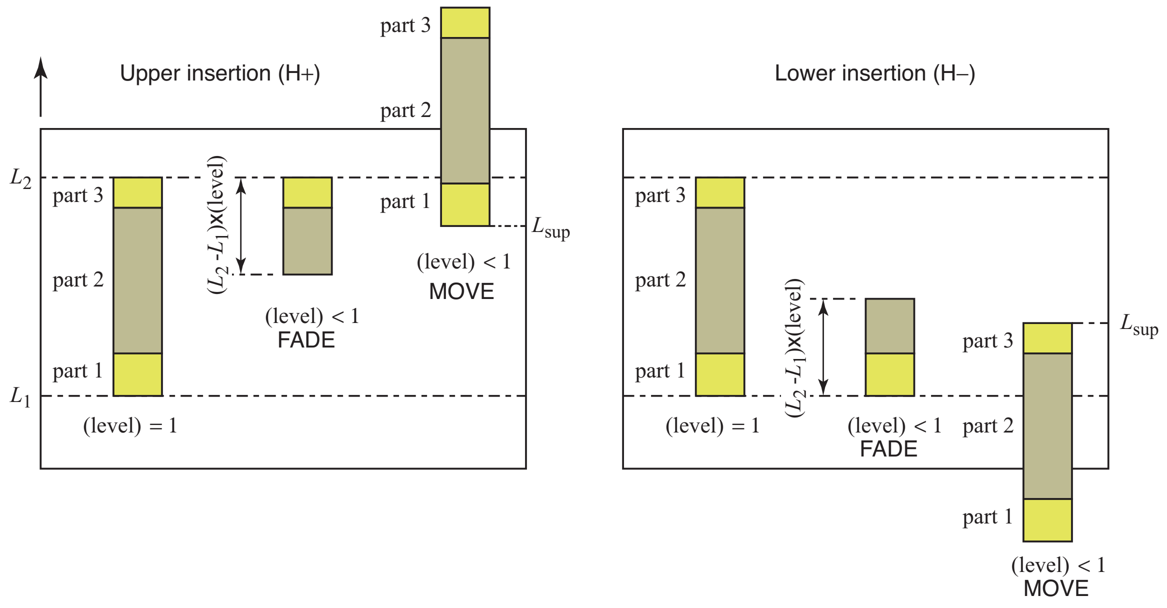

Figure 1: Presentation of fully- and partially-inserted 3-part control rods.

where

`EDIT`	keyword used to set *iprint*.
iprint	integer index used to control the printing on screen: = 0 for no print; = 1 for minimum printing (default value); larger values produce increasing amounts of output.
`NUM-ROD`	keyword used to specify *nrod*.
nrod	integer total number of the reactor rod-type devices. This number must be greater than 0.
`FADE`	fading rod keyword. A fraction of the fully inserted rod vanishes (default option).
`MOVE`	moving rod keyword. The complete rod is moving (DONJON3-type movement).
`CREATE`	keyword used to create the rod-groups of devices. The creation of groups is optional.
`ROD-GR`	keyword used to set *ngrp*.
ngrp	integer total number of the rod groups to be created. This number must be greater than 0.
(dev-rod)	structure describing the input data for each individual rod.
(rod-group)	structure describing the input data for each group of rods.

3.4.2 Description of dev-rod input structure

A rod position is referred by its *3-D* Cartesian coordinates only. Note that the devices positions can not overlap. The input order of data must be respected.

Table 10: Structure **(dev-rod)**

```
ROD id
  ROD-NAME NAME
  AXIS { X | Y | Z }
  FROM { H+ | H- }
  [ LEVEL value ]
  [ SPEED speed ]
  [ TIME time ]
  [[ MAXPOS (pos(i), i = 1, 6) DMIX mix1 mix2 ]]
ENDROD
```

where

`ROD`	keyword used to specify the rod *id* number.
id	integer identification number of the current rod. Each rod-type device must be assigned a unique *id* number, given in an ascending order ranging from 1 to *nrod*.
`ROD-NAME`	keyword used to specify the rod *NAME*.
NAME	`character*12` name of the current rod. In general, this name is composed by the rod specific type (e.g. SOR, ZCR, etc.) followed by its sequential number (e.g. 01, 02, etc.).
`AXIS`	keyword used to specify the rod movement axis. A rod can be displaced along only one of the axis.
`X`	keyword used to specify that a rod is displaced along X axis.
`Y`	keyword used to specify that a rod is displaced along Y axis.
`Z`	keyword used to specify that a rod is displaced along Z axis.
`FROM`	keyword used to specify the insertion side of geometry. The rod-devices can be inserted into the reactor core from only one side of geometry. For example, some vertically moving devices can be inserted only from the top, whereas other only from the bottom.
`H+`	keyword used to specify that a rod will be inserted into reactor core from the highest position (e.g. from the top for vertically moving rod-device).
`H-`	keyword used to specify that a rod will be inserted into reactor core from the lowest position (e.g. from the bottom for vertically moving rod-device).
`LEVEL`	keyword used to specify the actual rod insertion level *value*. By default, the rod insertion level is left undefined.
value	real positive value of the rod insertion level. This value is used to compute the actual rod position in the reactor core. The rod insertion level is minimal ($value = 0.0$) when the rod is completely withdrawn, and it is maximal ($value = 1.0$) when the rod is fully inserted. For the partially inserted rod the insertion level must be: $0.0 < value < 1.0$
`SPEED`	keyword used to specify *speed*. By default, the speed is left undefined.
speed	real positive value of the rod movement speed, given in cm/s. This value is needed only for the reactor regulating purpose.

`TIME`	keyword used to specify *time*. By default, the insertion time is left undefined.
time	real value of time for the rod insertion (or extraction), given in sec. This value is needed only for the reactor regulating purpose.
`MAXPOS`	keyword used to specify the full-inserted coordinates of a rod part. The sequence of `MAXPOS` and `DMIX` data structures is repeated for each part making the rod.
pos	real array containing 3-D Cartesian coordinates of the full-inserted rod. This is the limiting rod position in the reactor core, which may or may not be the same as the actual rod position. These coordinates must be given in the order: X−, X+, Y−, Y+, Z−, and Z+.
`DMIX`	keyword used to specify *mix1* and *mix2*.
mix1	first of two integer rod mixture indices. Index *mix1* corresponds to the perturbed cross sections.
mix2	second of two integer rod mixture indices. Index *mix2* corresponds to the reference cross sections. Indices *mix1* and *mix2* will be used to compute the incremental cross sections in the `NEWMAC:` module.
`ENDROD`	keyword used to end the rod description.

3.4.3 Description of rod-group input structure

The partition of devices into groups is very useful when the same action is to be applied to several rods, e.g. setting of new parameters (using the `DSET:` module) or rods moving (using the `MOVDEV:` module).

Table 11: Structure **(rod-group)**

`GROUP-ID` *igrp* { `ROD-ID` [[*id*]] | `ALL` }

where

`GROUP-ID`	keyword used to set *igrp* number.
igrp	integer identification number of a group to be created. Each rods group must be assigned a unique identification number, given in ascending order ranging from 1 to *ngrp*.
`ROD-ID`	keyword used to set the rod *id* numbers.
id	integer identification numbers of rods which belong to the same group *igrp*. A particular rod (or several rods) may belong to different groups, but it could not be repeated inside the same group. The total number of rods in any group must be between 1 and *nrod*.
`ALL`	keyword used to specify that all rods will belong to the same group *igrp*.

3.5 The `DETINI:` module

The `DETINI:` module is used to read and store detector information. A detector is represented by a 2-D or 3-D Cartesian/Hexagonal geometry.

The `DETINI:` module specification is:

Table 12: Structure `DETINI:`

DETECT `:=` `DETINI:` [*DETECT*] `::` **(descdet)**

where

DETECT	`character*12` name of the DETECT object that will be created by the module; it will contain the detector informations. If DETECT appear on RHS, it is updated, otherwise, it is created.
(descdev)	structure describing the input data to the `DETINI:` module.

3.5.1 Input data to the `DETINI:` module

Note that the input order must be respected.

Table 13: Structure **(descinidet)**

[`EDIT` *iprt*] [HEXZ] `NGRP` *ngrp*
[[`TYPE` *NAMTYP*
`INFO` *ndetect nrep* { `SPECTRAL` (*spec*(i), i=1,*ngrp*) | `DEFAULT` }
[`INVCONST` (*tinv*(i), i=1,*nrep*−2)] [`FRACTION` (*fract*(i), i=1,*nrep*−1)]
(**(descdet)**, i=1,*ndetect*)]]
;

where

`EDIT`	keyword used to set *iprt*.
iprt	index used to control the printing in module `INIDET:`. =1,2 for no print(default value); =3 for printing the contents of the output DETECT.
`HEXZ`	keyword to specify that only hexagonal detectors will be defined. If this keyword is absent, Cartesian detectors will be defined.
`NGRP`	keyword used to set *ngrp*.
ngrp	number of energy groups in the calculation. It must be equal to the number set in the `MACD:` module or by the COMPO files.

`TYPE`	keyword to specify the detector type.
NAMTYP	`character*12` name of the detector type. To correspond to the actual detector response model encoded, the type of detector must be in this list: • `PLATN_REGUL` • `PLATN_SAU` • `VANAD_REGUL` • `CHION_SAU` • `CHION_REGUL` For other type names, only a fixed normalisation can be performed.
`INFO`	keyword to specify the information associated with the detector type.
ndetect	number of detectors of the specified type.
nrep	number of detector response components for the specified type. It must be greater or equal to 2, corresponding to a response in fraction and the reference flux value.
`SPECTRAL`	keyword to specify the energy spectral of a detector type.
spec	array containing the energy spectral of a detector type.
`DEFAULT`	keyword to specify the energy spectral will be initialized as 1.0 for the highest energy group and 0.0 for other groups.
`INVCONST`	keyword to specify the inverse time constants of the detector type model. This option is only valid for platinum, (*NAMTYP*(1:5) = 'PLATN'), detector type.
tinv	array containing the inverse time constants of the detector model.
`FRACTION`	keyword to specify the fractions corresponding to each delayed or prompt reponse of the detector type model. This option is only valid for platinum, (*NAMTYP*(1:5) = 'PLATN'), detector type.
frac	array containing the detector type model fractions.
(descdet)	structure describing the format used to read detector information.

3.5.2 Description of the detector data

Note that the information input order must be respected.

Table 14: Structure **(descdet)**

```
NAME NAMDET
[ NHEX nhex HEX ( ihex(i), i=1,nhex ) ]
POSITION ( pos(i), i=1,6 )
RESP ( rep(i), i=1,nrep )
ENDN
```

where

`NAME`	keyword to specify the detector name.
NAMDET	`character*12` name of the detector. The different names in alphabetical order must fit their usual numbering in the core.(Ex: PLATN01, CHION01C)
`NHEX`	keyword to set the number of hexagons where the detector is placed.
nhex	number of hexagons.
`HEX`	keyword to set the hexagon numbers corresponding to the detector position.
ihex	array containing the hexagon numbers where the detector is present, as ordered in the geometry definition.
`POSITION`	keyword to specify the detector coordinates.
pos	array containing the positions of the specified detector. The positions must be read as X− X+ Y− Y+ Z− Z+ . For 2-D geometry, Z coordinates must be 0.0 and a value greater than 1.0. For hexagonal geometry, only Z coordinates are used in 3-D representation.
`RESP`	keyword to specify the detector initial responses.
rep	array containing the initial responses of the detector. To use the current detector models in DONJON, responses are given as • For vanadium detectors: current response, last response. • For platinum detectors: current response, reference flux, last detector slow responses. • For ion chamber detectors: current logarithmic response, current log rate response, reference flux.
`ENDN`	keyword to specify the end of the detector informations.

3.6 The `LZC:` module

The `LZC:` module is used for the modeling of liquid zone controllers, which are normally presented in the CANDU6-type reactor core. The liquid zone controllers specifications are read from the input data file. Note that this modeling can be made after the rod-type devices have been previously defined using the `DEVINI:` module (see Section **??**). In this case, the previously created DEVICE object will be updated by the `LZC:` module; it will store the additional and separate information with respect to the liquid controllers (see Section **??**).

The liquid zone controller specification includes several device parameters, such as: the whole device position, water filling level, direction of filling, etc. Note that a liquid zone controller is normally composed of two parts: one part is empty and the second part is full-filled. The water level can be adjusted according to the control reactivity requirements. The controllers positions are referred using *3-D*-Cartesian coordinates. Several devices parameters can be modified using the `DSET:` module (see Section **??**). The liquid controllers can also be divided into the several user-defined groups so that they can be manipulated simultaneously.

The `LZC:` module specification is:

Table 15: Structure `LZC:`

DEVICE MATEX `:=` `LZC:` [*DEVICE*] *MATEX* `::` **(desclzc)**

where

DEVICE	`character*12` name of the DEVICE object. Note, if the rod-type devices are not present in the reactor core, then *DEVICE* object must appear only on the LHS (i.e. in create mode), it will contain the information only with respect to the liquid zone controllers. However, if the rod-type devices are present in the reactor core, then they must be specified first (i.e. before the liquid controllers) using the `DEVINI:` module (see Section **??**). In the last case, the *DEVICE* object must also appear on the RHS (i.e. in modification mode), it will contain the additional and separate information with respect to the liquid zone controllers.
MATEX	`character*12` name of the MATEX object that will be updated by the module. The lzc-devices material mixtures are appended to the previous material index and the lzc-devices indices are also modified, accordingly.
(desclzc)	structure describing the input data to the `LZC:` module.

3.6.1 Input data to the `LZC:` module

Note that the input order must be respected.

Table 16: Structure **(desclzc)**

```
[ EDIT iprint ]
NUM-LZC nlzc
((dev-lzc), i = 1, nlzc)
[ CREATE LZC-GR ngrp ((lzc-group), i = 1, ngrp) ]
;
```

where

`EDIT`	keyword used to set *iprint*.
iprint	integer index used to control the printing on screen: = 0 for no print; = 1 for minimum printing (default value); larger values produce increasing amounts of output.
`NUM-LZC`	keyword used to specify *nlzc*.
nlzc	integer total number of liquid zone controllers. This number must be greater than 0.
`CREATE`	keyword used to create the lzc-groups of devices. The creation of groups is optional.
`LZC-GR`	keyword used to set *ngrp*.
ngrp	integer total number of the lzc groups to be created. This number must be greater than 0.
(dev-lzc)	structure describing the input data for each individual liquid controller.
(lzc-group)	structure describing the input data for each group of liquid controllers.

3.6.2 Description of dev-lzc input structure

Note that the devices positions can not overlap in the reactor core. The input order of data must be respected.

Table 17: Structure **(dev-lzc)**

```
LZC id
MAXPOS ( pos(i) , i = 1, 6 )
MAX-FULL fmax
AXIS { X | Y | Z }
LEVEL value
[ RATE rate ]
[ TIME time ]
EMPTY-MIX ( mixE(n), n = 1, 2 )
FULL-MIX ( mixF(n), n = 1, 2 )
```

where

`LZC`	keyword used to specify the liquid controller *id* number.
id	integer identification number of the current liquid controller. Each controller must be assigned a unique *id* number, given in an ascending order ranging from 1 to *nlzc*.
`MAXPOS`	keyword used to specify the entire position of a liquid zone controller, including its empty and full parts.
pos	real array containing 3-D Cartesian coordinates of the liquid zone controller position in the reactor core. These coordinates must be given in the order: X− X+ Y− Y+ Z− Z+
`MAX-FULL`	keyword used to specify *fmax*.
fmax	real value of the limiting coordinate along the controller filling axis, which corresponds to the maximum full-filling level for the current liquid controller.
`AXIS`	keyword used to specify the controller filling axis. A liquid controller can be filled along only one (vertical) axis.
`X`	keyword used to specify that a liquid controller is filled along X axis.
`Y`	keyword used to specify that a liquid controller is filled along Y axis.
`Z`	keyword used to specify that a liquid controller is filled along Z axis.
`LEVEL`	keyword used to specify the actual filling level.
value	real positive value of the water level. This value is minimal ($value = 0.0$) when the controller is empty, and it is maximal ($value = 1.0$) when the controller is full-filled. For the partially filled controller the water level must be: $0.0 < value < 1.0$
`RATE`	keyword used to specify *rate*.
rate	real positive value of the water filling rate, given in m^3/s. This value is needed only for the reactor regulating purpose.
`TIME`	keyword used to specify *time*.
time	real value of the filling time, given in sec. This value is needed only for the reactor regulating purpose.
`EMPTY-MIX`	keyword used to specify *mixE*.
mixE	two integer mixture indices, specified for the empty-part of liquid controller. The first and the second mixture indices correspond to the perturbed and the reference cross sections, respectively. These indices will be used to compute the incremental cross sections in the `NEWMAC:` module.
`FULL-MIX`	keyword used to specify *mixF*.
mixF	two integer mixture indices, specified for the full-part of liquid controller. The first and the second mixture indices correspond to the perturbed and the reference cross sections, respectively. These indices will be used to compute the incremental cross sections in the `NEWMAC:` module.

3.6.3 Description of lzc-group input structure

The partition of lzc-devices into groups is similar to that of rod-devices.

Table 18: Structure **(lzc-group)**

```
GROUP-ID igrp { LZC-ID [[ id ]] | ALL }
```

where

`GROUP-ID`	keyword used to set *igrp* number.
igrp	integer identification number of a group to be created. Each controllers group must be assigned a unique identification number, given in ascending order ranging from 1 to *ngrp*.
`LZC-ID`	keyword used to set the controllers *id* numbers.
id	integer identification numbers of the liquid controllers which belong to the same group *igrp*. A particular controller (or several devices) may belong to different groups, but it could not be repeated inside the same group. The total number of liquid controllers in any group must be between 1 and *nlzc*.
`ALL`	keyword used to specify that all liquid controllers will belong to the same group *igrp*.

3.7 The `DSET:` module

The `DSET:` module is used to set or to update some of the devices parameters. The new parameters can be applied for the rod-type devices and/or for the liquid zone controllers, such as: the new insertion level for the rods or water filling level for the lzc-type devices, etc. It is possible to apply the new parameters to the individual user-selected devices as well as to the user-selected groups of devices. If the device (rod-insertion or lzc-filling) level is selected for the modification, then a new device position is recomputed accordingly. The `DSET:` module can be used to perform the device reactivity studies and also to predict the reactivity worth of the rod-devices.

The `DSET:` module specification is:

Table 19: Structure `DSET:`

DEVICE `:=` `DSET:` *DEVICE* `::` **(descdset)**

where

DEVICE	`character*12` name of the DEVICE object that will be updated by the module.
(descdset)	structure describing the input data to the `DSET:` module.

3.7.1 Input data to the `DSET:` module

It is possible to set or to modify the parameters for several individual devices and/or for several groups of devices simultaneously.

Table 20: Structure **(descdset)**

`EDIT` *iprint*
[[{ `ROD` *irod* | `ROD-GROUP` *irgrp* | `LZC` *ilzc* | `LZC-GROUP` *ilgrp* }
[`LEVEL` *value*] [`SPEED` *speed*] [`TIME` *time*]
`END`]]
;

where

`EDIT`	keyword used to set *iprint*.
iprint	integer index used to control the printing on screen: = 0 for no print; = 1 for minimum printing; larger values produce increasing amounts of output.
`ROD`	keyword used to specify the rod *irod* number.
irod	integer identification number of a rod to be modified. Each rod-type device has a

	unique *irod* number, ranging from 1 to *nrod*, as been defined in the `DEVINI:` module (see Section **??**).
`ROD-GROUP`	keyword used to specify the rod-group *irgrp* number.
irgrp	integer identification number of a rod-group of devices that will be modifed with the same parameters. Each rod-group has a unique *irgrp* number, ranging from 1 to *ngrp*, as been defined in the `DEVINI:` module (see Section **??**).
`LZC`	keyword used to specify the liquid controller *ilzc* number.
ilzc	integer identification number of a liquid controller to be modified. Each lzc-type device has a unique *ilzc* number, ranging from 1 to *nlzc*, as been defined in the `LZC:` module (see Section **??**).
`LZC-GROUP`	keyword used to specify the lzc-group *ilgrp* number.
ilgrp	integer identification number of a lzc-group of devices that will be modifed with the same parameters. Each lzc-group has a unique *ilgrp* number, ranging from 1 to *ngrp*, as been defined in the `LZC:` module (see Section **??**).
`LEVEL`	keyword used to specify a new level *value*.
value	real positive value of the new device level. For the rod-type devices this *value* must correspond to the new rod insertion level (see Section **??**). For the lzc-type devices this *value* must correspond to the new water filling level (see Section **??**). In any case, the new level value must be: $0.0 \leq value \leq 1.0$
`SPEED`	keyword used to specify a new value for *speed*.
speed	real positive value of the device speed. For the rod-type devices this value must correspond to the speed of rod movement (insertion or extraction), given in cm/s. For the lzc-type devices this value must correspond to the water filling rate, given in m^3/s. The value of *speed* is required only for the reactor regulating purpose.
`TIME`	keyword used to specify a new value for *time*.
time	real value of time either for the rod insertion (or extraction) *or* for the liquid controller filling, given in sec. The value of *time* is required only for the reactor regulating purpose.
`END`	keyword used to indicate the end of input of the new parameters for the current device or group of devices.

3.8 The `MCC:` module

The `MCC:` module supplies tools to edit selectively one or several parameters of a fuel map object. These parameters are the ones defined when the fuel map is created, according to the calculation grid chosen to compute the macroscopic cross-sections libraries. Their selective edition is useful to study their impact on the core reactivity. For instance, this module enables to increase uniformly the fuel temperature in each cell, without modifying any other parameter such as the moderator temperature. It grants access to the Doppler coefficient, even at hot full power when the fuel temperature is different in every cell of the core.

This module enables the computation of (non-exhausive list) : the Doppler coefficient, the power Doppler coefficient (Doppler during a power transient), the moderator temperature coefficient, the boron concentration coefficient, the reactivity with the fuel temperature set to the moderator one, etc.

Note that this module does not perform any reactivity calculation (only fuel map edition).

The `MCC:` module specifications are:

Table 21: Structure `MCC:`

{ [*FLMAP1*] `:=` `MCC:` *FLMAP1* [*FLMAP2*] `::` **(descmcc1)** }

where

FLMAP1	`character*12` name of the MAP object that will contain the updated fuel-lattice information. If *FLMAP1* appears on both LHS and RHS, it will be updated; if it only appears on RHS, it will only be read to display its contents.
FLMAP2	`character*12` name of the MAP object that contains information to be recovered to update *FLMAP1*. If *FLMAP2* exists, data to update *FLMAP1* will be taken in it. If not, data to update *FLMAP1* will be taken in *FLMAP1*.
(descmcc1)	structure describing the main input data to the `MCC:` module. Note that this input data is mandatory and must be specified either if *FLMAP1* is updated or only read.

3.8.1 Main input data to the `MCC:` module

Table 22: Structure **(descmcc1)**

[`EDIT` *iprint*]
[[`REC` *rec1* { `UNI` *value1* | `ADD` *value2* | { `SAME` | `READ` } *rec2* }]]
[`TTD` *pcore*]

where

`EDIT`	keyword used to set *iprint*.
iprint	integer index used to control the printing on screen: = 0 for no print; = 1 for minimum printing (default value); larger values produce increasing amounts of output. A value of 5 or higher will display the contents of *FLMAP1* object after edition, and a value of 6 or higher will display the contents before edition.
`REC`	keyword used to indicate that the name of the record to be updated will follow.
rec1	name of the record to be updated. The authorised values are defined in the table **??**, page **??** (`P-NAME` variable). To be allowed, these values have to be previously defined in the fuel map before calling `MCC`.
`UNI`	keyword to specify that an uniform and absolute value is to be set. With this parameter, the *rec1* of every region will have the same value.
value1	absolute value (in Kelvin if it is a temperature) that will be set for *rec1* in every region.
`ADD`	keyword to specify that an uniform variation of *rec1* value is to be applied. With this parameter, the value of *rec1* will be altered in every region.
value2	value of *rec1* variation (in Kelvin if it is a temperature) that will be applied to every region.
`SAME`	keyword used to indicate that the values to edit *rec1* are to be recovered from *rec2* in the same fuel map, *FLMAP1*. For instance, it enables the user to set the fuel temperature with the moderator temperature.
`READ`	keyword used to indicate that the values to edit *rec1* are to be recovered from *rec2* in an other fuel map, *FLMAP2*. For instance, it enables the user to set the fuel temperature in *FLMAP1* with the fuel temperature of *FLMAP2*.
rec2	name of the record where the data to perform the update of *FLMAP1* is to be recovered. The authorised values are defined in the table **??**, page **??** (`P-NAME` variable). To be allowed, these values have to be previously defined in the fuel map before calling `MCC`.
`TTD`	keyword used to indicate that the values of the `'D-COOL'` record are to be updated. They will be computed from the moderator temperature stored in the `'T-COOL'` folder and the core pressure, using the water tables. Note that the position of the `TTD` keyword matters, the density being calculated when `TTD` is called and not at the end of `MCC:` execution.
pcore	core pressure (in Pa).

3.9 The `MOVDEV:` module

The `MOVDEV:` module can be used for the transient simulations and reactor control studies, which are related to the time-dependent rod-devices displacement in the reactor core. The rods can be inserted into or extracted from the reactor core, at constant or at variable speed of movement. The rod positions are recomputed at every given time step of movement. The new rod positions can be computed in several ways, based on either: current time increment and movement speed; relative change in rod positions; *or* current rod insertion level. The `MOVDEV:` module allows the rod-devices to be displaced individually or simultaneously in groups.

The `MOVDEV:` module specification is:

Table 23: Structure `MOVDEV:`

DEVICE `:=` `MOVDEV:` *DEVICE* `::` **(descmove)**

where

DEVICE	`character*12` name of the DEVICE object that will be modified by the module. The rods positions are updated according to the current time step of movement.
(descmove)	structure describing the input data to the `MOVDEV:` module.

3.9.1 Input data to the `MOVDEV:` module

It is possible to move several individual rods and/or several groups of rods simultaneously. A user must be aware that a particular device will not be displaced more than once during the same time step. Note that the input order of data to the module must be respected.

Table 24: Structure **(descmove)**

[`EDIT` *iprint*]
`DELT` *delt*
[[{ `ROD` *id* | `GROUP` *igrp* }
{ `INSR` | `EXTR` }
{ `LEVEL` *value* | `DELH` *delh* | `SPEED` *speed* }]]
;

where

`EDIT`	keyword used to set *iprint*.
iprint	integer index used to control the printing on screen: = 0 for no print; = 1 for minimum printing (default value); larger values produce increasing amounts of output.

`DELT`	keyword used to set *delt*.
delt	real value of the time increment for the current time step, given in sec.
`ROD`	keyword used to specify the rod *id* number.
id	integer identification number of a rod-type device to be displaced. Each rod has a unique *id* number, ranging from 1 to *nrod*, as been defined in the `DEVINI:` module (see Section **??**).
`GROUP`	keyword used to specify a rod-group *igrp* number.
igrp	integer number of a group of rods that will be displaced simultaneously, with the same parameters of movement. Each group of rod-devices has a unique *igrp* number, ranging from 1 to *ngrp*, as been defined in the `DEVINI:` module (see Section **??**).
`INSR`	keyword used to specify that a particular rod or a group of rods will be inserted into the reactor core during the period of time *delt*.
`EXTR`	keyword used to specify that a particular rod or a group of rods will be extracted from the reactor core during the period of time *delt*.
`LEVEL`	keyword used to specify the new level *value*.
value	real positive value of the rod insertion level at current time step. This value will be used to compute the new rod position in the reactor core. The insertion level is minimal ($value = 0.0$) when the rod is completely withdrawn, and it is maximal ($value = 1.0$) when the rod is fully inserted. For the partially inserted rod the insertion level must be: $0.0 < value < 1.0$
`DELH`	keyword used to specify the value *delh*.
delh	real positive (absolute) value of the relative change in the rod position during the period of time *delt*. This is a time-dependent rod displacement along the rod movement axis, which must be given in cm.
`SPEED`	keyword used to set the current value of *speed*.
speed	real positive (absolute) value of the rod movement speed, given in cm/s. The rod speed can be kept constant or it can be modified at any time step *delt*. The devices could also have the different speeds of movement.

3.10 The `NEWMAC:` module

The `NEWMAC:` module is used to create a complete MACROLIB with respect to the devices parameters. The resulting MACROLIB will contain the exact properties for every material region, over the whole mesh-splitted reactor geometry. The material properties of each region are recomputed with respect to the actual position of each rod-type and if present lzc-type device. The computing algorithm is based on the determination of the volumic fraction occupied by each device; the incremental cross sections are then adjusted, accordingly. Note that the `NEWMAC:` module must be executed each time the devices positions are modified from the previously computed ones.

The `NEWMAC:` module specification is:

Table 25: Structure `NEWMAC:`

MACRO3 MATEX `:=` `NEWMAC:` *MATEX MACRO2 DEVICE* `::` [`EDIT` *iprint*] [`XFAC` *xfac*] ;

where

MACRO3	`character*12` name of the MACROLIB to be created by the module. It will contain the updated properties of each material region with respect to the current position of each device.
MATEX	`character*12` name of the MATEX object, containing the complete reactor material index including devices. *MATEX* must be specified in the modification mode; it will store the updated h-factors, computed per each fuel region with respect to the devices positions.
MACRO2	`character*12` name of the read-only extended MACROLIB, previously created by the `MACINI:` module.
DEVICE	`character*12` name of the read-only DEVICE object containing the devices information and parameters.
`EDIT`	keyword used to set *iprint*.
iprint	integer index used to control the printing on screen: = 0 for no print; = 1 for minimum printing; larger values produce increasing amounts of output. The default value is *iprint* = 1.
`XFAC`	keyword used to specify the number of cells on which incremental cross sections were computed in the supercell code.
xfac	corrective factor for delta sigmas (real number). For DRAGON code, *xfac* is generally set to 2.0 and, for MULTICELL code, set to 1.0 . The default value is 2.0.

3.11 The `FLPOW:` module

The `FLPOW:` module is used to compute and print the flux and power distributions over the reactor core. It also computes and prints some additional information, for example: the fluxes ratios with respect to the thermal energy-group fluxes; the mean power density; the power- and flux-form factors; etc. The computed fluxes and powers are printed either on files or on the screen. Note that the calculation using the `FLPOW:` module can be performed once the numerical solution has been previously established using the `FLUD:` or `KINSOL:` module.

According to the user-selected module specification, the average fluxes and powers can be computed per each fuel region over the fuel lattice *and/or* per each material region over the whole reactor geometry. In either case, all fluxes are normalized to the given total reactor power corresponding to the reactor nominal conditions at core equilibrium. If the reactor is perturbed from its initial state, then a new total reactor power can be recomputed and, accordingly, the flux and power distributions will be updated using the previously computed normalization factor.

The `FLPOW:` module will create a new `POWER` object that will store the information related to the reactor fluxes and powers (see Section **??**). In addition, the `POWER` object will store several parameters that can be used as power and criticity constraints for the optimization and fuel management purposes, namely: the maximum channel and bundle powers; the channel and bundle power-form factors; the effective multiplication factor (recovered from the FLUX or KINET data structure).

The `FLPOW:` module specifications are:

Table 26: Structure `FLPOW:`

{
  POWER [*NRMFLUX*] [*FMAP*]
    := `FLPOW:` [*POWOLD*] *FMAP* { *FLUX* | *KINET* } *TRACK MATEX*
    :: **(descflpow)**
|
  POWER := `FLPOW:` [*POWOLD*] { *FLUX* | *KINET* } *TRACK MACRO*
    :: **(descflpow)**
}

where

POWER	`character*12` name of the POWER object that will be created by the module. It will contain the information related to the reactor fluxes and powers.
NRMFLUX	`character*12` name of the FLUX object, in creation mode. According to the chosen option, this object contains either the fluxes normalized to the given total reactor power or the fluxes per bundle. Is it useful if you want to compute the detectors readings with the `DETECT:` module.
POWOLD	`character*12` name of the read-only POWER object. It must contain the previously computed flux normalization factor, which corresponds to the reactor nominal or equilibrium conditions.
FMAP	`character*12` name of the FMAP object containing the fuel lattice specification. When *FMAP* is specified on the RHS, the fluxes and powers calculations are performed over

the fuel lattice as well as over the whole reactor geometry. If *FMAP* is specified on the LHS, its records `'BUND-PW'` and `'FLUX-AV'` will be set according to the information present in *POWER*.

FLUX	`character*12` name of the FLUX object, previously created by the `FLUD:` module. The numerical flux solution contained in *FLUX* is recovered and all flux are normalized to the given total reactor power.
KINET	`character*12` name of the KINET object, previously created by the `KINSOL:` module. The numerical flux solution contained in *KINET* is recovered.
TRACK	`character*12` name of the TRACK object, created by the `TRIVAT:` module. The information stored in *TRACK* is recovered and used for the average flux calculation.
MATEX	`character*12` name of the MATEX object, containing the reactor material index and the h-factors that will be recovered and used for the power calculation.
MACRO	`character*12` name of the MACROLIB object, containing the h-factors that will be recovered and used for the power calculation.
(descflpow)	structure describing the input data to the `FLPOW:` module .

3.11.1 Input data to the `FLPOW:` module

Note that the fuel-lattice power distribution can be printed only on the screen.

Table 27: Structure **(descflpow)**

```
[ EDIT iprint ]
[ { PTOT power | P-NEW } ]
[ FSTH fsth ] [ INIT ]
[ { NORM | BUND } ]
[ PRINT { MAP | DISTR [ FLUX ] [ RATIO ] [ POWER ] | ALL } ]
;
```

where

`EDIT`	keyword used to set *iprint*.
iprint	integer index used to control the printing on screen: = 0 for no print; = 1 for minimum editing (default value); = 2 only channel powers in radial plane are printed; = 3 only bundle powers per each radial plane are printed; = 10 only bundle powers per each channel are printed. Any combination of the values 2, 3 and 10 is possible, for example 5 = 2+3. Note that any other value of *iprint* behaves as the first lower possible value, for example 7 gives the same output as 5. Moreover channel and bundle powers can be printed only if the *FMAP* object was provided in the calling specification.
`PTOT`	keyword used to specify the input of *power*. By default, a power is recovered from the *KINET* object.

power	real total reactor power, given in MW. This value must correspond to the reactor nominal conditions.
`FSTH`	keyword to specify the thermal to fission power ratio.
fsth	thermal to fission power ratio. By default this value is not used, and the total power is the one given after the `PTOT` keyword.
`INIT`	keyword used to save the actual power distribution in the `BUND-PW-INI` record of the fuel map object *FMAP*. It is used by the `AFM:` module to apply power feedback during a fast transient using the initial power distribution instead of the actual power.
`P-NEW`	keyword used to indicate that a new total reactor power is to be recomputed, based on the previously calculated flux normalization factor. The flux and power distributions over the reactor core are updated, accordingly. Note that this option is valid only if a read-only *POWOLD* object is provided.
`PRINT`	keyword used to indicate the printing on files. Note that all produced files will have the same extension "`.res`".
`MAP`	keyword used to specify the printing of the average fluxes and flux ratios per fuel bundle. The normalized bundle fluxes are computed and printed for each reactor channel and per each energy group. The flux ratios are computed with respect to the thermal energy-group fluxes; they are printed on the same file.
`DISTR`	keyword used to indicate the printing of data computed over the whole reactor geometry.
`FLUX`	keyword used to specify the printing of flux distribution. The normalized fluxes are printed in separated files, one file per energy group; the number of produced files will then equal to the total number of energy groups. The flux values are printed for each mesh-splitted volume, in X, Y and Z planes; the virtual regions will have the fluxes values set to 0.
`RATIO`	keyword used to specify the printing of flux-ratio distribution. The flux ratios are computed with respect to the thermal energy-group fluxes per each mesh-splitted volume. They are printed in separated files; the number of produced files will equal to the total number of energy groups less one.
`POWER`	keyword used to specify the printing of power distribution. The power values are printed for each mesh-splitted volume, in X, Y, and Z planes; the non-fuel regions will have the power values set to 0.
`ALL`	keyword used to indicate the printing of all available information, i.e. without particular selection of data.
`NORM`	keyword to specify that the output flux object will contain a value per mesh-splitted element, normalized to the given power, as required by the `DETECT:` module. This is the default option.
`BUND`	keyword to specify that the output flux object will contain a value per bundle, normalized to the given power.

3.12 The `TAVG:` module

The `TAVG:` module is used to compute the burnup integration limits for each fuel bundle, the axial power-shape over the fuel lattice, the channel refuelling rates and the reactor core-average exit burnup. All calculations using the `TAVG:` module are performed according to the time-average model for the equilibrium-core conditions. The computing algorithm is based on bidirectional refuelling schemes of channels and average exit burnups specified over the fuel lattice, which should be recorded in the fuel map using the `RESINI:` module.

Note that the complete time-average calculation is a complex and iterative procedure, requiring of several full-core calculations (external iterations) to be performed. The main steps of the time-average calculation using DONJON are briefly described at the end of this section. The `TAVG:` module can also be used to compute the instantaneous fuel burnups according to the channel patterned-age-model, for the fuel management and optimization purposes.

The `TAVG:` module specification is:

Table 28: Structure `TAVG:`

```
FMAP := TAVG: FMAP POWER :: (desctavg)
```

where

FMAP	`character*12` name of a FMAP object, that will be updated by the `TAVG:` module. The *FMAP* object must contain the average exit burnups and refuelling schemes of channels.
POWER	`character*12` name of a POWER object containing the channel and bundle powers, previously computed by the `FLPOW:` module. The channel and bundle powers are used by the `TAVG:` module to compute the normalized axial power-shape over each channel.
(desctavg)	structure describing the input data to the `TAVG:` module.

3.12.1 Input data to the `TAVG:` module

Note that the input order must be respected.

Table 29: Structure **(desctavg)**

```
[ EDIT iprint ]
[ AX-SHAPE [ RELAX relval ] ]
[ B-EXIT ]
;
```

where

`EDIT`	keyword used to set *iprint*.
iprint	integer index used to control the printing on screen: = 0 for no print; = 1 for minimum printing (default value); = 2 only the burnup limits over each channel are printed; = 3 only the axial power-shape values over each channel are printed; = 4 only the channel refuelling rates are printed; for larger values of *iprint* everything will be printed.
`AX-SHAPE`	keyword used to indicate the calculation of the new axial power-shape and corresponding burnups limits over each reactor channel.
`RELAX`	keyword used to set the relaxation parameter *relval*.
relval	real value of the relaxation parameter, generally used to control the axial-shape convergence over the external time-average iterations. The optimal value, which corresponds to the minimal total number of such iterations, can be found by performing several runs at different *relval*. The default value of the relaxation parameter is set to 0.5
`B-EXIT`	keyword used to indicate the calculation of the core-average exit burnup and the channel refuelling rates.

3.12.2 Time-average calculation using DONJON

When the average exit burnups are provided for each channel, the exact burnup integration limits for each fuel bundle are unknown and need to be determined. The burnups integration limits are function of the normalized axial power-shape, which in turn depends on the flux solution over the fuel lattice. Moreover, the flux solution depends on the fuel-map macrolib (i.e. fuel properties), which in turn depends on the burnups integration limits for each fuel bundle. Consequently, the time-average calculation is an iterative procedure that consists to repeat all the steps required for the axial power-shape computation. This repetition is to be made until the relative error between the two (successives) axial power-shape calculations becomes as small as required for the precision.

The axial power-shape computing scheme is composed of several steps, each step is performed using an appropriate DONJON or TRIVAC module:

1. An initial axial power-shape is set as a flat distribution over the fuel lattice and the first burnup integration limits are calculated approximately, using the `RESINI:` module.
2. A time-average integration is performed and a new fuel-map MACROLIB is created, using either `NCR:`, `CRE:` or `AFM:` module.
3. An extended MACROLIB over the whole reactor geometry is created, using the `MACINI:` module.
4. If the devices are inserted into the reactor core, then the previously created MACROLIB is to be updated for the devices properties using the `NEWMAC:` module.
5. The complete MACROLIB is subsequently used by the `TRIVAA:` module in order to create a matrix SYSTEM.
6. The full-core numerical solution (i.e. fluxes and effective multiplication factor) is computed, using the `FLUD:` module.
7. The channel and bundle powers are next calculated, using the `FLPOW:` module.
8. Finally, the new axial power-shape and burnup limits are computed, using the `TAVG:` module.

Note that the steps from 2 to 8 are to be repeated until the required precision for the axial power-shape convergence is satisfied.

3.13 The `TINST:` module

The `TINST:` module is used to compute the instantaneous burnup for each fuel bundle. You can also use `TINST:` to refuel your reactor, according to a refueling-scheme. The scheme can be either specified with `RESINI:`, or directly in `TINST:`.

The `TINST:` module specification is:

Table 30: Structure `TINST:`

{ *FMAP* := `TINST:` *FMAP* [*POWER*] |
MICLIB3 FMAP := `TINST:` *FMAP MICLIB2 MICLIB* }
:: **(desctinst)**

where

FMAP	`character*12` name of a FMAP object, that will be updated by the `TINST:` module. The *FMAP* object must contain the instantaneous burnups for each fuel bundle and the weight of each fuel mixture.
POWER	`character*12` name of a POWER object containing the channel and bundle powers, previously computed by the `FLPOW:` module. The channel and bundle powers are used by the `TINST:` module to compute the new burn-up of each bundle. If bundle-powers are previously specified with the module `RESINI:`, you can refuel your core without a *POWER* object.
MICLIB3	`character*12` name of a LIBRARY object, that will be created by the `TINST:` module. This MICROLIB contains the fuel properties after refueling when keyword MICRO is used in **(desctinst)**.
MICLIB2	`character*12` name of a LIBRARY object, that will be read by the `TINST:` module. This must be a fuel-map *LIBRARY* created either created by the `NCR:` or the `EVO:` module.
MICLIB	`character*12` name of a LIBRARY object, that will be read by the `TINST:` module. This MICROLIB contains the new fuel properties, that should be used for the refueling.
(desctinst)	structure describing the input data to the `TINST:` module.

3.13.1 Input data to the `TINST:` module

Note that the input order must be respected.

Table 31: Structure **(desctinst)**

```
[ EDIT iprint ]
[ BURN-STEP rburn | TIME rtime { DAY | HOUR | MINUTE | SECOND } ]
[[ REFUEL [ MICRO ] CHAN NAMCHA nsh ]]
[[ NEWFUEL [ MICRO ] CHAN NAMCHA nsh { SOME ( imix(i), i=1,ABS(nsh) ) | ALL imix } ]]
[[ SHUFF CHAN NMCHA1 TO { NMCHA2 | POOL } ]]
;
```

where

`EDIT`	keyword used to set *iprint*.
iprint	integer index used to control the printing on screen: = 0 for no print; = 1 for minimum printing (default value); = 2 only the burnup limits over each channel are printed; = 3 only the axial power-shape values over each channel are printed; = 4 only the channel refueling rates are printed; for larger values of *iprint* everything will be printed.
`BURN-STEP`	keyword used to indicate an increase of core average burn-up.
rburn	keyword used to indicate in MWd/t the average increase of burn-up in the core.
`TIME`	keyword used to indicate the time of combustion at the power specified in *POWER* structure.
rtime	keyword used to set the time combustion value in `DAY` or `HOUR` or `MINUTE` or `SECOND`.
`DAY`	keyword used to specify that *rtime* is a number of days.
`HOUR`	keyword used to specify that *rtime* is a number of hours.
`MINUTE`	keyword used to specify that *rtime* is a number of minutes.
`SECOND`	keyword used to specify that *rtime* is a number of seconds.
`REFUEL`	key word to specify a channel refueling.
`MICRO`	keyword used to perform a microscopic refueling. In this case, three libraries have to be provided when `TINST:` is called.
`CHAN`	key word to specify the refueled channel information.
NAMCHA	channel name. In Cartesian geometry, *NAMCHA* is a character*4 variable defined by *NXNAME* and *NYNAME* and constructed as `WRITE(NAMCHA,'(A1,A3)')` *NYNAME*(1:1),*NXNAME*(1:2).
nsh	refueling scheme. The absolute value of *nsh* is the number of fuel bundles inserted in the channel *NAMCHA*. The sign of *nsh* define the refueling direction: positive direction is from the first to the *nk*-th bundle and negative is from the *nk*-th to the first bundle.

`NEWFUEL`	key word to specify that a channel will be refueled with a different type of fuel.
`SOME`	key word to specify that the *nsh* values of fuel types can be different.
imix(i)	index number of a fuel type with respect to the values defined in module `NCR:`, `CRE:` or `AFM:`.
`ALL`	key word to specify that the *nsh* values of fuel types will be identical to *imix*.
`SHUFF`	key word to specify that a specified channel will move into an other one or discharge into the pool.
`CHAN`	key word to specify the moved channel name.
NMCHA1	channel name as defined by *NXNAME* and *NYNAME*. It is constructed as *NAMCHA*.
`TO`	key word to specify the bundle destination.
NMCHA2	channel name as defined by *NXNAME* and *NYNAME*. It is constructed as *NAMCHA*.
`POOL`	key word to specify that the channel referenced by *NMCHA1* is discharged into the pool.

3.14 The `SIM:` module

The `SIM:` module can perform a sequence of operations related to fuel management in PWRs:

- simulate a refuelling and shuffling scheme and update the burnup distribution accordingly. The refuelling scheme is specified directly in `SIM:`.
- increase the burnup using the power available in the *POWER* object and compute the final instantaneous burnup of each assembly subdivision
- modify a local parameter such as the Boron concentration in the coolant.

The `SIM:` module specification is:

Table 32: Structure `SIM:`

```
FMAP := SIM: FMAP [ POWER ]
:: (descsim)
```

where

FMAP	`character*12` name of a FMAP object, that will be updated by the `SIM:` module. The *FMAP* object must contain the instantaneous burnups for each assembly subdivision, a basic naval-coordinate assembly layout and the weight of each assembly subdivision.
POWER	`character*12` name of a POWER object containing the channel and powers of the assembly subdivisions, previously computed by the `FLPOW:` module. The channel and powers of the assembly subdivisions are used by the `SIM:` module to compute the new burn-up of each assembly subdivision. If the powers of the assembly subdivisions are previously specified with the module `RESINI:`, you can burn your core without a *POWER* object.
(descsim)	structure describing the input data to the `SIM:` module.

3.14.1 Input data to the `SIM:` module

Note that the input order must be respected.

Table 33: Structure **(descsim)**

```
[ EDIT iprint ]
[ CYCLE hcnew [ FROM hcold [ BURN { indcycle | burncycle } ] ]
    [ { MAP (hx(i), i=1, lx )
            (hy(j), (hcase(i,j), i=1, lx ), j=1,ly ) |
      QMAP (hx(i), i=lx/2+1, lx )
            (hy(j), (hcase(i,j), i=lx/2+1, lx ), j=ly/2+1,ly ) } ]
```

continued on next page

Structure **(descsim)** continued from last page

```
    [ SPEC [[ [[ asmb1 ]]
      { SET AVGB avburn | SET FUEL ifuel | FROM hcold2 AT asmb2 [ BURN { indcycle | burncycle } ] }
          ]] ]
    [ DIST-AX [[ [[ asmb1 ]]
      { SET (axn(i), i=1,nb) | FROM hcold2 AT asmb2 [ BURN { indcycle | burncycle } ] }
              ]] ]
    [ BURN-STEP rburn | TIME rtime { DAY | HOUR | MINUTE | SECOND } ]
ENDCYCLE ]
[[ COMPARE hc1 [ BURN { indcycle1 | burncycle1 } ] hc2 [ BURN { indcycle2 | burncycle2 } ]
      { DIST-BURN >> epsburn << | DIST-POWR >> epspowr << } ]]
[[ SET-PARAM PNAME pvalue ]]
;
```

where

`EDIT`	keyword used to set *iprint*.
iprint	integer index used to control the printing on screen: = 0 for no print; = 1 for minimum printing (default value); for larger values of *iprint* everything will be printed.
`CYCLE`	keyword defining operations based on the actual fuel cycle.
hcnew	`character*12` identification name of the specific fuel cycle.
`FROM`	keyword defining the previous fuel cycle in case that some information needs to be transmitted to the actual fuel cycle.
hcold	`character*12` identification name of the previous fuel cycle.
`BURN`	keyword defining the burnup at which the assembly is recycled in the previous fuel cycle. By default, the last burnup step is used.
indcycle	integer index of the burnup step in the previous fuel cycle.
burncycle	real value of the burnup in the previous fuel cycle.
`MAP`	keyword defining the assembly layout in naval-coordinate positions in the actual fuel cycle. Here, *lx* and *ly* values are those defined in the fuel map (see Section **??**).
`QMAP`	keyword defining the assembly layout in naval-coordinate positions using quarter-core symmetry conditions. Here, the lower-right quarter is defined. The full map is reconstructed through rotations around the center.
hx	ordered list of available `character*1` prefixes for the X-oriented naval-coordinate positions. Values are generally chosen between `A` and `T`.
hy	ordered list of available `character*2` suffixes for the Y-oriented naval-coordinate positions. Values are generally chosen between `01` and `17`.
hcase	`character*4` identification value for the (i,j) position. Accepted values are: • `|`, `-` or `-|-` for a position outside the core, • `NEW` for a new assembly (at zero burnup) selected according to the fuel map specified in Sect. **??**,

- `SPC` for an assembly described later in the dataset using a `SPEC` specification,
- or a naval-coordinate position referring to the position of an assembly in cycle *hcold* that is recycled in the current cycle.

`SPEC`	keyword defining specifications related to all assemblies previously identified with the `SPC` keyword. If `QMAP` keyword has been used with `SPC` values, the 4 equivalent assemblies must be specified (i.e. not only the lower-right quarter assembly).
asmb1	`character*3` naval-coordinate position of an assembly identified with a `SPC` keyword. Up to 30 coordinates can be set aside if many assemblies have the same specification.
`SET`	keyword indicating that a user-defined value will be assigned to the assembly.
`AVGB`	keyword indicating that an averaged burnup will be assigned to the assembly.
avburn	real value of the average burnup in MWd/t.
`FUEL`	keyword indicating that a new fuel assembly will be used.
fuel	integer index of the fuel type corresponding to the new fuel assembly. Fuel type indices are those used in the `RESINI: PLANE` descriptions of Sect. **??**.
`FROM`	keyword indicating that a value recovered from another assembly will be assigned to the current assembly.
hcold2	`character*12` identification name of a previous fuel cycle.
`AT`	keyword indicating that the naval-coordinate position of the other assembly will be given.
asmb2	`character*3` naval-coordinate position of the other assembly in cycle *hcold2*.
`DIST-AX`	keyword used to impose an axial burnup distribution to the assembly. The burnup distribution is recovered from an existing assembly or is set to user-suppled values.
axn	real values of the axial burnup distribution.
`BURN-STEP`	keyword used to indicate an increase of core average burn-up.
rburn	keyword used to indicate in MWd/t the average increase of burn-up in the core.
`TIME`	keyword used to indicate the time of combustion at the power specified in *POWER* structure.
rtime	keyword used to set the time combustion value in `DAY` or `HOUR` or `MINUTE` or `SECOND`.
`DAY`	keyword used to specify that *rtime* is a number of days.
`HOUR`	keyword used to specify that *rtime* is a number of hours.
`MINUTE`	keyword used to specify that *rtime* is a number of minutes.
`SECOND`	keyword used to specify that *rtime* is a number of seconds.
`ENDCYCLE`	keyword indicating the end of data specific to the actual fuel cycle.
`COMPARE`	keyword for obtaining a CLE-2000 variable that is a measure of the discrepancy between two cycles.
hc1	`character*12` identification name of the first fuel cycle to compare.
hc2	`character*12` identification name of the second fuel cycle to compare.

`DIST-BURN`	keyword used to recover the discrepancy on burnup distribution in a CLE-2000 variable.
epsburn	`character*12` CLE-2000 variable name in which the extracted burnup discrepancy (expressed in MW-day/tonne) will be placed.
`DIST-POWR`	keyword used to recover the relative error on power distribution in a CLE-2000 variable.
epspowr	`character*12` CLE-2000 variable name in which the extracted power relative error will be placed.
`SET-PARAM`	keyword used to indicate the input (or modification) of the actual values for a parameter specified using its *PNAME*.
`PNAME`	keyword used to specify *PNAME*.
PNAME	`character*12` name of a parameter.
pvalue	single real value containing the actual parameter's values. Note that this value will not be checked for consistency by the module. It is the user responsibility to provide the valid parameter's value which should be consistent with those recorded in the multicompo or Saphyb database.

3.15 The XENON: module

The XENON: module is used to correct the Xenon distribution coming from an interpolation calculation. This module computes the new densities according to the bundle flux, and the equation providing the balance concentration of Xenon-135 :

$$N_{X_{eq}} = \frac{(Y_I + Y_X)\Sigma_f \phi}{\lambda_X + \sigma_X \phi} \tag{3.1}$$

where

- Y_I is the fission yield of I135
- Y_X is the fission yield of Xe135
- σ_X is the capture cross section of Xe135
- λ_X is the decay constant of Xe135
- Σ_f is the total fission cross section
- ϕ is the bundle flux

The XENON: module specification is:

Table 34: Structure XENON:

```
MICROLIB := XENON: MICROLIB [ POWER ]
:: (descxenon)
```

where

MICROLIB	`character*12` name of a LIBRARY object, that will be updated by the XENON: module. The Xenon should be extracted in this library for the use of this module.
POWER	`character*12` name of a POWER object containing the bundle fluxes, previously computed by the FLPOW: module. The fluxes should be normalized to the reactor power.
(descxenon)	structure describing the input data to the XENON: module.

3.15.1 Input data to the XENON: module

Note that the input order must be respected.

Table 35: Structure **(descxenon)**

```
[ EDIT iprint ]
```

continued on next page

Structure **(descxenon)** continued from last page

```
[ INIT ]
;
```

where

`EDIT`	keyword used to set *iprint*.
iprint	integer index used to control the printing on screen.
`INIT`	keyword used to indicate the initialization of the library for a recursive calculation using the `XENON:` module. The Xenon concentration is set to zero for all the bundles.

3.16 The `DETECT:` module

The `DETECT:` module is used to compute the mean flux at each detector site and the response of each detector.
The `DETECT:` module specifications are:

Table 36: Structure `DETECT:`

DETEC `:=` `DETECT:` *DETEC FLUX TRACK GEOM* `::` **(descdetect)** `;`

where

DETEC	`character*12` name of the DETECT containing the detector positions and responses.
FLUX	`character*12` name of the FLUX containing the flux solution computed by the `FLUD:` or `FLPOW:` modules. To obtain a correct result, the best is to use a normalized flux, coming from the `FLPOW:` module. In this case, the fluxes are normalized to the reactor power.
TRACK	`character*12` name of the TRACK containing the TRIVAC tracking.
GEOM	`character*12` name of the GEOMETRY containing the mesh-splitting geometry created by the `USPLIT:` or `GEO:` modules.
(descdetect)	structure containing the data to module `DETECT:`.

3.16.1 Input data to the `DETECT:` module

Note that the fuel-lattice power distribution can be printed only on the screen.

Table 37: Structure **(descdetect)**

[`EDIT` *iprt*] `TIME` *dt* `REF` *kc*
[`NORM` *vnorm*]
[SIMEX { SPLINE | PARAB }]
;

where

`EDIT`	key word used to set *iprt*.
iprt	index used to control the printing in module `DETECT:`. =0 for no print; =1 for minimum printing(default value); =4 for printing each detector name; =5 for finite element numbers and total number of finite elements for each detector.

`TIME`	key word used to set dt.
dt	time step between two calls to the `DETECT:` module.
`REF`	key word used to set kc.
kc	index used to control the type of calculation, =0 for reference calculation; =1 normal calculation. The reference responses are used to obtain detector current responses in full power fractions.
`NORM`	key word used to set *vnorm*.
vnorm	value used to normalized responses of all the detectors present in DETECT.
`SIMEX`	key word used to specify that a polynomial interpolation of detector fluxes according to HQSIMEX method. This interpolation will be applied only for vanadium detectors, under *NAMTYP* of value `VANAD_REGUL`.
`SPLINE`	key word to specify that the flux at detector site will be computed with a spline method.
`PARAB`	key word to specify that the flux at detector site will be computed with a parabolic method.

3.17 The `CVR:` module

The `CVR:` module is used to update the fuel-type index and the coolant densities throughout the reactor core as required for the voiding simulations. A particular core-voiding pattern is either selected from the several pre-defined patterns *or* directly defined by the user in an arbitrary fashion. In the last case, the user may specify the individual voided channels by indicating their identification names. The `CVR:` module will create a new (perturbed) FMAP object, in which the fuel-type mixtures indices are modified according to the specified core-voiding pattern. The information with respect to the relative coolant densities is required only for the subsequent interpolation of fuel properties using the `NCR:` module. These data will also be reordered by the `CVR:` module according to the specified voiding pattern and recorded as local parameter in the perturbed fuel-map object (see Section **??**).

The `CVR:` module specification is:

Table 38: Structure `CVR:`

FMAPV `:=` `CVR:` *FMAP* `::` **(descrcvr)**

where

FMAP	`character*12` name of a read-only FMAP object, created in the `RESINI:` module. This object must contain the non-perturbed fuel-cell properties.
FMAPV	`character*12` name of a new FMAP object, that will contain the modified fuel-type indices and reordered coolant densities according to the specified core-voiding pattern.
(descrcvr)	structure describing the input data to the `CVR:` module.

3.17.1 Input data to the `CVR:` module

Note that the input order must be respected.

Table 39: Structure **(descrcvr)**

`EDIT` *iprint*
(`MIX-FUEL` *mixF*(i) `MIX-VOID` *mixV*(i) , i = 1, *nfuel*)
[`DENS-COOL` *PNAME* `SET` *dcoolV*]
`VOID-PATTERN` { `FULL` | `HALF` | `QUARTER` | `CHECKER` | `CHECKER-1/2` | `CHECKER-1/4` |
`CHAN-VOID` *nvoid* (*YNAME*(i) *XNAME*(i) , i = 1, *nvoid*) }
;

where

`EDIT`	keyword used to set *iprint*.

iprint	integer index used to control the printing on screen: = 0 for no print; = 1 for minimum printing; = 2 modified fuel indices and coolant densities are printed per bundle over each channel; = 3 modified fuel indices are printed per each radial plane; for larger values of *iprint* everything will be printed.
`MIX-FUEL`	keyword used to specify *mixF*.
mixF	integer fuel-type mixture number of the non-perturbed fuel cell. This number must be specified for each fuel type as been recorded in the MATEX object (see Section **??**).
`MIX-VOID`	keyword used to specify *mixV*.
mixV	integer new mixture number assigned to the voided fuel cell. Note that this number must be specified for each fuel type and it must be different from any other reactor material mixtures.
`DENS-COOL`	keyword used to specify *PNAME*. This information is required only for the interpolation of fuel properties using the `NCR:` module.
PNAME	`character*12` user-defined identification name of local parameter associated with the relative coolant density. The recommended name is `D-COOL`. This parameter name and the unperturbed densities values should be previously recorded in the *FMAP* object (see Section **??**). The same *PNAME* will be set for the coolant density in the perturbed *FMAPV*, but the actual values of coolant densities throughout the core will be reordered by the `CVR:` module according to the specified voiding pattern.
`SET`	keyword used to specify the value *dcoolV*.
dcoolV	real value of the relative coolant density (with respect to the nominal or unperturbed conditions) associated with the voided reactor channels. In general, this value equals to 0.0 for the complete voiding of a channel and to 1.0 for an unperturbed channel. Intermediate values of *dcoolV* will then correspond to the partially voided channels. It is supposed that all voided channels will have the same *dcoolV* value.
`VOID-PATTERN`	keyword used to specify the core voiding pattern, which will be used for a particular voiding simulation.
`FULL`	keyword used to specify the full-core voiding pattern. According to this pattern, the fuel mixtures will be modified for all reactor channels.
`HALF`	keyword used to specify the half-core voiding pattern. According to this pattern, the fuel mixtures will be modified only for the upper-half of reactor channels.
`QUARTER`	keyword used to specify the quarter-core voiding pattern. According to this pattern, the fuel mixtures will be modified only for the upper-left quarter of reactor channels.
`CHECKER`	keyword used to specify the checkerboard-full voiding pattern. According to this pattern, the fuel mixtures will be modified for all reactor channels in which the direction of coolant flow is positive.
`CHECKER-1/2`	keyword used to specify the checkerboard-half voiding pattern. According to this pattern, the fuel mixtures will be modified only for the upper-half of reactor channels in which the direction of coolant flow is positive.
`CHECKER-1/4`	keyword used to specify the checkerboard-quarter voiding pattern. According to this pattern, the fuel mixtures will be modified only for the upper-left quarter of reactor channels in which the direction of coolant flow is positive.

`CHAN-VOID`	keyword used to specify the user-defined voiding pattern. Each voided channel must be identified by its *YNAME* name followed by its *XNAME*.
nvoid	integer total number of the voided channels. This number must be greater than 0 and less than (or equal to) the total number of reactor channels.
YNAME	`character*2` vertical name of the voided channel. A vertical channel name is identified by the channel row using an alphabetical letter ('A', 'B', 'C', etc). The total number of the specified Y-names must equal to the total number of voided channels *nvoid*.
XNAME	`character*2` horizontal name of the voided channel. A horizontal channel name is identified by the channel column using a numerical character ('1', '2', '3', etc.). The total number of the specified X-names must equal to the total number of voided channels *nvoid*.

3.18 The HST: module

The `HST:` module has been designed to manage a full reactor execution in DONJON using explicit DRAGON calculations for each cell.[?] This module saves in an HISTORY data structure the information available in BURNUP data structures generated by DRAGON. It can also read MAP data structure generated by DONJON to prepare the HISTORY data structure for a new series of cell calculations in DRAGON. The HISTORY data structure can also be used to update the MAP data structure. Finally, the module `HST:` can be used to create an initial BURNUP data structure that can be used to evolve the cell another time step in DRAGON.

The `HST:` module can be used to create or update an HISTORY data structure. The possible options are:

Jpdating an HISTORY structure using a MAP structure

```
HISTORY := HST: [ HISTORY ] MAP [ :: [ (hstdim) ] [ GET (hstpar) ] ]
```

dating an HISTORY structure using a BURNUP structure

```
HISTORY := HST: [ HISTORY ] [ BURNUP ] [ :: [ (hstdim) ]
        [ GET (hstpar) ] [ CELLID icha ibun [ idfuel ] [ GET (hstpar) ] ] ]
```

It can also be used to create a BURNUP data structure from the information available on an HISTORY data structure:

dating a BURNUP structure using an HISTORY structure

```
BURNUP := HST: HISTORY [ :: [ (hstdim) ]
        [ PUT (hstpar) ]
        CELLID icha ibun
        [ PUT { BREFL (hstbrn) (hstpar) AREFL (hstbrn) (hstpar) | [ AREFL ] (hstbrn) (hstpar)
} ] ]
```

It can also be used to update a MAP data structure from the information available on an HISTORY data structure:

Jpdating an HISTORY structure using a MAP structure

```
MAP := HST: MAP HISTORY
```

where

HISTORY	`character*12` name of an HISTORY data structure.
BURNUP	`character*12` name of a BURNUP data structure.
MAP	`character*12` name of a MAP data structure.
(hstdim)	structure containing the dimensions for the HISTORY data structure.
`CELLID`	keyword to identify the cell for which history information is to be processed.
icha	channel number for which history information is to be processed.
ibun	bundle number for which history information is to be processed.
idfuel	fuel type number associated with this cell. One can associate to each fuel cell a different fuel type. By default a single fuel type is defined and it fills every fuel cell. Only the initial properties of each fuel type are saved. These properties are used for refueling.
`GET`	keyword to specify that the values of the parameters selected in **(brnpar)** will be read from the input stream or CLE-2000 local variables and stored on the HISTORY data structure.
`PUT`	keyword to specify that the values of the parameters selected in **(brnpar)** will be read from the HISTORY data structure and transferred to local CLE-2000 variables.
`BREFL`	to specify that the information to extract from the HISTORY data structure is related to the properties of the cell before refueling takes place.
`AREFL`	to specify that the information to extract from the HISTORY data base is related to the properties of the cell after refueling took place.
(hstbrn)	structure containing the burnup options.
(hstpar)	structure containing the local parameters options.

The **(hstdim)** input structure is required for general dimensioning purpose. It is generally used only when creating the HISTORY data structure. However, the number of global and local parameters used in a HISTORY data structure can be increased at all time. The number of channels, bundles and the refueling scheme must be defined at the creation of the HISTORY data structure. This information can be provided manually or extracted from a MAP data structure. The general form of the **(hstdim)** input structure follows:

Table 44: Structure **(hstdim)**

[`EDIT` *iprint*]
[`DIMENSIONS` [`GLOBAL` *nglo*] [`LOCAL` *nloc*] [`BUNDLES` *nbun bunl*] [`CHANNELS` *ncha*]]

where

`EDIT`	keyword used to modify the print level *iprint*.
iprint	index used to control the printing in this module. It must be set to 0 if no printing on the output file is required.
`DIMENSIONS`	keyword used to indicate that the general dimensioning of the HISTORY data structure will be modified.

`GLOBAL`	keyword used to modify the number of global parameters on the HISTORY data structure.
nglo	the number of global parameters. Note that the history module will use the maximum value between the current *nglob* and the value, if any, defined on the HISTORY data structure.
`LOCAL`	keyword used to modify the number of local parameters on the HISTORY data structure.
nloc	the number of local parameters. Note that the history module will use the maximum value between the current *nloc* and the value, if any, defined on the HISTORY data structure.
`BUNBLES`	keyword used to specify the number of bundles per channels for the reactor model considered in the HISTORY data structure.
nbun	the number of bundles per channels for the reactor model. Note that if *nbun* is different from the value already defined on the HISTORY data structure or the MAP data structure, the execution will be aborted.
bunl	bundle length in cm. This information is required to compute inital fuel weight.
`CHANNELS`	keyword used to specify the number of fuel channels for the reactor model considered in the HISTORY data structure.
ncha	the number of fuel channels for the reactor model. Note that if *ncha* is different from the value already defined on the HISTORY data structure or the MAP data structure, the execution will be aborted.

The **(hstbrn)** serves a unique purpose, mainly to extract from the HISTORY file the information required to process a burnup evaluation in DRAGON using the `EVO:` module. The information must be stored inside CLE-2000 variables. The general form of this output structure is:

Table 45: Structure **(hstbrn)**

```
BURN period power
```

where

`BURN`	keyword to indicate that burnup information follows.
period	the burnup period (in days) that will be transferred to a real CLE-2000 variable.
power	the power density (in kW/kg) that will be transferred to a real CLE-2000 variable.

The **(hstpar)** serves two purposes. First, it is used to define the names of the local and global parameters that may be used in our calculations as well as the values of these local parameters. In can also be used to extract from a HISTORY data structure the values of these parameters. The general form of this structure is:

Table 46: Structure **(hstpar)**

```
[[ NAMPAR valpar ]]
```

where

NAMPAR	name of a local or global parameter to process. The parameters specified before the keyword *CELLID* is read will be considered global otherwise they will be considered local.
valpar	real value for the local or global parameter to process. In the case where the `GET` option is activated, the history module will extract this parameter from the input data stream. In the case where the `PUT` option is activated, the history module will try to transfer this information into a real CLE-2000 variable.

3.18.1 Example

The history interface between the codes DRAGON and DONJON has been written as a new module in order to facilitate the access to the GANLIB utilities that manage the required hierarchical data structures. The resulting `HST:` module can be called both by DRAGON and DONJON.

The reactor model we will consider as an example is a 3–D model with an $x = 3$, $y = 3$ and $z = 3$ mesh. Here we will assume that the $x - y$ plane describes fuel channels. The z plane will be associated with the so-called fuel bundles. This choice is somewhat arbitrary, however it is useful if the refueling takes place in a specific direction as in a CANDU reactor. Here, a 2–bundle shift fueling strategy will be considered. To each fresh fuel cell introduced in the core the `HST:` module will associate a unique cell number between 1 and `Nc`, the maximum number of cells in the reactor. Most of the information associated with the fresh fuel cells will be extracted from a DRAGON BURNUP file or defined using variable local parameters. Each fresh fuel cell inserted in the core will also be associated with a specific fuel type. Each fuel type is defined as a unique initial fuel composition. The fuel management for the reactor, including burnup and refueling will be performed by the DONJON code. Here the `HST:` will interact with this code via the MAP data structure. Typically, each cell in the reactor will be burned inside DRAGON using the power provided in the `AX-SHAPE` record and the depletion time provided in the `BURNUP-BEG` record stored in the MAP structure. When refueling takes place some of the fuel cells will be extracted, other will be displaced from one position to another and finally new fresh fuel cells inserted. The fresh fuel cells properties will be extracted from the fuel types properties available on the HISTORY data structure.

In a coupled DRAGON/DONJON execution, the `HST:` module will be called at various points and for various reasons. The first call to `HST:` can be performed using:

```
MODULE HST: ;
*----
* Map data structure for initialization: MAP0
* History data structure : History
*----
SEQ_ASCII MAP0 ;
XSM_FILE History ;
XSM_FILE Reseau ;
*----
* Reactor parameters
* ncha = nunber of channels = 9
* nbun = nunber of bundles = 3
* nevo = nunber of evolution = 3
* nglo = nunber of global parameters = 1
* nloc = nunber of local parameters = 2
* bunl = bundle length in cm = 49.53 cm
*----
INTEGER ncha nbun nevo nglo nloc :=
9 3 3 1 2 ;
REAL bunl := 49.53 ;
*----
* Initialize History using MAP0
*----
Reseau := MAP0 ;
History := HST: Reseau ::
DIMENSIONS GLOBAL <<nglo>> LOCAL <<nloc>>
BUNDLES <<nbun>> <<bunl>>
CHANNELS <<ncha>> ;
```

Here, the HISTORY data structure will be stored in the XSM file `History`. One global and two local parameters are considered. No information about the name or the value of the global and local parameters will be available. This initialization procedure stores information only on the main level of the HISTORY data structure if the MAP data structure is not available. In this case the HISTORY is updated using a MAP data structure (in sequential ASCII file `MAP0`). The number of channels and bundles per channel are stored and compared with the same information in the MAP structure. For each bundle in the MAP, cell type and fuel type directories are constructed. The bundle powers and burnups available in MAP are used to generate the power rates in kW/kg and the depletion time in days required to reach the specified burnups. These values are stored in the HISTORY in the `PARAMBURNTAR` record. The fuel mass is mandatory for such calculation, thus the fuel weight is recovered from the MAP. If the HISTORY is in modification mode, the fuel weight is computed using the bundle lenght and the initial fuel density. Now, let us assume that a DRAGON calculation was performed for the cell located in bundle $j = 1$ of channel $i = 1$. We will also assume that these cells contain a single type of fuel. Here the moderator temperature `TMod` is a global parameter while the fuel (`TComb`) and coolant (`TCalo`) temperatures are considered local parameters. We assume that after the cell flux calculation a BURNUP data structure was generated using the following instructions:

```
*----
* Procedures for cell calculation: CellCalc
*----
PROCEDURE CellCalc ;
*----
* Global parameter: Tmod for moderator temperature
* Local parameters: TComb for fuel temperature
* TCalo for coolant temperature
*----
REAL TMod := 345.66 ;
REAL TComb TCalo := 941.29 560.66 ;
*----
* Initial burnup options for cell calculation
*----
REAL Power DeltaT := 31.9713 5.0 ;
*----
* Local data structures
*----
```

```
LINKED_LIST Burnup Edition ;
*----
* Execution control parameters
* icha = channel number = 1
* ibun = bundle number = 1
*----
INTEGER icha ibun := 1 1 ;
*----
* Perform cell calculation
*----
Burnup Edition := CellCalc Burnup ::
<<TComb>> <<TCalo>> <<TMod>>
<<Power>> <<DeltaT>> ;
```

Then, assuming that the history structure `HistXSM` was created using the options above, we can use

```
*----
* Update history structure
*----
History := HST: History Burnup ::
GET TMod <<TMod>>
CELLID <<icha>> <<ibun>>
GET TComb <<TComb>> TCalo <<TCalo>> ;
```

where no `idfuel` is given (see Table **??**), thus we have used the default value for `idfuel`= 1 to store in HISTORY the general information associated with fuel channel 1 and bundle 1. Here, the initial properties associated with fuel type 1 will be generated from the initial isotope densities in the BURNUP. For the `CELLID`, here `icha`= 1 and `ibun`= 1, the burnup information, isotope densities, depletion parameters and initial fuel density are stored in a /celldir/ directory. Moreover the power rate 31.9713 kW/kg and the depletion time 5.0 days are kept in the `PARAMBURNTAR` record.

A HISTORY data structure that contains the initial cell information can be updated using a MAP data structure:

```
*----
* Map data structure for refueling: MAP1
*----
SEQ_ASCII MAP1 ;
*----
* Refuel
*----
Reseau := MAP1 ;
History := HST: History Reseau ;
```

Here, new burnup power ratings will be stored in the HISTORY data structure reflecting the power distribution in the DONJON calculation. The refueling information available in the MAP structure will also be used to redistribute the fuel in the HISTORY structure at various cell location.

Finally the last option is to recover this information in DRAGON to perform a new series of cell calculations:

```
*----
* Local parameters
* Initial burnup options for cell calculation
* *A is after refueling
* *B is before refueling
*----
```

```
REAL TCombA TCaloA TCombB TCaloB ;
REAL PowerA DeltaTA PowerB DeltaTB ;
Burnup := HST: History ::
PUT TMod >>TMod<<
CELLID <<icha>> <<ibun>>
PUT BREFL BURN >>DeltaTB<< >>PowerB<<
TComb >>TCombB<< TCalo >>TCaloB<<
AREFL BURN >>DeltaTA<< >>PowerA<<
TComb >>TCombA<< TCalo >>TCaloA<< ;
IF DeltaTB 0.0 > THEN
*----
* Burn before refueling
*----
Burnup Edition := CellCalc Burnup ::
<<TCombB>> <<TCaloB>> <<TMod>>
<<PowerB>> <<DeltaTB>> ;
Edition := DELETE: Edition ;
ENDIF ;
*----
* Burn after refueling
*----
Burnup Edition := CellCalc Burnup ::
<<TCombA>> <<TCaloA>> <<TMod>>
<<PowerA>> <<DeltaTA>> ;
*----
* Update History
*----
History := HST: History Burnup ::
CELLID <<icha>> <<ibun>> ;
```

Note that here, there are two sets of local parameters that can be extracted from the history data structure, namely the before (`BREFL`) and the after (`AREFL`) refueling information. In the case of fresh fuel (single fuel description or a refueled bundle) extracting the before information is not required. However, if one uses the general procedure described above to extract the before and after information, one will be able to identify the new fuel bundles as well as the bundle that have not been moved in the core by the fact that $\Delta t = 0$ for burnup before refueling. For bundles that have been displaced in the core during refueling then $\Delta t > 0$.

4 CROSS-SECTION INTERPOLATION MODULES

4.1 The `CRE:` module

The `CRE:` module is used for the recovering and interpolation of nuclear properties from one or many COMPO objects, originated from the transport calculations using lattice code DRAGON. A resulting MACROLIB will be created (or updated) by the `CRE:` module, it will contain the nuclear properties of some selected reactor materials.

Two types of MACROLIB can be constructed using the `CRE:` module:

- A MACROLIB that will be constructed for the few reactor materials, namely for the devices and/or reflector properties. It can also be created for the few fuel regions defined in the reactor core. This MACROLIB is permitted to be updated for the new properties in the subsequent calls to the `CRE:` module.
- A fuel-map MACROLIB that will be constructed over the fuel lattice only. This MACROLIB will contain a set of interpolated fuel properties with respect to the burnup distribution over the fuel lattice and according to the interpolation option defined in the FMAP object. The total number of mixtures in the resulting MACROLIB will equal to the total number of fuel bundles.

Note that the `CRE:` module can be used only with the mono-parameter COMPO objects and the nuclear properties can be interpolated only with respect to the burnup data. In case of the MACROLIB construction from a multi-parameter database, the `NCR:` module should be used instead. In this case, the interpolation of nuclear properties can be made with respect to global and local parameters, if they were previously specified in the fuel-map (see Section **??**).

The `CRE:` module specifications are:

Table 47: Structure `CRE:`

{ *MACRO* `:=` `CRE:` [*MACRO*] [[*CPO*]] `::` **(desccre1)** |
MACFL `:=` `CRE:` [[*CPO*]] *FMAP* `::` **(desccre2)** }

where

MACRO	`character*12` name of the MACROLIB object to be created or updated for the few reactor material properties. Note that if *MACRO* appears on the RHS, the information previously stored in *MACRO* is kept.
CPO	`character*12` name of the COMPO object containing the mono-parameter database from transport calculations.
MACFL	`character*12` name of the fuel-map MACROLIB that will be created only for the fuel properties over the fuel lattice.
FMAP	`character*12` name of the FMAP object containing the fuel-map specification and burnup informations.
(desccre1)	structure describing the input data to the `CRE:` module when the FMAP object is not specified.

(desccre2)	structure describing the input data to the `CRE:` module for the fuel-map MACROLIB construction.

4.1.1 Input data for the `CRE:` module

Table 48: Structure **(desccre1)**

```
[ EDIT iprint ]
[ NMIX nmix ]
READ [[ COMPO CPO (descdata1) ]]
;
```

Table 49: Structure **(desccre2)**

```
[ EDIT iprint ]
READ [[ TABLE CPO (descdata2) ]]
;
```

where

`EDIT`	keyword used to set *iprint*.
iprint	integer index used to control the printing of information on screen: = 0 for no print; = 1 for minimum printing; larger values will produce increasing amounts of output.
`NMIX`	keyword used to define the number of material mixtures *nmix*. This data must be given only if *MACRO* is created and the FMAP object is not specified.
nmix	integer maximum number of reactor material mixtures, as defined in the reactor geometry.
`READ`	keyword used to read the MACROLIB specification from the input data file.
`COMPO`	keyword used to indicate a simple MACROLIB creation, i.e. according to the first calling specification when FMAP object is not specified.
`TABLE`	keyword used to indicate a fuel-map MACROLIB creation, i.e. according to the second calling specification with FMAP object specified.
CPO	`character*12` name of the selected COMPO object. This name must appear in the calling specification to the `CRE:` module.
(descdata1)	structure containing the interpolation specification if `COMPO` is the selected option.
(descdata2)	structure containing the interpolation specification if `TABLE` is the selected option.

Table 50: Structure **(descdata1)**

```
[[ MIX mix NAMDIR  [ DERIV ] [ UPS ]
    [ { I-BURNUP burn | T-BURNUP burn0 burn1 } ]
    [ MICRO { [[ HISO { conc | * } ]] | ALL } ]
ENDMIX ]]
```

Table 51: Structure **(descdata2)**

```
[[ MIX mix NAMDIR  [ DERIV ] [ UPS ]
    [ { TIMAV-BURN | INST-BURN | AVG-EX-BURN ivarty } ]
    [ MICRO { [[ HISO { conc | * } ]] | ALL } ]
ENDMIX ]]
```

where

`MIX`	keyword used to set the material mixture *mix*.
mix	integer identifier for the material mixture that will be included in the MACROLIB. The maximum number of identifiers permitted is *nmix* and the maximum value that *mix* may have is *nmix*. Note that if `TABLE` is the selected option, then *mix* identifies the fuel type as defined in the reactor geometry.
NAMDIR	`character*12` directory name in the *CPO* object from which the nuclear properties for material mixture *mix* are to be recovered.
`DERIV`	keyword used to compute the derivative of the MACROLIB information with respect to *burn* or *burn1* value. By default, the MACROLIB information is not differentiated.
`UPS`	keyword used to compute properties with no up-scattering contribution.
`TIMAV-BURN`	keyword used to compute time-averaged cross-section information. This option is available *only if* `TABLE` is the selected option. By default, the type of calculation (`TIMAV-BURN` or `INST-BURN`) is recovered from the *FMAP* object.
`INST-BURN`	keyword used to compute cross-section information at specific bundle burnups. This option is available *only if* `TABLE` is the selected option. By default, the type of calculation (`TIMAV-BURN` or `INST-BURN`) is recovered from the *FMAP* object.
`AVG-EX-BURN`	keyword used to compute the derivatives of cross-section information relative to the exit burnup of a single combustion zone. The derivatives are computed using Eq. (3.3) of Ref. **?**, written as $$\frac{\partial \bar{\Sigma}_x}{\partial B_j^{\rm e}} = \frac{1}{B_j^{\rm e}\,(B_{j,k}^{\rm eoc} - B_{j,k}^{\rm boc})}\left[-\int_{B_{j,k}^{\rm boc}}^{B_{j,k}^{\rm eoc}} dB\,\Sigma_x(B) + B_{j,k}^{\rm eoc}\,\Sigma_x(B_{j,k}^{\rm eoc}) - B_{j,k}^{\rm boc}\,\Sigma_x(B_{j,k}^{\rm boc})\right]$$ where $B_{j,k}^{\rm boc}$, $B_{j,k}^{\rm eoc}$, and $B_j^{\rm e}$ are the beginning of cycle burnup of bundle $\{j,k\}$, end of cycle burnup of bundle $\{j,k\}$ and exit burnup of channel j. This option is available *only if* `TABLE` is the selected option.

ivarty	index of the combustion zone for differentiation of cross-section information.
`I-BURNUP`	keyword used to perform a single interpolation and to set the burnup interpolation value *burn*.
burn	real interpolation value of the burnup, given in MW·day per tonne of initial heavy elements.
`T-BURNUP`	keyword used to perform a time-average MACROLIB evaluation between the burnup values *burn0* and *burn1*.
burn0	real initial value of the burnup, given in MW·day per tonne of initial heavy elements.
burn1	real final value of the burnup, given in MW·day per tonne of initial heavy elements.
`MICRO`	keyword used to set the number densities of the extracted isotopes present in the COMPO linked list or XSM file. By default, the extracted isotopes are not added to the resulting MACROLIB.
HISO	`character*12` name of an extracted isotope.
conc	user-defined real number density of the extracted isotope, given in 10^{24} particles per cm^3.
`*`	keyword used to indicate that the number density for the isotope *HISO* will be recovered from the COMPO object.
`ALL`	keyword used to indicate that all the number densities are to be recovered from the COMPO object.
`ENDMIX`	keyword used to indicate the end of data specification for the material mixture *mix*.

4.2 The `NCR:` module

This component of DONJON is dedicated to the interpolation of MICROLIB and MACROLIB data from a MULTICOMPO object, the reactor database produced by `COMPO:`. A set of *global* and/or *local parameters* are defined for each material mixture and used as multi-dimensional interpolation variables.

The calling specifications are:

Table 52: Structure **(NCR:)**

MLIB `:=` `NCR:` [{ *MLIB* | *MLIB2* }] *CPONAM1* [[*CPONAM2*]] [*MAPFL*] `::` **(ncr_data)**

where

MLIB	`character*12` name of a MICROLIB (type `L_LIBRARY`) or MACROLIB (type `L_MACROLIB`) containing the interpolated data. If this object also appears on the RHS, it is open in modification mode and updated. A MACROLIB object cannot be specified on the RHS.
MLIB2	`character*12` name of an optional MICROLIB object whose content is copied on *MLIB*.
CPONAM1	`character*12` name of the LCM object containing the MULTICOMPO data structure (`L_MULTICOMPO` signature).
CPONAM2	`character*12` name of an additional LCM object containing an auxiliary MULTICOMPO data structure (`L_MULTICOMPO` signature). This object is optional.
MAPFL	`character*12` name of the MAP object containing fuel regions description, global and local parameter information (burnup, fuel/coolant temperatures, coolant density, etc). Keyword `TABLE` is expected in **(ncr_data)**.
ncr_data	input data structure containing interpolation information (see Section **??**).

4.2.1 Interpolation data input for module `NCR:`

Table 53: Structure **(ncr_data)**

[`EDIT` *iprint*]
[`ALLX` *nbfuel*] [`RES`]
[{ `MACRO` | `MICRO` }] [{ `LINEAR` | `CUBIC` }] [`LEAK` *b2*]
[`NMIX` *nmixt*]
{ [[`COMPO` *CPONAM NAMDIR* **(descintf)**]]
 | [[`TABLE` *CPONAM NAMDIR* [*namburn*] **(descintf)**]] }
;

where

`EDIT`	keyword used to set *iprint*.

iprint	index used to control the printing in module `NCR:`. =0 for no print; =1 for minimum printing (default value).
`ALLX`	keyword used to register the region number of each isotope before merging. This option is useful if the same keyword has been specified in `EDI:` and `COMPO:` before.
nbfuel	number of fuel rings used for micro-depletion calculations.
`RES`	keyword indicating that the interpolation is done only for the microscopic cross sections and not for the isotopic densities. In this case, a RHS MICROLIB must be defined and the number densities are recovered from it. This option is useful for micro-depletion applications. **Important note:** It is possible to force interpolation of some isotopic densities with `RES` option if these isotopes are explicitely specified with a "`*`" flag after `MICRO` keyword in *descintf* input data structure (see Section **??**).
`MACRO`	keyword indicating that *MLIB* is a MACROLIB (default option).
`MICRO`	keyword indicating that *MLIB* is a MICROLIB. Object *MLIB* contains an embedded MACROLIB, but the CPU time required to obtain it is longer.
`LINEAR`	keyword indicating that interpolation of the MULTICOMPO uses linear Lagrange polynomials.
`CUBIC`	keyword indicating that interpolation of the MULTICOMPO uses the Ceschino method with cubic Hermite polynomials, as presented in Ref. **?** (default option).
`LEAK`	keyword used to introduce leakage in the embedded MACROLIB. This option should only be used for non-regression tests.
b2	the imposed buckling corresponding to the leakage.
`NMIX`	keyword used to define the maximum number of material mixtures. This information is required only if *MLIB* is created.
nmixt	the maximum number of mixtures (a mixture is characterized by a distinct set of macroscopic cross sections) the MACROLIB may contain. The default value is *nmixt* = 0 or the value recovered from *MLIB* if it appears on the RHS.
`COMPO`	keyword used to set *CPONAM* and to define each global and local parameter.
`TABLE`	keyword used to set *CPONAM* and to recover some global and local parameter from a MAP object named *MAPFL*.
CPONAM	`character*12` name of the LCM object containing the MULTICOMPO data structure where the interpolation is performed. This name must be set in the RHS of the **(NCR:)** data structure.
NAMDIR	access the MULTICOMPO structure of *CPONAM* from the sub-directory named *NAMDIR*. This value must be set equal to `'default'` if not previously defined by a `STEP UP` keyword in module `COMPO`.
namburn	name of the parameter for burnup (or irradiation) in the sub-directory named *NAMDIR*. This value is defined if option `TABLE` is set *and* if burnup (or irradiation) is to be considered as parameter.
descintf	input data structure containing interpolation information relative to the MULTICOMPO data structure named *CPONAM* (see Section **??**).

4.2.2 Defining local and global parameters

If a MAP object is defined on the RHS of structure **(ncr_data)**, and if the `TABLE` keyword is set, some information required to set the interpolation points is found in this object. In this case, the `NCR:` operator search the MULTICOMPO object for global or local parameters having an arbitrary name specified in the MAP object or set directly in this module. Note that any parameter's value set directly in this module prevails on a value stored in the MAP object.

Each instance of *descintf* is a data structure specified as

Table 54: Structure **(descintf)**

```
[[ MIX imix [ { FROM imixold | USE } ]
     [ { TIMAV-BURN | INST-BURN | AVG-EX-BURN ivarty } ]
     [[ { SET | DELTA | ADD } } [ { LINEAR | CUBIC } ] PARKEY { val1 | MAP } [ { val2 | MAP } ]
          [ REF [[ PARKEY { valref | SAMEASREF } ]] ENDREF ] ]]
     [ MICRO { ALL | ONLY } [[ HISO { conc | * } ]] ]
ENDMIX ]]
```

where

`MIX`	keyword used to set *imix*. Discontinuity factor information present in the Multicompo is interpolated as mixture 1 values.
imix	index of the mixture that is to be created in the MICROLIB and MACROLIB.
`FROM`	keyword used to set the index of the mixture in the MULTICOMPO object.
imixold	index of the mixture that is recovered in the MULTICOMPO object. By default, *imixold*= 1.
`USE`	keyword used to set the index of the mixture in the MULTICOMPO object equal to *imix*.
`TIMAV-BURN`	keyword used to compute time-averaged cross-section information. This option is available *only if* a *MAPFL* object is set. By default, the type of calculation (`TIMAV-BURN` or `INST-BURN`) is recovered from the *MAPFL* object.
`INST-BURN`	keyword used to compute cross-section information at specific bundle burnups. This option is available *only if* a *MAPFL* object is set. By default, the type of calculation (`TIMAV-BURN` or `INST-BURN`) is recovered from the *MAPFL* object.

`AVG-EX-BURN` keyword used to compute the derivatives of cross-section information relative to the exit burnup of a single combustion zone. The derivatives are computed using Eq. (3.3) of Ref. **?**, written as

$$\frac{\partial \bar{\Sigma}_x}{\partial B_j^{\rm e}} = \frac{1}{B_j^{\rm e}\,(B_{j,k}^{\rm eoc} - B_{j,k}^{\rm boc})} \left[- \int_{B_{j,k}^{\rm boc}}^{B_{j,k}^{\rm eoc}} dB\, \Sigma_x(B) + B_{j,k}^{\rm eoc}\, \Sigma_x(B_{j,k}^{\rm eoc}) - B_{j,k}^{\rm boc}\, \Sigma_x(B_{j,k}^{\rm boc}) \right]$$

where $B_{j,k}^{\rm boc}$, $B_{j,k}^{\rm eoc}$, and $B_j^{\rm e}$ are the beginning of cycle burnup of bundle $\{j,k\}$, end of cycle burnup of bundle $\{j,k\}$ and exit burnup of channel j. This option is available *only if* a *MAPFL* object is set. By default, the type of calculation (`TIMAV-BURN` or `INST-BURN`) is recovered from the *MAPFL* object.

ivarty index of the combustion zone for differentiation of cross-section information.

`SET`	keyword used to indicate a simple interpolation at *val1* or an averaging between *val1* and *val2*. The result $\sigma_{\rm ref}$ is also used as the reference value when the `ADD` is used. Note: see at the ending note of this section for a detailed description and examples.
`DELTA`	keyword used to indicate a delta-sigma calculation between *val2* and *val1* (i.e., $\Delta\sigma_{\rm ref} = \sigma_{\rm val2} - \sigma_{\rm val1}$ is computed). Note: see at the ending note of this section for a detailed description and examples.
`ADD`	keyword used to indicate a delta-sigma calculation between *val2* and *val1* is added to the reference value (i.e., $\Delta\sigma = \sigma_{\rm val2} - \sigma_{\rm val1}$ is used as contribution, $\sigma_{\rm ref} + \Delta\sigma$ or $\Delta\sigma_{\rm ref} + \Delta\sigma$ is returned). Note: see at the ending note of this section for a detailed description and examples.
`LINEAR`	keyword indicating that interpolation of the MULTICOMPO for parameter *PARKEY* uses linear Lagrange polynomials. It is possible to set different interpolation modes to different parameters. By default, the interpolation mode is set in Sect. **??**.
`CUBIC`	keyword indicating that interpolation of the MULTICOMPO for parameter *PARKEY* uses the Ceschino method with cubic Hermite polynomials, as presented in Ref. **?**. By default, the interpolation mode is set in Sect. **??**.
PARKEY	`character*12` user-defined keyword associated to a global or local parameter to be set.
val1	value of a global or local parameter used to interpolate. *val1* is the initial value of this parameter in case an average is required. *val1* can be an integer, real or string value.
val2	value of the final global or local parameter. By default, a simple interpolation is performed, so that *val2*=*val1*. *val2* is always a real value with *val2*≥*val1*.
`MAP`	keyword used to indicate that the value of parameter *val1* or the second value for the $\Delta\sigma$ calculation is recovered from *MAPFL*, i.e. the MAP object containing fuel regions description.
`REF`	keyword only available together with the `ADD` option. It is used to set all the other variable values when a Δ contribution is performed for one variable.
valref	value of the reference parameter, when it is directly given by the user. Note that there is no default value.
`SAMEASREF`	keyword used to specify that the reference value will be the same as in the refence case, i.e. for the $\sigma_{\rm ref}$ computation.
`ENDREF`	keyword only available together with the `ADD` option. It is used to specify that all the other variable values which are required are given.
`MICRO`	keyword used to set the number densities of some isotopes present in the MULTICOMPO object. The data statement "`MICRO ALL`" is used by default.
`ALL`	keyword to indicate that all the isotopes present in the MULTICOMPO object will be used in the MICROLIB and MACROLIB objects. Concentrations of these isotopes will be recovered from the MULTICOMPO object or set using the "*HISO conc*" data statement.
`ONLY`	keyword to indicate that only the isotopes set using the "*HISO conc*" data statement will be used in the MICROLIB and MACROLIB objects.
HISO	`character*8` name of an isotope.
conc	user-defined value of the number density (in 10^{24} particles per cm^3) of the isotope.

`*`	the value of the number density for isotope *HISO* is recovered from the MULTICOMPO object.
`ENDMIX`	end of specification keyword for the material mixture.

4.2.3 Interpolation in the parameter grid

The following example corresponds to a delta-sigma computation in mixture 1 corresponding to a perturbation. Note that in this case, the `MACROLIB` object may content negative cross-section.

```
MACROLIB := NCR: CPO ::
   EDIT 40 NMIX 1 MACRO COMPO CPO default
   MIX 1  !(* delta sigma contribution *)
          SET 'CELL' '3D'
          DELTA 'PITCH' 0.0 1.0
   ENDMIX
;
```

When the number of parameters used for the interpolation is increased, all the lattice computations corresponding to all the combinations of parameters may not be done for computation time reasons. In this case, some approximations may be required. The choice for the `SET`, `DELTA` and `ADD` is then dependent of the structure of the database (i.e. how the database grid of possibilities is filled). When a MAP object containing fuel regions description is used, the problem become even more complex, because values have to be automatically changed for all bundles. In order to clarify all the different possibilities and limitations dependently of the database structure, we will use a 3 parameter case. The paramaters are referenced by 'A', 'B' and 'C'. But before we explain the different cases, we want to remind that the interpolation factors are computed on each axis seperatly.

The first case corresponds to a complete grid, represented by a gray paralepiped on Fig. **??** and **??**. The figure **??** shows that the interpolated value in point V can be obtained directly without MAP object. For time-average (TA) computation, lets assume that the parameter 'B' represents the burnup (and keep this convention for other database structure also). In this case the figure **??** shows also that the direct interpolation can be done to compute an average value between the points V' and V. Note that the TA burnups are stored in the MAP object, and are then recovered automatically.

The second case corresponds to a partial grid where all the lattice computations have been perfomed for several pairs of parameters, which are represented as the gray rectangles on Fig. **??** and **??**. If we use the notations of Fig. **??** and **??**, the best estimate interpolated values, f, we can get are given by:
$f = f(V) \approx f(V_B) + (f(V_{BA}) - f(V_B)) + (f(V_{BC}) - f(V_B)) = f(V_{BC}) + (f(V_{BA}) - f(V_B)) = f(V_{BA}) + (f(V_{BC}) - f(V_B))$ for instataneous
$f = f(V', V) \approx f(V'_B, V_B) + (f(V'_{BA}, V_{BA}) - f(V'_B, V_B)) + (f(V'_{BC}, V_{BC}) - f(V'_B, V_B)) = f(V'_{BC}, V_{BC}) + (f(V'_{BA}, V_{BA}) - f(V'_B, V_B)) = f(V'_{BA}, V_{BA}) + (f(V'_{BC}, V_{BC}) - f(V'_B, V_B))$ for TA
where $f(.,.)$ represents the average value between two points.

The third case corresponds to a minimal grid, where the lattice computations have been perfomed only for one parameter variation at a time. In this case, the grid is represented by the thick gray lines on the axis on Fig. **??** and **??**. If we use the notations of Fig. **??** and **??**, the best estimate interpolated values, f, we can get are given by:
$f = f(V) \approx f(V_0) + (f(V_A) - f(V_0)) + (f(V_B) - f(V_0)) + (f(V_C) - f(V_0)) = f(V_B) + (f(V_A) - f(V_0)) + (f(V_C) - f(V_0))$ for instataneous
$f = f(V', V) \approx f(V'_B, V_B) + (f(V_A) - f(V_0)) + (f(V_C) - f(V_0))$ for TA
Note that the reference point (V_0 in the example) does not have to be the same for all parameters. Database structures such as represented on Fig **??** can also been used. In this case, we even have two choices for the Δf computation on axis 'A'.

The last case is in fact a mix of cases 2 and 3. The gray rectangle and the gray line on Fig. **??** and **??** reprensent where all the lattice computations have been performed. With the notations used on those

figures, one can write that the best estimate interpolated values, f, we can get are given by:
$f = f(V) \approx f(V_B) + (f(V_{BC}) - f(V_B)) + (f(V_A) - f(V_0)) = f(V_{BC}) + (f(V_A) - f(V_0))$ for instataneous
$f = f(V', V) \approx f(V'_B, V_B) + (f(V'_{BC}, V_{BC}) - f(V'_B, V_B)) + (f(V_A) - f(V_0)) = f(V'_{BC}, V_{BC}) + (f(V_A) - f(V_0))$ for TA
Note once again that the reference point (V_0 in the example) does not have to be the same for all parameters. Database structures such as represented on Fig **??** can also been used.

The input files will actually reflect the previous equations. However, they are different if the parameters are stored in a MAP object, *MAPFL*, or provided directly by the user. For the case of one point interpolation (i.e. instantaneous), the input files will be:

case	all parameters explicitly set	all parameters in MAP

GRID (Fig. **??**)	`MACROLIB := NCR: CPO ::`
	`   NMIX 1 MACRO`
	`   COMPO CPO default`
	`   MIX 1`
	`      SET ’A’ <<va>>`
	`      SET ’B’ <<vb>>`
	`      SET ’C’ <<vc>>`
	`   ENDMIX`
	`;`
		`MACROLIB := NCR: CPO FMAP ::`
		`   NMIX 1 MACRO`
		`   TABLE CPO default ’B’`
		`   MIX 1`
		`   ENDMIX`
		`;`

PLANE (Fig. **??**)	`MACROLIB := NCR: CPO ::`
	`   NMIX 1 MACRO`
	`   COMPO CPO default`
	`   MIX 1`
	`      SET ’A’ <<va>>`
	`      SET ’B’ <<vb>>`
	`      SET ’C’ <<vc0>>`
	`      ADD ’C’ <<vc0>> <<vc>>`
	`         REF  ’A’ <<va0>>`
	`              ’B’ <<vb>> ENDREF`
	`!or           ’B’ SAMEASREF ENDREF`
	`   ENDMIX`
	`;`
		`MACROLIB := NCR: CPO FMAP ::`
		`   NMIX 1 MACRO`
		`   TABLE CPO default ’B’`
		`   MIX 1`
		`      SET ’C’ <<vc0>>`
		`      ADD ’C’ <<vc0>> MAP`
		`         REF  ’A’ <<va0>>`
		`              ’B’ SAMEASREF ENDREF`
		`!or   SET ’A’ <<va0>>`
		`!or   ADD ’A’ <<va0>> MAP`
		`!or      REF  ’C’ <<vc0>>`
		`!or           ’B’ SAMEASREF ENDREF`
		`   ENDMIX`
		`;`

AXE (Fig. **??**)	`MACROLIB := NCR: CPO ::`
	`   NMIX 1 MACRO`
	`   COMPO CPO default`
	`   MIX 1`
	`      SET ’A’ <<va0>>`
	`      SET ’B’ <<vb>>`
	`      SET ’C’ <<vc0>>`
	`      ADD ’A’ <<va0>> <<va>>`
	`         REF  ’C’ <<vc0>>`
	`              ’B’ <<vb0>> ENDREF`
	`      ADD ’C’ <<vc0>> <<vc>>`
	`         REF  ’A’ <<va0>>`
	`              ’B’ <<vb0>> ENDREF`
	`   ENDMIX`
	`;`
		`MACROLIB := NCR: CPO FMAP ::`
		`   NMIX 1 MACRO`
		`   TABLE CPO default ’B’`
		`   MIX 1`
		`      SET ’A’ <<va0>>`
		`      SET ’C’ <<vc0>>`
		`      ADD ’A’ <<va0>> MAP`
		`         REF  ’C’ <<vc0>>`
		`              ’B’ <<vb0>> ENDREF`
		`      ADD ’C’ <<vc0>> MAP`
		`         REF  ’A’ <<va0>>`
		`              ’B’ <<vb0>> ENDREF`
		`   ENDMIX`
		`;`

continued on next page

case	all parameters explicitly set	all parameters in MAP

PLANE + AXE (Fig. ??)	MACROLIB := NCR: CPO ::
	   NMIX 1 MACRO
	   COMPO CPO default
	   MIX 1
	      SET ’A’ <<va0>>
	      SET ’B’ <<vb>>
	      SET ’C’ <<vc>>
	      ADD ’A’ <<va0>> <<va>>
	         REF ’C’ <<vc0>>
	              ’B’ <<vb0>> ENDREF
	   ENDMIX
	;
		MACROLIB := NCR: CPO FMAP ::
		   NMIX 1 MACRO
		   TABLE CPO default ’B’
		   MIX 1
		      SET ’A’ <<va0>>
		      ADD ’A’ <<va0>> MAP
		         REF ’C’ <<vc0>>
		              ’B’ <<vb0>> ENDREF
		   ENDMIX
		;

Table 55: NCR inputs for instantaneous cases

For the TA, the burnup variable has no other choice than to be stored in the MAP object, *MAPFL*. Then the input files will be:

case	only the burnup in MAP	all parameters in MAP

GRID (Fig. ??)	MACROLIB := NCR: CPO FMAP ::
	   NMIX 1 MACRO
	   TABLE CPO default ’B’
	   MIX 1
	      SET ’A’ <<va>>
	      SET ’C’ <<vc>>
	   ENDMIX
	;
		MACROLIB := NCR: CPO FMAP ::
		   NMIX 1 MACRO
		   TABLE CPO default ’B’
		   MIX 1
		   ENDMIX
		;

PLANE (Fig. ??)	MACROLIB := NCR: CPO FMAP ::
	   NMIX 1 MACRO
	   TABLE CPO default ’B’
	   MIX 1
	      SET ’A’ <<va>>
	      SET ’C’ <<vc0>>
	      ADD ’C’ <<vc0>> <<vc>>
	         REF ’A’ <<va0>>
	              ’B’ SAMEASREF ENDREF
	   ENDMIX
	;
		MACROLIB := NCR: CPO FMAP ::
		   NMIX 1 MACRO
		   TABLE CPO default ’B’
		   MIX 1
		      SET ’C’ <<vc0>>
		      ADD ’C’ <<vc0>> MAP
		         REF ’A’ <<va0>>
		              ’B’ SAMEASREF ENDREF
		!or SET ’A’ <<va0>>
		!or ADD ’A’ <<va0>> MAP
		!or REF ’C’ <<vc0>>
		!or ’B’ SAMEASREF ENDREF
		   ENDMIX
		;

continued on next page

case	only the burnup in MAP	all param. in MAP
AXE (Fig. ??)	`MACROLIB := NCR: CPO FMAP ::`	`MACROLIB := NCR: CPO FMAP ::`
	`   NMIX 1 MACRO`	`   NMIX 1 MACRO`
	`   TABLE CPO default 'B'`	`   TABLE CPO default 'B'`
	`   MIX 1`	`   MIX 1`
	`      SET 'A' <<va0>>`	`      SET 'A' <<va0>>`
	`      SET 'C' <<vc0>>`	`      SET 'C' <<vc0>>`
	`      ADD 'A' <<va0>> <<va>>`	`      ADD 'A' <<va0>> MAP`
	`         REF  'C' <<vc0>>`	`         REF  'C' <<vc0>>`
	`              'B' <<vb0>> ENDREF`	`              'B' <<vb0>> ENDREF`
	`      ADD 'C' <<vc0>> <<vc>>`	`      ADD 'C' <<vc0>> MAP`
	`         REF  'A' <<va0>>`	`         REF  'A' <<va0>>`
	`              'B' <<vb0>> ENDREF`	`              'B' <<vb0>> ENDREF`
	`   ENDMIX`	`   ENDMIX`
	`;`	`;`
PLANE + AXE (Fig. ??)	`MACROLIB := NCR: CPO FMAP ::`	`MACROLIB := NCR: CPO FMAP ::`
	`   NMIX 1 MACRO`	`   NMIX 1 MACRO`
	`   TABLE CPO default 'B'`	`   TABLE CPO default 'B'`
	`   MIX 1`	`   MIX 1`
	`      SET 'A' <<va0>>`	`      SET 'A' <<va0>>`
	`      SET 'C' <<vc>>`	`      ADD 'A' <<va0>> MAP`
	`      ADD 'A' <<va0>> <<va>>`	`         REF  'C' <<vc0>>`
	`         REF  'C' <<vc0>>`	`              'B' <<vb0>> ENDREF`
	`              'B' <<vb0>> ENDREF`	`   ENDMIX`
	`   ENDMIX`	`;`
	`;`	

Table 56: NCR inputs for TA cases

The following pictures correspond to the previous different examples:

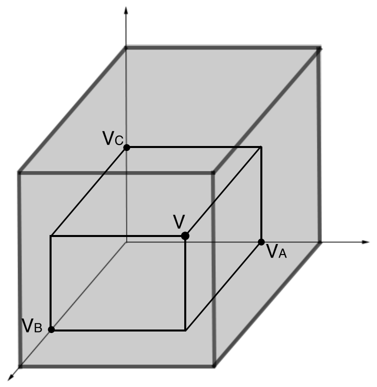

Figure 2: Complete grid, one point case

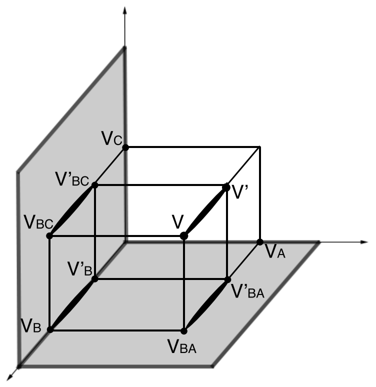

Figure 5: Partial grid, complete planes, TA case

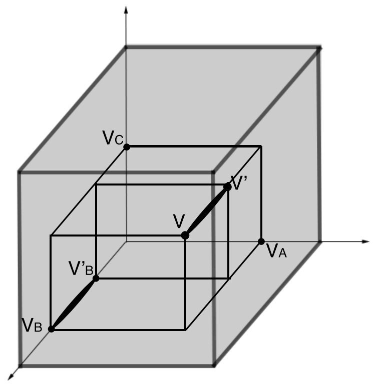

Figure 3: Complete grid, TA case

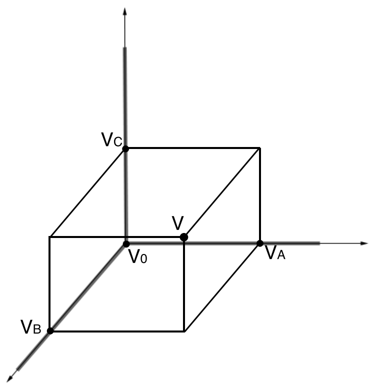

Figure 6: Partial grid, complete axis, one point case

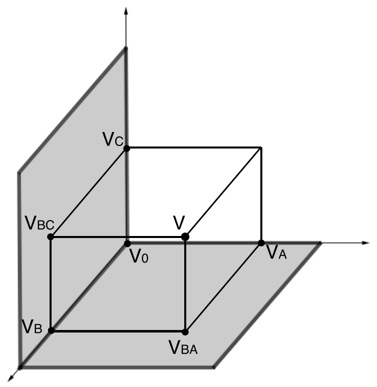

Figure 4: Partial grid, complete planes, one point case

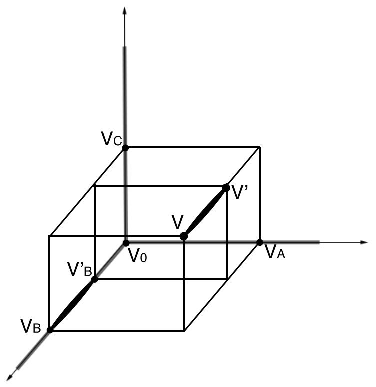

Figure 7: Partial grid, complete axis, TA case

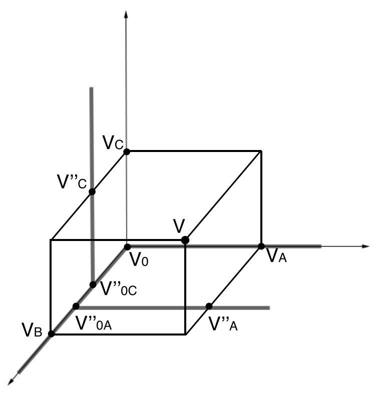

Figure 8: Partial grid, complete axis with another configuration, one point case

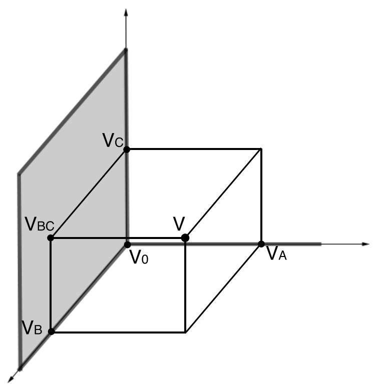

Figure 9: Partial grid, one complete plane and one complete axis, one point case

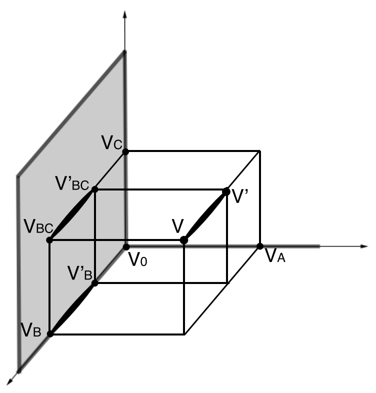

Figure 10: Partial grid, one complete plane and one complete axis, TA case

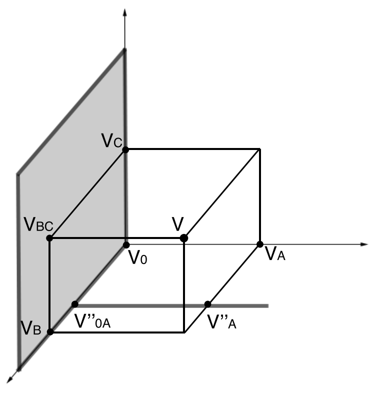

Figure 11: Partial grid, one complete plane and one complete axis with another configuration, one point case

4.3 The SCR: module

This component of DONJON is dedicated to the interpolation of MACROLIB data from a SAPHYB object, the reactor database produced by module `SAP:` in DRAGON or by module `SAPHYB:` in APOLLO2.[?] A set of *global parameters* are defined for each material mixture and used as multi-dimensional interpolation variables.

The calling specifications are:

Table 57: Structure **(SCR:)**

MLIB := `SCR:` [{ *MLIB* | *MLIB2* }] *SAPNAM1* [[*SAPNAM2*]] [*MAPFL*] :: **(scr_data)**

where

MLIB	`character*12` name of a MICROLIB (type `L_LIBRARY`) or MACROLIB (type `L_MACROLIB`) containing the interpolated data. If this object also appears on the RHS, it is open in modification mode and updated. A MACROLIB object cannot be specified on the RHS.
MLIB2	`character*12` name of an optional MICROLIB object whose content is copied on *MLIB*.
SAPNAM1	`character*12` name of the LCM object containing the SAPHYB data structure (`L_SAPHYB` signature).
SAPNAM2	`character*12` name of an additional LCM object containing an auxiliary SAPHYB data structure (`L_SAPHYB` signature). This object is optional.
MAPFL	`character*12` name of the MAP object containing fuel regions description, global parameter information (burnup, fuel/coolant temperatures, coolant density, etc). Keyword `TABLE` is expected in **(scr_data)**.
scr_data	input data structure containing interpolation information (see Section **??**).

Note: SAPHYB files generated by APOLLO2 don't have a signature. If such a SAPHYB is given as input to module `SCR:`, a signature must be included before using it. The following instruction can do the job:

```
Saphyb := UTL: Saphyb :: CREA SIGNATURE 3 = 'L_SA' 'PHYB' ' ' ;
```

4.3.1 Interpolation data input for module SCR:

Table 58: Structure **(scr_data)**

[`EDIT` *iprint*]
[`MEMORY`]
[`RES`]
[{ `MACRO` | `MICRO` }] [{ `LINEAR` | `CUBIC` }] [`LEAK` *b2*] [`EQUI` *TEXT4*]
[`NMIX` *nmixt*]
{ [[`SAPHYB` *SAPNAM* **(descints)**]] | [[`TABLE` *SAPNAM* [*namburn*] **(descints)**]] }
[**(descdepl)**]
;

where

`EDIT`	keyword used to set *iprint*.
iprint	index used to control the printing in module `SCR:`. =0 for no print; =1 for minimum printing (default value).
`MEMORY`	keyword activating a copy of the Saphyb into memory before performing interpolation. In some cases, this operation may reduce CPU resources in `SCR:`.
`RES`	keyword indicating that the interpolation is done only for the microscopic cross sections and not for the isotopic densities. In this case, a RHS MICROLIB must be defined and the number densities are recovered from it. This option is useful for micro-depletion applications. **Important note:** It is possible to force interpolation of some isotopic densities with `RES` option if these isotopes are explicitely specified with a "`*`" flag after `MICRO` keyword in *descints* input data structure (see Section **??**).
`MACRO`	keyword indicating that *MLIB* is a MACROLIB (default option).
`MICRO`	keyword indicating that *MLIB* is a MICROLIB. Object *MLIB* contains an embedded MACROLIB, but the CPU time required to obtain it is longer.
`LINEAR`	keyword indicating that interpolation of the SAPHYB uses linear Lagrange polynomials.
`CUBIC`	keyword indicating that interpolation of the SAPHYB uses the Ceschino method with cubic Hermite polynomials, as presented in Ref. **?** (default option).
`LEAK`	keyword used to introduce leakage in the embedded MACROLIB. This option should only be used for non-regression tests.
b2	the imposed buckling corresponding to the leakage.
`EQUI`	keyword used to select a SPH factor set in the Saphyb. By default, the cross sections and diffusion coefficients are not SPH-corrected.
TEXT4	character*4 user-defined keyword of the SPH factor set selected in the Saphyb. These sets are stored as local parameter information in the Saphyb.
`NMIX`	keyword used to define the maximum number of material mixtures. This information is required only if *MLIB* is created.
nmixt	the maximum number of mixtures (a mixture is characterized by a distinct set of macroscopic cross sections) the MACROLIB may contain. The default value is *nmixt* = 0 or the value recovered from *MLIB* if it appears on the RHS.
`SAPHYB`	keyword used to set *SAPNAM* and to define each global parameter.
`TABLE`	keyword used to set *SAPNAM* and to recover some global parameter from a MAP object named *MAPFL*.
SAPNAM	`character*12` name of the LCM object containing the SAPHYB data structure where the interpolation is performed. This name must be set in the RHS of the **(SCR:)** data structure.
namburn	name of the parameter for burnup (or irradiation). This value is defined if option `TABLE` is set *and* if burnup (or irradiation) is to be considered as parameter.
descints	input data structure containing interpolation information relative to the SAPHYB data structure named *SAPNAM* (see Section **??**).

(descdepl) input structure describing the depletion chain (see Section **??**). This input structure requires option `MICRO`. By default, the depletion chain data is not written in the output MICROLIB.

4.3.2 Defining global parameters

If a MAP object is defined on the RHS of structure **(scr_data)**, and if the `TABLE` keyword is set, some information required to set the interpolation points is found in this object. In this case, the `SCR:` operator search the SAPHYB object for global parameters having an arbitrary name specified in the MAP object or set directly in this module. Note that any parameter's value set directly in this module prevails on a value stored in the MAP object.

Each instance of *descints* is a data structure specified as

Table 59: Structure **(descints)**

```
[[ MIX imix [ { FROM imixold | USE } ]
    [ { TIMAV-BURN | INST-BURN | AVG-EX-BURN ivarty } ]
    [[ { SET | DELTA | ADD } } [ { LINEAR | CUBIC } ] PARKEY { val1 | MAP } [ { val2 | MAP } ]
        [ REF [[ PARKEY { valref | SAMEASREF } ]] ENDREF ] ]]
    [ MICRO { ALL | ONLY } [[ HISO { conc | * } ]] ]
ENDMIX ]]
```

where

`MIX`	keyword used to set *imix*. Discontinuity factor information present in the Saphyb is interpolated as mixture 1 values.
imix	index of the mixture that is to be created in the MICROLIB and MACROLIB.
`FROM`	keyword used to set the index of the mixture in the SAPHYB object.
imixold	index of the mixture that is recovered in the SAPHYB object. By default, *imixold*= 1.
`USE`	keyword used to set the index of the mixture in the SAPHYB object equal to *imix*.
`TIMAV-BURN`	keyword used to compute time-averaged cross-section information. This option is available *only if* a *MAPFL* object is set. By default, the type of calculation (`TIMAV-BURN` or `INST-BURN`) is recovered from the *MAPFL* object.
`INST-BURN`	keyword used to compute cross-section information at specific bundle burnups. This option is available *only if* a *MAPFL* object is set. By default, the type of calculation (`TIMAV-BURN` or `INST-BURN`) is recovered from the *MAPFL* object.
`AVG-EX-BURN`	keyword used to compute the derivatives of cross-section information relative to the exit burnup of a single combustion zone. The derivatives are computed using Eq. (3.3) of Ref. **?**, written as

$$\frac{\partial\bar{\Sigma}_x}{\partial B^{\rm e}_j} = \frac{1}{B^{\rm e}_j\,(B^{\rm eoc}_{j,k} - B^{\rm boc}_{j,k})}\left[-\int_{B^{\rm boc}_{j,k}}^{B^{\rm eoc}_{j,k}} dB\,\Sigma_x(B) + B^{\rm eoc}_{j,k}\,\Sigma_x(B^{\rm eoc}_{j,k}) - B^{\rm boc}_{j,k}\,\Sigma_x(B^{\rm boc}_{j,k})\right]$$

where $B^{\rm boc}_{j,k}$, $B^{\rm eoc}_{j,k}$, and $B^{\rm e}_j$ are the beginning of cycle burnup of bundle $\{j,k\}$, end of cycle burnup of bundle $\{j,k\}$ and exit burnup of channel j. This option is available

	only if a *MAPFL* object is set. By default, the type of calculation (`TIMAV-BURN` or `INST-BURN`) is recovered from the *MAPFL* object.
ivarty	index of the combustion zone for differentiation of cross-section information.
`SET`	keyword used to indicate a simple interpolation at *val1* or an averaging between *val1* and *val2*. The result $\sigma_{\rm ref}$ is also used as the reference value when the `ADD` is used. Note: see at the ending note of this section for a detailed description and examples.
`DELTA`	keyword used to indicate a delta-sigma calculation between *val2* and *val1* (i.e., $\Delta\sigma_{\rm ref} = \sigma_{\rm val2} - \sigma_{\rm val1}$ is computed). Note: see at the ending note of this section for a detailed description and examples.
`ADD`	keyword used to indicate a delta-sigma calculation between *val2* and *val1* is added to the reference value (i.e., $\Delta\sigma = \sigma_{\rm val2} - \sigma_{\rm val1}$ is used as contribution, $\sigma_{\rm ref} + \Delta\sigma$ or $\Delta\sigma_{\rm ref} + \Delta\sigma$ is returned). Note: see at the ending note of this section for a detailed description and examples.
`LINEAR`	keyword indicating that interpolation of the SAPHYB for parameter *PARKEY* uses linear Lagrange polynomials. It is possible to set different interpolation modes to different parameters. By default, the interpolation mode is set in Sect. **??**.
`CUBIC`	keyword indicating that interpolation of the SAPHYB for parameter *PARKEY* uses the Ceschino method with cubic Hermite polynomials, as presented in Ref. **?**. By default, the interpolation mode is set in Sect. **??**.
PARKEY	`character*12` user-defined keyword associated to a global parameter to be set.
val1	value of a global parameter used to interpolate. *val1* is the initial value of this parameter in case an average is required. *val1* can be an integer, real or string value.
val2	value of the final global parameter. By default, a simple interpolation is performed, so that *val2*=*val1*. *val2* is always a real value with *val2*≥*val1*.
`MAP`	keyword used to indicate that the value of parameter *val1* or the second value for the $\Delta\sigma$ calculation is recovered from *MAPFL*, i.e. the MAP object containing fuel regions description.
`REF`	keyword only available together with the `ADD` option. It is used to set all the other variable values when a Δ contribution is performed for one variable.
valref	value of the reference parameter, when it is directly given by the user. Note that there is no default value.
`SAMEASREF`	keyword used to specify that the reference value will be the same as in the refence case, i.e. for the $\sigma_{\rm ref}$ computation.
`ENDREF`	keyword only available together with the `ADD` option. It is used to specify that all the other variable values which are required are given.
`MICRO`	keyword used to set the number densities of some isotopes present in the SAPHYB object. The data statement "`MICRO ALL`" is used by default.
`ALL`	keyword to indicate that all the isotopes present in the SAPHYB object will be used in the MICROLIB and MACROLIB objects. Concentrations of these isotopes will be recovered from the SAPHYB object or set using the "*HISO conc*" data statement.
`ONLY`	keyword to indicate that only the isotopes set using the "*HISO conc*" data statement will be used in the MICROLIB and MACROLIB objects.

HISO	`character*8` name of an isotope.
conc	user-defined value of the number density (in 10^{24} particles per cm^3) of the isotope.
`*`	the value of the number density for isotope *HISO* is recovered from the SAPHYB object.
`ENDMIX`	end of specification keyword for the material mixture.

4.3.3 Depletion data structure

Part of the depletion data used in the isotopic depletion calculation (the fission yields and the radioactive decay constants) is recovered from the Saphyb file. Remaining depletion data is recovered from the input data structure **(descdepl)**. This data describes the heredity of the radioactive decay and the neutron activation chain.

Table 60: Structure **(descdepl)**

```
CHAIN
  [[ NAMDPL [ izae ]
    [[ reaction [ energy ] ]]
    [ { STABLE | FROM [[ { DECAY | reaction } [[ yield NAMPAR ]] ]] } ] ]]
ENDCHAIN
```

with:

`CHAIN`	keyword to specify the beginning of the depletion chain.
NAMDPL	`character*12` name of an isotope (or isomer) of the depletion chain that appears as a particularized isotope of the Saphyb.
izae	optional six digit integer representing the isotope. The first two digits represent the atomic number of the isotope; the next three indicate its mass number and the last digit indicates the excitation level of the nucleus (0 for a nucleus in its ground state, 1 for an isomer in its first exited state, etc.). For example, ^{238}U in its ground state will be represented by *izae*=922380.
reaction	`character*6` identification of a neutron-induced reaction that takes place either for production of this isotope, its depletion, or for producing energy. Example of reactions are following:

`NG`	indicates that a radiative capture reaction takes place either for production of this isotope, its depletion or for producing energy.
`N2N`	indicates that the following reaction is taking place: $$n +^A X_Z \rightarrow 2n +^{A-1} X_Z$$
`N3N`	indicates that the following reaction is taking place: $$n +^A X_Z \rightarrow 3n +^{A-2} X_Z$$

`N4N`	indicates that the following reaction is taking place: $$n +^A X_Z \rightarrow 4n +^{A-3} X_Z$$
`NP`	indicates that the following reaction is taking place: $$n +^A X_Z \rightarrow p +^A Y_{Z-1}$$
`NA`	indicates that the following reaction is taking place: $$n +^A X_Z \rightarrow^4 \mathrm{He}_2 +^{A-3} X_{Z-2}$$
`NFTOT`	indicates that a fission is taking place.

energy	energy (in MeV) recoverable per neutron-induced reaction of type *reaction*. If the energy associated to radiative capture is not explicitely given, it should be added to the energy released per fission. By default, *energy*=0.0 MeV.
`STABLE`	non depleting isotope. Such an isotope may produces energy by neutron-induced reactions (such as radiative capture).
`FROM`	indicates that this isotope is produced from decay or neutron-induced reactions.
`DECAY`	indicates that a decay reaction takes place for its production.
yield	branching ratio or production yield expressed in fraction.
NAMPAR	`character*12` name of the a parent isotope (or isomer) that appears as a particularized isotope of the Saphyb.
`ENDCHAIN`	keyword to specify the end of the depletion chain.

4.3.4 Interpolation in the parameter grid

The following example corresponds to a delta-sigma computation in mixture 1 corresponding to a perturbation. Note that in this case, the `MACROLIB` object may content negative cross-section.

```
MACROLIB := SCR: SAP ::
   EDIT 40 NMIX 1 SAPHYB SAP
   MIX 1  !(* delta sigma contribution *)
          SET 'CELL' '3D'
          DELTA 'PITCH' 0.0 1.0
   ENDMIX
;
```

When the number of parameters used for the interpolation is increased, all the lattice computations corresponding to all the combinations of parameters may not be done for computation time reasons. In this case, some approximations may be required. The choice for the `SET`, `DELTA` and `ADD` is then dependent of the structure of the database (i.e. how the database grid of possibilities is filled). When a MAP object containing fuel regions description is used, the problem become even more complex, because values have to be automatically changed for all bundles. In order to clarify all the different possibilities and limitations dependently of the database structure, we will use a 3 parameter case. The paramaters are referenced by 'A', 'B' and 'C'. But before we explain the different cases, we want to remind that the interpolation factors are computed on each axis seperatly.

The first case corresponds to a complete grid, represented by a gray paralepiped on Fig. **??** and **??**. The figure **??** shows that the interpolated value in point V can be obtained directly without MAP object. For time-average (TA) computation, lets assume that the parameter 'B' represents the burnup (and keep this convention for other database structure also). In this case the figure **??** shows also that the direct interpolation can be done to compute an average value between the points V' and V. Note that the TA burnups are stored in the MAP object, and are then recovered automatically.

The second case corresponds to a partial grid where all the lattice computations have been perfomed for several pairs of parameters, which are represented as the gray rectangles on Fig. **??** and **??**. If we use the notations of Fig. **??** and **??**, the best estimate interpolated values, f, we can get are given by:

$f = f(V) \approx f(V_B) + (f(V_{BA}) - f(V_B)) + (f(V_{BC}) - f(V_B)) = f(V_{BC}) + (f(V_{BA}) - f(V_B)) = f(V_{BA}) + (f(V_{BC}) - f(V_B))$ for instataneous

$f = f(V', V) \approx f(V'_B, V_B) + (f(V'_{BA}, V_{BA}) - f(V'_B, V_B)) + (f(V'_{BC}, V_{BC}) - f(V'_B, V_B)) = f(V'_{BC}, V_{BC}) + (f(V'_{BA}, V_{BA}) - f(V'_B, V_B)) = f(V'_{BA}, V_{BA}) + (f(V'_{BC}, V_{BC}) - f(V'_B, V_B))$ for TA

where $f(.,.)$ represents the average value between two points.

The third case corresponds to a minimal grid, where the lattice computations have been perfomed only for one parameter variation at a time. In this case, the grid is represented by the thick gray lines on the axis on Fig. **??** and **??**. If we use the notations of Fig. **??** and **??**, the best estimate interpolated values, f, we can get are given by:

$f = f(V) \approx f(V_0) + (f(V_A) - f(V_0)) + (f(V_B) - f(V_0)) + (f(V_C) - f(V_0)) = f(V_B) + (f(V_A) - f(V_0)) + (f(V_C) - f(V_0))$ for instataneous

$f = f(V', V) \approx f(V'_B, V_B) + (f(V_A) - f(V_0)) + (f(V_C) - f(V_0))$ for TA

Note that the reference point (V_0 in the example) does not have to be the same for all parameters. Database structures such as represented on Fig **??** can also been used. In this case, we even have two choices for the Δf computation on axis 'A'.

The last case is in fact a mix of cases 2 and 3. The gray rectangle and the gray line on Fig. **??** and **??** reprensent where all the lattice computations have been performed. With the notations used on those figures, one can write that the best estimate interpolated values, f, we can get are given by:

$f = f(V) \approx f(V_B) + (f(V_{BC}) - f(V_B)) + (f(V_A) - f(V_0)) = f(V_{BC}) + (f(V_A) - f(V_0))$ for instataneous

$f = f(V', V) \approx f(V'_B, V_B) + (f(V'_{BC}, V_{BC}) - f(V'_B, V_B)) + (f(V_A) - f(V_0)) = f(V'_{BC}, V_{BC}) + (f(V_A) - f(V_0))$ for TA

Note once again that the reference point (V_0 in the example) does not have to be the same for all parameters. Database structures such as represented on Fig **??** can also been used.

The input files will actually reflect the previous equations. However, they are different if the parameters are stored in a MAP object, *MAPFL*, or provided directly by the user. For the case of one point interpolation (i.e. instantaneous), the input files will be:

case	all parameters explicitly set	all parameters in MAP

GRID (Fig. **??**)	`MACROLIB := SCR: SAP ::`
	`   NMIX 1`
	`   SAPHYB SAP`
	`   MIX 1`
	`      SET 'A' <<va>>`
	`      SET 'B' <<vb>>`
	`      SET 'C' <<vc>>`
	`   ENDMIX`
	`;`
		`MACROLIB := SCR: SAP FMAP ::`
		`   NMIX 1`
		`   TABLE SAP 'B'`
		`   MIX 1`
		`   ENDMIX`
		`;`

PLANE (Fig. **??**)	`MACROLIB := SCR: SAP ::`
	`   NMIX 1`
	`   SAPHYB SAP`
	`   MIX 1`
	`      SET 'A' <<va>>`
	`      SET 'B' <<vb>>`
	`      SET 'C' <<vc0>>`
	`      ADD 'C' <<vc0>> <<vc>>`
	`         REF  'A' <<va0>>`
	`              'B' <<vb>> ENDREF`
	`!or           'B' SAMEASREF ENDREF`
	`   ENDMIX`
	`;`
		`MACROLIB := SCR: SAP FMAP ::`
		`   NMIX 1`
		`   TABLE SAP 'B'`
		`   MIX 1`
		`      SET 'C' <<vc0>>`
		`      ADD 'C' <<vc0>> MAP`
		`         REF  'A' <<va0>>`
		`              'B' SAMEASREF ENDREF`
		`!or   SET 'A' <<va0>>`
		`!or   ADD 'A' <<va0>> MAP`
		`!or      REF  'C' <<vc0>>`
		`!or           'B' SAMEASREF ENDREF`
		`   ENDMIX`
		`;`

AXE (Fig. **??**)	`MACROLIB := SCR: SAP ::`
	`   NMIX 1`
	`   SAPHYB SAP`
	`   MIX 1`
	`      SET 'A' <<va0>>`
	`      SET 'B' <<vb>>`
	`      SET 'C' <<vc0>>`
	`      ADD 'A' <<va0>> <<va>>`
	`         REF  'C' <<vc0>>`
	`              'B' <<vb0>> ENDREF`
	`      ADD 'C' <<vc0>> <<vc>>`
	`         REF  'A' <<va0>>`
	`              'B' <<vb0>> ENDREF`
	`   ENDMIX`
	`;`
		`MACROLIB := SCR: SAP FMAP ::`
		`   NMIX 1`
		`   TABLE SAP 'B'`
		`   MIX 1`
		`      SET 'A' <<va0>>`
		`      SET 'C' <<vc0>>`
		`      ADD 'A' <<va0>> MAP`
		`         REF  'C' <<vc0>>`
		`              'B' <<vb0>> ENDREF`
		`      ADD 'C' <<vc0>> MAP`
		`         REF  'A' <<va0>>`
		`              'B' <<vb0>> ENDREF`
		`   ENDMIX`
		`;`

continued on next page

case	all parameters explicitly set	all parameters in MAP

PLANE + AXE (Fig. ??)	MACROLIB := SCR: SAP ::
	   NMIX
	   SAPHYB SAP
	   MIX 1
	      SET 'A' <<va0>>
	      SET 'B' <<vb>>
	      SET 'C' <<vc>>
	      ADD 'A' <<va0>> <<va>>
	         REF 'C' <<vc0>>
	              'B' <<vb0>> ENDREF
	   ENDMIX
	;
		MACROLIB := SCR: SAP FMAP ::
		   NMIX 1
		   TABLE SAP 'B'
		   MIX 1
		      SET 'A' <<va0>>
		      ADD 'A' <<va0>> MAP
		         REF 'C' <<vc0>>
		              'B' <<vb0>> ENDREF
		   ENDMIX
		;

Table 61: SCR inputs for instantaneous cases

For the TA, the burnup variable has no other choice than to be stored in the MAP object, *MAPFL*. Then the input files will be:

case	only the burnup in MAP	all parameters in MAP

GRID (Fig. ??)	MACROLIB := SCR: SAP FMAP ::
	   NMIX 1
	   TABLE SAP 'B'
	   MIX 1
	      SET 'A' <<va>>
	      SET 'C' <<vc>>
	   ENDMIX
	;
		MACROLIB := SCR: SAP FMAP ::
		   NMIX 1
		   TABLE SAP 'B'
		   MIX 1
		   ENDMIX
		;

PLANE (Fig. ??)	MACROLIB := SCR: SAP FMAP ::
	   NMIX 1
	   TABLE SAP 'B'
	   MIX 1
	      SET 'A' <<va>>
	      SET 'C' <<vc0>>
	      ADD 'C' <<vc0>> <<vc>>
	         REF 'A' <<va0>>
	              'B' SAMEASREF ENDREF
	   ENDMIX
	;
		MACROLIB := SCR: SAP FMAP ::
		   NMIX 1
		   TABLE SAP 'B'
		   MIX 1
		      SET 'C' <<vc0>>
		      ADD 'C' <<vc0>> MAP
		         REF 'A' <<va0>>
		              'B' SAMEASREF ENDREF
		!or SET 'A' <<va0>>
		!or ADD 'A' <<va0>> MAP
		!or REF 'C' <<vc0>>
		!or 'B' SAMEASREF ENDREF
		   ENDMIX
		;

continued on next page

case	only the burnup in MAP	all param. in MAP

AXE (Fig. ??)	`MACROLIB := SCR: SAP FMAP ::`
	`   NMIX 1`
	`   TABLE SAP 'B'`
	`   MIX 1`
	`      SET 'A' <<va0>>`
	`      SET 'C' <<vc0>>`
	`      ADD 'A' <<va0>> <<va>>`
	`         REF  'C' <<vc0>>`
	`              'B' <<vb0>> ENDREF`
	`      ADD 'C' <<vc0>> <<vc>>`
	`         REF  'A' <<va0>>`
	`              'B' <<vb0>> ENDREF`
	`   ENDMIX`
	`;`
		`MACROLIB := SCR: SAP FMAP ::`
		`   NMIX 1`
		`   TABLE SAP 'B'`
		`   MIX 1`
		`      SET 'A' <<va0>>`
		`      SET 'C' <<vc0>>`
		`      ADD 'A' <<va0>> MAP`
		`         REF  'C' <<vc0>>`
		`              'B' <<vb0>> ENDREF`
		`      ADD 'C' <<vc0>> MAP`
		`         REF  'A' <<va0>>`
		`              'B' <<vb0>> ENDREF`
		`   ENDMIX`
		`;`

PLANE + AXE (Fig. ??)	`MACROLIB := SCR: SAP FMAP ::`
	`   NMIX 1`
	`   TABLE SAP 'B'`
	`   MIX 1`
	`      SET 'A' <<va0>>`
	`      SET 'C' <<vc>>`
	`      ADD 'A' <<va0>> <<va>>`
	`         REF  'C' <<vc0>>`
	`              'B' <<vb0>> ENDREF`
	`   ENDMIX`
	`;`
		`MACROLIB := SCR: SAP FMAP ::`
		`   NMIX 1`
		`   TABLE SAP 'B'`
		`   MIX 1`
		`      SET 'A' <<va0>>`
		`      ADD 'A' <<va0>> MAP`
		`         REF  'C' <<vc0>>`
		`              'B' <<vb0>> ENDREF`
		`   ENDMIX`
		`;`

Table 62: SCR inputs for TA cases

4.4 The `AFM:` module

The `AFM:` module is used to create an extended MACROLIB containing set of interpolated nuclear properties from a feedback model database.[?] The DATABASE information are obtained by previous DRAGON calculations using module `CFC:`.[?]

There are two possible utilizations:

- Construction of an extended MACROLIB for fuel properties directly from DATABASE information with respect to local parameters contained in the fuel map object or directly input.
- Construction of an extended MACROLIB containing only one set of cross sections derivated from the DATABASE information. Properties can be obtained for fuel or reflector.

The calling specifications are:

Table 63: Structure `AFM:`

MACRO `:=` `AFM:` [*MACRO*] *DBASE* [*MAPFL*] `::` **(descafm)**

where

MACRO	`character*12` name of the extended MACROLIB. The MACROLIB can be in modification mode.
DBASE	`character*12` name of the DATABASE object containing fuel properties with respect to local parameters.
MAPFL	`character*12` name of the MAP object containing fuel regions description and burnup informations. This file is only required when a MACRO is created for fuel area.
(descafm)	structure containing the data to module `AFM:`.

4.4.1 Input data to the `AFM:` module

Table 64: Structure **(descafm)**

{ `MAP` | `MCR` *mmix* } `INFOR` *NAMDB*
`DNAME` *ntyp* (*NAMTYP*(i), i=1,*ntyp*)
`REFT` (*imix*(i) *NAMTYP*(i), i=1,*ntyp*)
[`EDIT` *iprint*]
[`FIXP` { `INIT` | *pow* }]
[{ `PWF` | `NPWF` }]
[`TFUEL` *tfuel*]
[`TCOOL` *tcool*]
[`TMOD` *tmod*]

continued on next page

Structure **(descafm)** continued from last page

```
[ BORON nB ]
[ RDCL dcool ]
[ RDMD dmod ]
[ PUR purity ]
[ BURN bval ]
[ { XENON nXe | XEREF } ]
[ { NEP nNp | NREF } ]
[ SAM nSm ]
[ IMET imet ]
[ BLIN ]
;
```

where

`MAP`	keyword to specify that a MACROLIB for fuel properties will be computed.
`MCR`	keyword to specify that a MACROLIB containing only one non-zero mixture will be created.
mmix	maximum number of mixtures in the MACROLIB.
`INFOR`	keyword to specify the data base name.
NAMDB	`character*72` title of the database as it has been created.
`DNAME`	keyword to specify the number of fuel types and their names as stored in the data base.
ntyp	number of fuel types. For `MCR` option, *ntyp* must be 1.
NAMTYP(i)	`character*12` name of the directory where each fuel type information has been stored.
`REFT`	keyword to specify a number associated with a fuel type name.
imix(i)	fuel type index as specified for the fuel map or a non-zero mixture number for the single-property sc macrolib.
`EDIT`	keyword used to set *iprint*.
iprint	index used to control the printing in module `AFM:`. =0 for no print(default value); =1 for minimum printing; larger values produce increasing amounts of output.
`FIXP`	keyword used to set the power used for cross-section interpolation.
`INIT`	a distributed beginning-of-transient bundle power in kW is used. This power distribution has to be pre-calculated in the `FLPOW:` module using the `INIT` keyword.
pow	uniform bundle power in kW. If this data is omitted, the reference value in the data base is used or the bundle powers present in a MAP. The reference value is 615 kW if none were provided at the database computation time.
`PWF`	keyword used to activate power bundle feedback on fuel properties using powers recovered from `'BUND-PW'` record in *MAPFL*. This is the default option if `MAP` is selected.
`NPWF`	keyword used to desactivate `PWF` feedback. This is the only possible option if `MCR` is selected.

`TFUEL`	keyword used to set *tfuel*.
tfuel	fuel temperature in K. If this data is omitted and the bundle powers present in a MAP, fuel temperatures are computed with respect to powers. If this data is omitted and there is no bundle power, the reference value in the data base is used, where it is 941.29 K if none were provided at the database computation time.
`TCOOL`	keyword used to set *tcool*.
tcool	coolant temperature in K. If this data is omitted, the reference value in the data base is used. The reference value is 560.66 K if none were provided at the database computation time.
`TMOD`	keyword used to set *tmod*.
tmod	moderator temperature in K. If this data is omitted, the reference value in the data base is used. The reference value is 345.66 K if none were provided at the database computation time.
`BORON`	keyword used to set *nB*.
nB	Boron concentration in ppm. If this data is omitted, the reference value in the data base is used. The reference value is 0.0 ppm. See note below for inside equations.
`RDCL`	keyword used to set *dcool*.
dcool	coolant density in g/cm^3. If this data is omitted, the reference value in the data base is used. The reference value is 0.81212 g/cm^3 if none were provided at the database computation time.
`RDMD`	keyword used to set *dmod*.
dmod	moderator density in g/cm^3. If this data is omitted, the reference value in the data base is used. The reference value is 1.082885 g/cm^3 if none were provided at the database computation time.
`PUR`	keyword used to set *purity*.
purity	moderator purity in atm%. If this data is omitted, the reference value in the data base is used. The reference value is 99.911 atm% if none were provided at the database computation time.
`BURN`	keyword used to set *bval*. This option is valid only when `MCR` is used and can not be omitted.
bval	fuel burnup in MWd/t. This value must be positive.
`XENON`	keyword used to set *nXe*.
nXe	Xenon concentration in $10^{24}at/cm^3$. This concentration will be applied to every bundle.
`XEREF`	keyword used to specify that the Xenon concentrations as computed with DRAGON will be taken. If this option is omitted and MAP contains bundle fluxes, new Xenon concentrations will be computed and used.
`NEP`	keyword used to set *nNp*.
nNp	Neptunium concentration in $10^{24}at/cm^3$.

`XEREF`	keyword used to specify that the Neptunium concentrations as computed with DRAGON will be taken. If this option is omitted and MAP contains bundle fluxes, new Neptunium concentrations will be computed and used.
`SAM`	keyword used to set *nSm*.
nSm	Samarium concentration in $10^{24} at/cm^3$. If this data is omitted, bundle concentrations as computed by DRAGON is used.
`IMET`	keyword used to set *imet*.
imet	interpolation type for time-average calculations. $imet = 1$: using Lagrange approximations; $imet = 2$: using spline approximations; $imet = 3$: using Hermite approximations (default value).
`BLIN`	keyword used to linear interpolation for burnup instead of the Lagrangian interpolation method.

Note: The concentration of boron is provided in terms of $10^{24} at/cm^3$ in the database. However, the usual units are *ppm*(*wt*) of Boron. Thus, the input asks for *ppm* of Boron (n_B), and automatically transform the units into $10^{24} at/cm^3$ using the following equations:

$$\begin{aligned}
\rho_B(g/cm^3) &= n_B \ . \ \rho_{\text{water}}(g/cm^3) \\
&\text{and} \\
\rho_{\text{water}}(at/cm^3) &= 3\rho_{\text{water}}(molecule/cm^3) = \frac{3.N}{M_{\text{water}}}\rho_{\text{water}}(g/cm^3) \\
\rho_B(at/cm^3) &= \rho_B(molecule/cm^3) = \frac{N}{M_B}\rho(g/cm^3) \\
&\text{thus} \\
\rho_B(10^{24} at/cm^3) &= n_B \ . \ \frac{M_{\text{water}}}{3.M_B}\rho_{\text{water}}(10^{24} at/cm^3)
\end{aligned}$$

where M molar mass and N the Avogadro number.

They are many options on how to use the module `AFM:` for its different purposes. A compact summary is presented on Tab. **??**.

The Rozon correlation for fuel temperature as a function of bundle power is:

$$T_{\text{fuel}} = T_{\text{cool}} + 0.476\,P + 2.267\,P^2 \times 10^{-4}$$

where P is in kW and temperatures are in Kelvin.

Table 65: AFM options summary

Option	Keywords	Parameter values
MCR	REFT	Nominal values
	REFT + {TFUEL, TCOOL, ...}	Nominal values except for specified parameters
TAB	REFT	Nominal values except for TFUEL parameter which is computed according to the Rozon correlation using nominal power
	REFT + {TFUEL, TCOOL, ...}	Same as above except for specified parameters which will have a constant value
MAP with local parameters	REFT	Nominal values except for local parameters included in MAP
	REFT + {TFUEL, TCOOL, ...}	Same as above except for specified parameters which will have a constant value
MAP without local parameters	REFT	Nominal values except for TFUEL parameter which is computed according to the Rozon correlation if power distribution is available
	REFT + {TFUEL, TCOOL, ...}	Same as above except for specified parameters which will have a constant value

4.5 The `T16CPO:` module

The WIMS–AECL `Tape16` file is a FORTRAN sequential binary file which is used to transfer the results of a WIMS–AECL calculation to other applications.[?] The explicit contents of this file may vary from application to application since the output of most records to this file is controlled by the user who can activate specific keywords in the WIMS–AECL input file.

The standard CPO data structure used by the code DONJON is generally generated by the cell code DRAGON. This data structure can be stored on a FORTRAN direct access binary file in the form of a hierarchical data base. There is also the possibility to keep the contents of this data structure in memory (with the same hierarchical structure) for faster access. The structure of the data base is in the form of a list of material directories which contain burnup sub-directories. Inside each of these burnup sub-directories the isotopic contents of a mixture is described and the multigroup cross sections associated with a specific isotope are stored in individual sub-directories. Note that in this database the macroscopic cross sections associated with a mixture are stored in a default isotopic sub-directory.

The interface between the `Tape16` file and the CPO data structure should be written as a new module of the code DONJON in order to facilitate the access to the GANLIB utilities which manage the hierarchical data structures. This module will be called `T16CPO:`. The transfer of information from a `Tape16` format file to a CPO data structure will require the following DONJON instructions:

The `T16CPO:` module specifications for creating or updating a CPO data structure from a `Tape16` file are:

Table 66: Structure `T16CPO:`

`DONCPO := T16CPO: [ DONCPO ] WIMS16 ::` **(desct16cpo)** ;

where

`DONCPO`	name of data structure where the output CPO is stored. This can be a new data structure or an old data structure which will be updated.
(desct16cpo)	input specifications for the execution of the `T16CPO:` module.
;	end of record keyword. This keyword is used to delimit the part of the input data stream associated the current module.

In the following dataset

```
MODULE T16CPO: ;
SEQ_BINARY WIMS16 ;
LINKED_LIST DONCPO ;

DONCPO :=  T16CPO: WIMS16 ::
...
;
```

means that that the module will read the sequential binary file `WIMS16` file (in readonly mode) and create the CPO data structure `DONCPO` while the dataset

```
MODULE T16CPO: ;
SEQ_BINARY WIMS16 ;
LINKED_LIST DONCPO ;
```

```
DONCPO :=  T16CPO: DONCPO WIMS16 ::
...
;
```

means that the data structure DONCPO will be updated. The input instructions (replaced by ... here) should indicate what part of the information located on WIMS16 should be transferred to DONCPO and in what order.

4.5.1 Input data for the T16CPO: module

The input data structure **(desct16cpo)** will take the form:

Table 67: Structure **(desct16cpo)**

```
[ EDIT iprint ]
[ NMIX nmixt ]
[ CONDG ngcond (igc(i) , i=1,ngcond ) ]
[ LIST ]
[ MIX [[ MIXNAM [ { CELLAV | REGION noreg } ]
      [ RC [ nburn ] frstrec ]
      [[ NAMPER valref npert (valper(i), frstrec(i) , i=1,npert ) ]]
      [ MTMD [ valreft valrefd ] npert (valpert(i), valperd(i), frstrec(i) , i=1,npert ) ]
      ]] ]
```

where

EDIT	optional keyword used to modify the print level *iprint*.
iprint	index used to control the printing in this module. It must be set to 0 if no printing on the output file is required while values <10 will print general information about each record requested on Tape16 as well as other generic information pertinent to the T16CPO: module. Finally for values of *iprint*≥10, additional information required for debugging will be printed. The default value is *iprint*=1.
NMIX	optional keyword used to define the number of mixtures created on the CPO data structure.
nmixt	the maximum number of mixtures created. The default value is *nmixt*=1.
CONDG	optional keyword used to define the group structure for condensation. In the case where the CPO is to be updated, the information following CONDG must yield an energy group structure compatible with that already available on this data structure. If it is absent, the code will first try to use the CPO group structure (if available). Then, it will try to use the editing group structure corresponding to NGREAC on the following Tape16 record: REACTION␣␣, FLUX␣␣␣␣␣␣, NEL Finally, if everything else fails, it will select the main transport group structure corresponding to NGMTR on the following Tape16 record:

`WIMS␣␣␣␣␣␣`, `CONSTANT␣␣`, `NEL`

ngcond	the number of condensed groups required.
ilg	the last group number associated with each condensed group.
`LIST`	keyword to specify that the complete contents of `Tape16` must be listed on the output file.
`MIX`	keyword to specify that the remaining information will be associated with mixture properties definition.
MIXNAM	`character*6` name of the mixture to create or update on the CPO.
`CELLAV`	optional keyword to specify that cell averaged data will be taken from `Tape16`. This is the default option.
`REGION`	optional keyword to specify that regional data will be taken from `Tape16`. The default option is `CELLAV`.
noreg	region number associated with this material in `Tape16`.
`RC`	optional keyword to specify that the cross section taken from `Tape16` are at reference value. This information must be defined at least once for each mixture. It must also precede the definition of perturbation parameters.
nburn	number of consecutive burnup steps associated with mixture. The default value is *nburn*=1. We will assume that the same number of burnup steps is also available for the nuclear properties associated with the perturbed local parameters.
frstrec	first `Tape16` record number associated with this mixture.

NAMPER `character*2` name of the perturbation. Each perturbation is associated with a single local parameter. The values permitted for *NAMPER* are the following:

1. `FT` for fuel temperature
2. `MT` for moderator temperature
3. `MD` for moderator density
4. `MP` for moderator purity
5. `MB` for moderator boron
6. `CT` for coolant temperature
7. `CD` for coolant density
8. `CP` for coolant purity
9. `RT` for reflector temperature
10. `RD` for reflector density
11. `RP` for reflector purity

Note that these keywords are identical to those used in the Proc16 program.[?] Here the moderator, coolant and reflector can be D_2O, H_2O or any other mixture since DONJON is not aware of the compositions of these mixtures. In the case where many different `Tape16` files contains the reference and the individual perturbation effects, one must first define the reference case before updating the CPO using the `Tape16` files containing the perturbations.

valref reference value of the associated local parameter.

npert	number of local parameter perturbations.
valper	perturbed values of the local parameter.
`MTMD`	`character*4` name of perturbation associated with combine temperature and density changes effects. Note that this keyword is equivalent to the `MTS` keyword used in the Proc16 program.[?] In principle, any combined perturbations effects could be built from the catenation of two individual perturbations given in *NAMPER*.
valreft	reference temperature. This is required if either the `MT` or the `MD` perturbation is not defined.
valrefd	reference density. This is required if either the `MT` or the `MD` perturbation is not defined.
npert	number of simultaneous perturbations in moderator temperature and density.
valpert	perturbed values of the moderator temperature.
valperd	perturbed value of the moderator density.

The explicit name of the mixtures MIXDIR that will be stored on the main CPO directory will correspond to a catenation of *MIXNAM* and a perturbation name and an index i describing the perturbation order. It is created using the following FORTRAN instructions for the reference mixture:

`WRITE(MIXDIR,'(A6,A6)')` *MIXNAM*, `'RC␣␣␣␣'`

while for the i^{th} perturbed state associated with $NAMPER(J)$ we will use:

`WRITE(MIXDIR,'(A6,A2,A2,I2)')` *MIXNAM*, $NAMPER(J)$,`'␣␣'`, i

Finally, for the i^{th} perturbed state associated with the `MTMD` perturbation we will use:

`WRITE(MIXDIR,'(A6,A4,I2)')` *MIXNAM*, `'MTMD'`, i

Typically if the **(desct16cpo)** structure takes the form:

```
EDIT 0
NMIX 2
MIX
    Candu  RC 15 1
           FT 900.0 2 1100.0 16 1300.0 46
    Maple  RC 70
           RP 1.0 1 0.5 71
```

Then the first 15 cases stored on the `Tape16` file will correspond to a reference CANDU fuel with burnup. The reference fuel temperature is 900.0 K. The next 15 cases are for a fuel temperature of 1100.0 K. Finally cases 46 to 60 are for a fuel temperature of 1300.0 K. The Maple mixture will have no burnup. The reference Maple cross sections correspond to case 70, while case 71 contains the effect on the Maple fuel mixture cross sections of a 50 % reduction in reflector purity . As a result we will end up with a CPO data structure which contains 5 mixtures called respectively

```
Candu␣RC␣␣␣␣
Candu␣FT␣␣␣1
Candu␣FT␣␣␣2
Maple␣RC␣␣␣␣
Maple␣RP␣␣␣1
```

The beginning of a new case on `Tape16` will be identified by the presence of the record:

```
CELLAV␣␣␣␣,MODERATOR
```

in a `Tape16` file. Accordingly, the keyword `CELLAV` should be used in the WIMS–AECL run creating this file. In addition, if the `REGION` option is used in the `T16CPO:` input data structure, then it should also be used in the WIMS–AECL run creating this file.

4.6 The `D2P:` module

The objective of the `D2P:` module is to produce a file containing the macroscopic cross sections generated by the DRAGON5 lattice code and readable by the GenPMAXS software. This module makes possible the use of DRAGON-integrated XS into the PARCS core code. PARCS (*Purdue Advanced Reactor Core Simulator*) from the U.S. NRC [?] is a full 3D core code for the simulation of nuclear reactor steady state and transient behavior, at a specific burnup state.

The main objective of the `D2P:` module is to produce an output file with which can be accepted by GenPMAXS to produce the PMAXS file. In order to minimize the development in the GenPMAXS code, the choice has been made to reproduce an existing format already accepted by GenPMAXS. The HELIOS output format has been selected.

A Microlib is extracted from a Saphyb (or Multicompo) obtained by DRAGON (or APOLLO) in a previous calculation. The `D2P:` module extracts cross sections contained in this microlib and creates two files:

- an input file needed by GenPMAXS to produce a PMAXS (extention "`.inp`")
- a file containing data cross sections in HELIOS-like format (extention "`.dra`")

Note that the `D2P:` module is compatible with the last version of GenPMAXS (v6.1.3) and PARCS (v32m17 etc.).

4.6.1 The PMAXS format

The depletion capabilty of the PARCS code is reachable thanks to a specific format of cross section file, named PMAXS (*Purdue Macrosocopic Cross Section File*)[?]. This format is generated using the GenPMAXS code, based on output files of several lattice codes such as HELIOS, CASMO, TRITON, WIMS, CONDOR and SERPENT. This module intend to add DRAGON in this list.

The macroscopic cross sections are stored in the PMAXS file using partial derivatives as a function of state variables. Consequently the PMAXS format is a multi-dimentional table including burnup dependance. This format is a flexible way to obtain a more or less accurate meshing of cross sections. In addition of burnup, the list of state variables around which PMAXS is built is the following:

1. `CR`: control rod fraction
2. `DC`: density of coolant
3. `PC`: soluble poison concentration in coolant
4. `TF`: temperature of fuel
5. `TC`: temperature of coolant
6. `IC`: impurity of coolant
7. `DM`: density of moderator
8. `PM`: soluble poison concentration in moderator
9. `TM`: temperature of moderator
10. `IM`: impurity of moderator
11. `DN`: density difference between neighbor and current assembly
12. `BN`: burnup difference between neighbor and current assembly

These variables **should** be specified in this order. With the exception of burnup, each variable is optional. The following equation is used to compute a cross section Σ in the PMAXS formalism (including 4 state variables), with r the reference state and m the mid point between the reference state and the current node state $(CR, DC, \sqrt{TF}, TC)$:

$$\begin{aligned}\Sigma(CR, DC, \sqrt{TF}, TC) \;&=\; \Sigma^r(DC^r, \sqrt{TF}^r, TC^r) + \Delta DC\, \frac{\partial\Sigma}{\partial DC}\bigg|_{(CR,DC^m,\sqrt{TF}^r,TC^r)} + \\ &\Delta\sqrt{TF}\, \frac{\partial\Sigma}{\partial\sqrt{TF}}\bigg|_{(CR,DC,\sqrt{TF}^m,TC^r)} + \Delta TC\, \frac{\partial\Sigma}{\partial TC}\bigg|_{(CR,DC,\sqrt{TF},TC^m)}\end{aligned}$$

The PMAXS formalism and the procedure branching generation are completely described in the GenPMAXS manual [?].

The PMAXS file contains eight blocks, few of them are optional and others mandatory. A precise description of each block is proposed in the APPENDIX A of GenPMAXS manual [?]. In this section, a short description of blocks is given.

Block `XS CONTROL information (mandatory)`

The first block stores data reflecting the conditions in which cross sections are obtained and what kind of informations is contained in the PMAXS file. It is composed of five integers for the number of energy groups, of delay neuton groups etc. Then, fifteen logical flags indicates if the PMAXS contains the correponding data such as assembly discontinuity factor (ADF), Xe and Sm microscopic cross sections ... The block is ended by five lines of comments to be filled by the user.

```
GLOBAL_V       1  2 6 6 1 1 45 17 F F F F F F F F F F F F F F T
 Contents of T/H Invariant Variables(TIV) block and Cross Sections(XS) block
       TIV:
        XS:tr,ab,nf,kf/sct/
 2  Group value of each variable are put together in a line.
 Some variables(separated by ",") share a line,"/" means change line
 Generated by GenPMAXS-V6.1.1
```

Block `BRANCHES information (optional)`

This blocks identifies the state variables used for the branching and the correponding states for all branches.

```
STA_VAR    4 CR DC PC TF
BRANCHES   1   2   6  18  54
    RE   1     0.00000     0.71100  1000.00000   900.00000
    CR   1     1.00000     0.71100  1000.00000   900.00000
    CR   2     2.00000     0.71100  1000.00000   900.00000
    DC   1     0.00000     0.66100  1000.00000   900.00000
    DC   2     0.00000     0.75200  1000.00000   900.00000
    DC   3     1.00000     0.66100  1000.00000   900.00000
    DC   4     1.00000     0.75200  1000.00000   900.00000
    DC   5     2.00000     0.66100  1000.00000   900.00000
    DC   6     2.00000     0.75200  1000.00000   900.00000
    PC   1     0.00000     0.66100     0.00000   900.00000
    ...
```

In this example, the PMAXS cross sections depend on 4 state variables: CR, DC, PC and TF. There are 2 branches for control rods, 6 for coolant density, 18 for boron concentration, and 54 for fuel temperature.

Block `BURNUP information (optional)`

It contains the number of burnup sets and burnup points. Each cross sections will be repeated for each burnup points.

```
BURNUPS    1
   1  35 0.00000 0.00900 0.01900 0.07500 0.15000 0.50000 1.00000 2.00000 3.00000
         4.00000 6.00000 8.00000 10.0000 12.0000 14.0000 16.0000 18.0000 20.0000
         24.0000 28.0000 32.0000 36.0000 40.0000 44.0000 48.0000 52.0000 56.0000
         60.0000 64.0000 68.0000 72.0000 76.0000 80.0000 84.0000 86.0000
```

In the above example, one can observe a set of 35 burnup points from 0 to 86 GWd/t.

Block `XS SET identification (mandatory)`

In this block, the geometrical configuration of core reactor is specified and also the number of ADF in each group, the number of rod rows and columns in whole assembly, the rod lattice pitch etc. Some of these parameters have default values

```
XS_SET  00000001  1 1 1 17 17 3 1.44270 0.72135 0.72135 2.78613 0.73659 0.00016
        0.00000   0.00000
```

Block `HISTORY CASE identification (mandatory)`

This block describes the state variables values for all histories contained in the PMAXS file. The first parameter refers to the burnup set.

```
HISTORYC   1      0.00000     0.71100  1000.00000   900.00000
```

Block `T/H invariant variables (mandatory)`

It contains invariant variables, if the corresponding logical flag in block `XS CONTROL` is set to 'T'. The list of invariant variables, repeated for each burnup points, is:

- Chi spectra
- Yield of I, Xe, and Sm
- Beta of delayed neutron
- Lambda of delayed neutron
- Decay heat data

```
 1.00000E+00 0.00000E+00 5.13949E-08 2.17697E-06
 8.87406E-14 1.13375E-13 4.25558E-15
 2.68628E-04 1.43419E-03 1.39641E-03 3.23740E-03 1.43931E-03 5.99082E-04
 1.33535E-02 3.26045E-02 1.21056E-01 3.05531E-01 8.60559E-01 2.89034E+00
 ...
```

This block contains the necessary information for 1 burnup point and for each energy group `Chi,inV/YLD/Bet/Lam/`.

Block `XS data (mandatory)`

Cross sections in PMAXS file are listed for each burnup point and for each neutron energy group. Some cross sections are optional (see table below)

XS data block		
`STR`	*Transport cross section*	mandatory
`SAB`	*Absorption cross section*	mandatory
`SNF`	*Nu-fission cross section*	mandatory
`SKF`	*Kappa-fission cross section*	mandatory
`XENG`	*Microscopic capture cross section of Xenon*	optional
`SMNG`	*Microscopic capture cross section of Samarium*	optional
`SFI`	*Fission cross section*	optional
`DET`	*Detector response parameter*	optional
`SCT`	*Scattering cross section*	mandatory
`ADF`	*Assembly discontinuity factor*	optional
`DED`	*Direct energy deposition*	optional
`J1`	*J1 factors*	optional
`CDF`	*Corner discontinuity factor*	optional
`GFF`	*Group-Wise form function*	optional

4.6.2 General format of the module

The `D2P:` module can perform a sequence of phases related to the generation of a cross section format readable by GenPMAXS :

- PHASE 1 : recover input data from Saphyb and create output files
  1. recover information from a Saphyb file,
  2. store general information in output file,
  3. generate the GenPMAXS input file
- PHASE 2 : recover crosss section from microlib thanks to the `SCR:` (or `NCR:`) module and store in memory,
- PHASE 3 : store cross sections in output file.

The general format of the data for the `D2P:` module is the following:

Table 68: Structure `D2P:`

{ *HEL GEN INF* `:=D2P:` *INF* { *SAP* | *MCO* } `::` `PHASE` *1* [`EDIT` *iprint*] `(descphase1)`
| *GEN INF* `:=D2P:` *MIC INF GEN* { *SAP* | *MCO* } `::` `PHASE` *2* [`EDIT` *iprint*]
| *HEL GEN INF* `:=D2P:` *INF GEN HEL* `::` `PHASE` *3* [`EDIT` *iprint*]
}

In the DRAGON formalism, the Left-Hand-Side (LHS) is dedicated to the objects created or modified by the module, the Right-Hand-Side (RHS) is used for input objects, all parameters are passed to the module after the "::" delimiter .
where

HEL	`ascii file` Output file with HELIOS-like format, compulsory if *iphase*= 1 (in creation mode) or if *iphase*=3 (in modification mode).

GEN	`ascii file` Input file for running GenPMAXS, compulsory if *iphase*= 1 (in creation mode) or if *iphase*=2 or 3 (in modification mode)
INF	`LCM object` Block of data for the dialogue between different sequence of operations.
SAP	Saphyb object with cross sections to be extracted,compulsory if *iphase*= 1 or 2.
MCO	Multicompo object with cross sections to be extracted,compulsory if *iphase*= 1 or 2.
MIC	microlib object with cross sections for one burnup point,compulsory if *iphase*= 2.
`PHASE`	keyword used to set *iphase*.
iph	integer index used to control the current phase of `D2P:` module
`EDIT`	keyword used to set *iprint*.
iprint	integer index used to control the printing on screen: = 0 for no print (default value) ; = 1 for minimum printing ; for larger values of *iprint* everything will be printed.
`(descphase1)`	input data structure for PHASE 1 of this module

In the case where the current PHASE of the `D2P:` module is *iphase=1*, the (`descphase1`) takes the form:

Table 69: Structure (`descphase1`)

```
[NAMDIR mixdir ] [MIX imix ]
[PKEY (refnam(i) sapnam(i), i=1, npkey) ENDPKEY ]
FUEL {
BARR { DEF unrodded aicg aicn | USER unrodded (compo(i),i=1,ncompo) ENDBARR }
[GRID{SAP|DEF|
USER { GLOBAL (pkey(i),nval(i), i=1,npkey) ENDGLOBAL |
[NEW] ADD (pkey(i),nval(i),(val(j),j=1,nval(i)),i=1,npkey) ENDADD } } } ]
[ADF { DRA nadf (hadf(i), i=1,nadf) | GET | SEL} ]
[CDF DRA ncdf (hcdf(i), i=1,ncdf) ]
[GFF DRA ]
[ISOTOPES { [XE135 xenam ] [SM149 smnam ] [I135 inam ] [PM149 pmam ] } ENDISOTOPES ]
[YLD{REF| MAN (pkey(i),val(i), i=1,npkey) | FIX yldi yldxe yldpm} ]
}
|REFLECTOR
HELIOS [
  [FILE_CONT_1 ncols nrows part hm_dens bypass ]
  [FILE_CONT_2 (emin(g), g=1,ngroup) ]
  [FILE_CONT_3 vcool vwatr vmodr vcnrd vfuel vclad vchan ]
  [FILE_CONT_4 pitch xbe ybe ]
  [XS_CONT nside ncorner vfcm ] ]
GENPMAXS [
  [JOB_TIT jobtit ]
  [FILE_NAME fname ]
  [DERIVATIVE der ]
  [VERSION vers ]
  [COMMENT comment ]
```

continued on next page

Structure (`descphase1`) continued from last page

```
[JOB_OPT ladf lxes lded lj1f lchi lchd linv ldet lyld lcdf lgff lbet lamb ldec ]
[IUPS iups ]
[SFAC sfac ]
[BFAC bfac ]
[XESM xesmopt ] ]
```

where

`NAMDIR`	keyword used to set *mixdir* .
mixdir	name of sub-directory in Multicompo containing information to be recovered. Default *mixdir* = *'default'*.
`MIX`	keyword used to set *imix*
imix	index of the mixture in the *SAP* object which will be considered by the module. Default *imix*=1.
`PKEY`	keyword used to associate a name of PKEY in the `SAP` object to a type of state variable.
refnam	type of state variable. The possible *refnam* are : `BARR` for the control rod (-), `DMOD` for the density of coolant (g/cc), `CBOR` for the boron concentration (ppm), `TCOM` for the fuel temperature (C), `TMOD` for the moderator temperature (C), `BURN` for the burnup exposure (MWd/T). It is not necessary to specify all state variable names, only state variables with a different name compared to *refnam* are expected.
sapnam	name of state variable in the `SAP` object associated to *refnam* . Default values are *sapnam=refnam*. NB: If a state variable name is not correctly associated, an error will occur during processing.
`FUEL`	keyword used to indicate that the input *SAP* object contains cross sections for fuel assembly .
`BARR`	keyword used to associate an index of control rod in the *SAP* object to an index composition in PMAXS.
unrodded	index of control rod in the `SAP` object for the unrodded cross section. No default.
aicg	index of control rod in the `SAP` object for the aicg cross section. No default.
aicn	index of control rod in the `SAP` object for the aicn cross section. No default.
compo	index of control rod in the `SAP` object for the composition i.No default.
`ISOTOPES`	keyword used to associate a name of isotope in the `SAP` object to a specific isotope.
`XE135`	keyword used to indicate that the following record correponds to the name of Xe 135 in the `SAP` object .
xenam	name of Xe 135 isotope in the `SAP` (or `MCO`) object. Default *xenam* = *'XE135PF'*.
`SM149`	keyword used to indicate that the following record correponds to the name of Sm 149 in the `SAP` (or `MCO`) object .
smnam	name of Xe 135 isotope in the `SAP` (or `MCO`) object. Default *smnam* = *'SM149PF'*.

`I135`	keyword used to indicate that the following record correponds to the name of I 135 in the `SAP` (or `MCO`) object .
inam	name of Xe 135 isotope in the `SAP` (or `MCO`) object. Default *inam* = *'I135PF'*.
pmnam	name of Pm 149 isotope in the `SAP` (or `MCO`) object. Default *inam* = *'PM149PF'*.
`GRID`	keyword used to select the grid of state variables used for the branching generation of the `PMAXS` file.
`SAP`	keyword used to indicate that the meshing is the one used in the `SAP` (or `MCO`) object. Default option.
`USER`	keyword used to indicate that the meshing is defined by the user.
`GLOBAL`	keyword used to set a global meshing by defining for each desired state variables a number of points for the branching calculation.
`ADD`	keyword used to add a set of new points for the branching calculation. The new points are added to the meshing contained in the `SAP` (or `MCO`) object.
`NEW`	keyword used to indicate that the points contained in the `SAP` (or `MCO`) object are ignored, consequently only the set of points indicated using `ADD` will be considered for the branching calculation.
pkey	name of the state variable. If *pkey* does not correpond to any name in the `SAP` (or `MCO`) object, it will be ignored. It is not necessary to set all state variable contained in `SAP` (or `MCO`) , if a state variable is missing, the `SAP` (or `MCO`) meshing for this state variable will be considered. NB : the BARR parameter cannot be modified by the user.
nval	number of points for the state variable *pkey*. In the case `GLOBAL`, the *nval* points are obtained by splitting the pkey range from the first to the last values contained in the `SAP` (or `MCO`) object, otherwise it corresponds to the number of new points to be introduced in the meshing.
val	value to be added in the branching calculation corresponding to the *pkey(i)*. In the case where *pkey(i)* is `TCOM` or `TMOD`, the temperature must be in Celsius.
`DEF`	keyword used to call a default meshing : the values for `BARR` and `BURN` are extracted from Saphyb, four default values are considered for `DMOD`,`CBOR`, `TCOM` and `TMOD` (if exists). These values correspond to the first, mid and last values of the initial `SAP` (or `MCO`) meshing. This otpion is used if the number of banches in the Saphyb or defined by the user exceeds 1000.
`ADF`	keyword used to set the type of Assembly Discontonuity Factor to be recovered from the `SAP` (or `MCO`) object. NB : the *ladf* flag must be set to true.
`DRA`	Discontinuity factors are generated using the *DRAGON V5* procedure. Discontinuity factors could be Assembly Discontinuity Factors (ADF), Corner Discontinuity Factors (CDF) or Group-wise Form Function (GFF). Default option. NB: This option is available with Multicompo produced by the *DRAGON V5* lattice code using a 2-level flux calculation with the Method Of Characteristics. $$\mathrm{ADF}_{g,f} = \frac{\phi_g^{Het}}{\phi_g^{Hom}},$$ where g is the energy group, f, the assembly surface and ϕ_g^{Het},ϕ_g^{Hom} are the average surfacic homogene and heterogene fluxes in asssembly.

`GET`	Assembly discontinuity factors are generated using the *Generalized Equivalence Theory*. NB: This option is available with Saphyb produced by the *APOLLO2* lattice code using a 2-level flux calculation with the Method of Characteristics.

$$\mathrm{ADF}_{g,f} = \frac{\phi_g^{Het}}{\left(\frac{\pm J_g^{Net}\times h}{2\times D_g}\right) + \phi_g^{Hom}},$$

where g is the energy group, f, the assembly surface, ϕ_g^{Het},ϕ_g^{Hom} are the homogene and heterogene fluxes in asssembly,D_g , the diffusion coefficient, h the mesh dimension and J_g^{Net} the net average surfacic current.

`SEL`	Assembly discontinuity factors are generated using the *Selengut* normalization. This option is available with Saphyb prduced by the *APOLLO2* lattice code using a 2-level flux calculation with the Method of Characteristics.

$$\mathrm{ADF}_{g,f} = \frac{2\times\left(J_g^{+} + J_g^{-}\right)}{\left(\frac{\pm J_g^{Net}\times h}{2\times D_g}\right) + \phi_g^{Hom}},$$

where g is the energy group, f, the assembly surface,ϕ_g^{Hom} is the homogene fluxe in asssembly, D_g , the diffusion coefficient,h the mesh dimension and J_g^{+}, J_g^{-}, J_g^{Net} the incoming, outgoing, and net average surfacic currents.

nadf	number of the ADF-type boundary flux edit to be recovered from Multicompo. Allowed values *nadf* = 1 or 4.
hadf	name of the ADF-type boundary flux edit to be recovered from Multicompo. Default all *hadf* = $'FD\_B'$. In case *nadf*=4, the ADF values correspond to the following sides of the assembly: #1 for North , #2 for East, #3 for South and #4 for West. (same order as the SAPHYB)
`CDF`	keyword used to set the type of Assembly Discontonuity Factor to be recovered from the `SAP` (or `MCO`) object. NB : the *ladf* flag must be set to true.
ncdf	number of the CDF-type boundary flux edit to be recovered from Multicompo. Allowed values *nadf* = 1 or 4.
hcdf	name of the CDF-type boundary flux edit to be recovered from Multicompo. Default all *hadf* = $'FD\_C'$. In case *ncdf*=4, the CDF values correspond to the following corner of the assembly: #1 for North-West, #2 for South-West, #3 for South-East and #4 for North-East.
`GFF`	keyword used to set the type of Group Form Factor to be recovered from the `MCO` object. NB : the *lgff* flag must be set to true. In case of symmetry (*part*≥2), the numbering is different with the PARCS version (numbering as for CASMO starting from v3.2m18).
`YLD`	keyword used to set the type of fission yields to be recovered from the `SAP` (or `MCO`) object. NB : the *lyld* flag must be set to true.
`REF`	keyword used to indicate that fission yields are recovered from the reference branch calculation.
`MAN`	keyword used to indicate that fission yields are fixed by the user according to the local conditions specified.

`FIX`	keyword used to indicate that fission yields are fixed for all local conditions including burnup.
yldi	value for the iodine fission yield. Default: *yldi*=0.06386.
yldxe	value for the xenon fission yield. Default: *yldxe*=0.00228.
yldpm	value for the promethium fission yield. Default: *yldpm*=0.0113.
`REFLECTOR`	keyword used to indicate that the input *SAP* object contains cross sections for reflector.
`HELIOS`	keyword used to indicate that the input data for the *HEL* file will be set by the user.
`FILE_CONT_1`	keyword used to set the `FILE_CONT_1` block. See Ref. ?.
ncols	number of rod columns. Default: *ncols*=17.
nrows	number of rod rows. Default: *nrows*=17.
part	index for computed part of assembly (0/1/2/3 : whole/half/quarter/eight). By default, *part*=3.
hm_dens	initial heavy metal density ($g.cm^{-3}$). By default, *hm_dens*=2.78613. The initial heavy metal density can be computed as follow : $$hm\_dens = \left(\frac{d_{fuel} \times \pi \times r_{pellet} \times n_{pellet}}{(pitch \times nrows)^2} \right) \times \left(\frac{m_{hm}}{m_{fuel}} \right)$$ where d_{fuel} is the fuel density, r_{pellet} the radius of fuel pellet, n_{pin} the number of fuel rods, $pitch$ the rod lattice pitch, $nrows$ the number of rows in whole assembly, m_{hm} the mass of heavy metal, and m_{fuel} the total mass of fuel.
bypass	the saturated moderator density ($g.cm^{-3}$). By default, *bypass*=0.73659.
`FILE_CONT_2`	keyword used to set the lower energy limits of neutron groups.
emin	lower energy limits of neutron groups. Default: *emin*={ 6.2506E-01,1E-04 }
`FILE_CONT_3`	keyword used to set the `FILE_CONT_3` block (volume of regions). See Ref. ?.
vcool	volume of coolant. Default: *vcool*=2.4921E+02.
vwatr	volume of water. By default, *vwatr*=0.0000E+00.
vmodr	volume of moderator. Default: *vmodr*=2.4921E+02.
vcnrd	volume of control rods. By default, *vcnrd*=2.3020E+01.
vfuel	volume of fuel. By default, *vfuel*=1.4407E+02 .
vclad	volume of cladding. By default, *vclad*=4.5099E+01.
vchan	volume of channel. By default, *vchan*=4.5099E+01.
`FILE_CONT_4`	keyword used to set the `FILE_CONT_4` block. See Ref. ?.
pitch	rod lattice pitch (cm). Default: *pitch*=1.44270E+00.
xbe	starting position of first column rods (cm), i.e. water gap thickness. By default, *xbe*=7.21350E-01.

ybe	starting position of first row rods (cm), i.e. water gap thickness. By default, *ybe*=7.21350E-01.
`XS_CONT`	keyword used to set the `XS_CONT` block. See Ref. **?**.
nside	number of sides in assembly. Default: *nside*=1.
ncorner	number of corners in assembly. By default, *ncorner*=1.
vfcm	value of vfcm . By default, *vfcm*=5.32151E-01.
`GENPMAXS`	keyword used to indicate that the input data for the *GEN* file will be set by the user.
`JOB_TIT`	keyword used to set *jobtit*
jobtit	`character*16` name of the PMAXS file created by the `D2P:` module. Default: *jobtit*='`D2P.PMAX`'
`FILE_NAME`	keyword used to set *fname* .
fname	`character*12` name of the HELIOS-like file (*HEL*) created by the `D2P:` module. Default: *fname*='`HELIOS.dra`'
`DERIVATIVE`	keyword used to set *der*.
der	`character` (T/F) type of data in non-reference branches of output PMAXS file. If *der*='`T`', data are partial derivatives, otherwise it is raw cross sections. Default: *der*='`T`'.
`VERSION`	keyword used to set *vers*.
vers	the version of PARCS which will be used. If *vers*≥2.705, generate PMAXS for PARCS 2.71 or later versions, otherwise it is for PARCS 2.7 or earlier versions. Default: *vers*=3.0. The version number is as follows: v3.2m17 *vers*=3.217
`COMMENT`	keyword used to set a comment line.
comment	`character*40` Comment line for the user. Default: *comment*='`PWR CASE : UOX/MOX CORE FUEL`'.
`JOB_OPT`	keyword use to set logical flags, it indicates write or not write correponding data into PMAXS file. If the flag is '`F`', default values given in Ref. **?**, will be used in PARCS. For reflector case, all flags will be forced to '`F`', except for *ladf* and *linv*.
ladf	`character` (T/F) assembly discontinuity factor. Default: *ladf*='`F`'.
lxes	`character` (T/F) microscopic cross section of Xe and Sm. Default: *lxes*='`F`'.
lded	`character` (T/F) direct energy deposition fraction. Default: *lded*='`F`'.
lj1f	`character` (T/F) J1 factor for minimal critical power ratio. Default: *lj1f*='`F`'.
lchi	`character` (T/F) fission spectrum. Default: *lchi*='`F`'.
lchid	`character` (T/F) delay neutron fission spectrum. Default: *lchid*='`F`'.
linv	`character` (T/F) inverse neutron velocity. Must be '`T`' for transient. Default: *linv*='`F`'.
ldet	`character` (T/F) Detector response. Default: *ldet*='`F`'.
lyld	`character` (T/F) yield values of I, Xe,and Pm. Default: *lyld*='`F`'.
lcdf	`character` (T/F) corner discontinuity factor. Default: *lcdf*='`F`'.

lgff	`character` (T/F) group wise power form function. Default: *lgff*='`F`'.
lbet	`character` (T/F) Beta of delayed neutron. Default: *lbet*='`F`'.
lamb	`character` (T/F) Lambda of delayed neutron. Default: *lamb*='`F`'.
ldec	`character` (T/F) Decay heat beta and lambda. Default: *ldec*='`F`'.
`IUPS`	keyword used to set the treatment for up-scattering.
iups	(0/1) 0: keep up scatter XS, 1: remove up scatter XS, modify down scatter XS with with infinite medium spectrum. Default:*iups*=0.

$$\Sigma^{'}_{s,g\longleftarrow g'} = \Sigma^{'}_{s,g\longleftarrow g'} - \Sigma^{'}_{s,g'\longleftarrow g} \times \frac{\phi_g}{\phi_{g'}}$$

for $g' < g$

where $\phi_g, \phi_{g'}$ are the spectra flux either provided by DRAGON or infinite spectra computed in GenPMAXS.

`SFAC`	keyword used to set *sfac*. See Ref. **?**.
sfac	the scattering cross section factor. If *sfac* is different from 1, then the scattering cross section will be multiplied by *sfac*. Default: *sfac*=1.0.
`BFAC`	keyword used to set *bfac*. See Ref. **?**.
bfac	the multiplier for betas. If *bfac* is different from 1, then the betas will be multiplied by *bfac*. Default: *bfac*=1.0.
`XESM`	keyword used to set *xesmopt*. See Ref. **?**.
xesmopt	Compare k-inf in genpmaxs using 1:Pm/Sm data, 2: I/Xe data, 3: I/Xe/Pm/Sm data

5 THERMAL-HYDRAULICS MODULES

5.1 The `THM:` module

The `THM:` module is a simplified thermal-hydraulics module where the reactor is represented as a collection of independent channels with no cross-flow between them. Each channel is represented using 1D convection equations along the channel and 1D cylindrical equations for a single pin cell. A two-fluid homogeneous model is used. The `THM:` module is built around *freesteam*, an open source implementation of IAPWS-IF97 steam tables for light water.[?]. The `THM:` module works both in steady-state and in transient conditions and includes a subcooled flow boiling model based on the *Jens & Lottes correlation* [?] and on *Bowring's model* for two-phase homogeneous flows [?].

The 1D thermal-hydraulics equations are solved in each channel as a fonction of two *fixed* inlet conditions for the coolant velocity and temperature and one *fixed* outlet condition for the pressure.
The `THM:` module specification is:

Table 70: Structure `THM:`

THERMO MAPFL `:=` `THM:` [*THERMO*] *MAPFL* `::` **(descthm)**

where

THERMO	`character*12` name of the THERMO object that will be created or updated by the `THM:` module. Object THERMO contains thermal-hydraulics information set or computed by `THM:` in transient or in permanent conditions such as the distribution of the enthalpy, the pressure, the velocity, the density and the temperatures of the coolant for all the channels in the geometry. It also contains all the values of the fuel temperatures in transient or in permanent conditions according to the discretisation chosen for the fuel rods.
MAPFL	`character*12` name of the MAP object containing fuel regions description and local parameter informations.
(descthm)	structure describing the input data to the `THM:` module.

5.1.1 Input data to the `THM:` module

Table 71: Structure **(descthm)**

[`EDIT` *iprint*]
[`RELAX` *relax*]
[`TIME` *caltype timestep timeiter time*]
[`FPUISS` *fract*] [`CRITFL` *cflux*]

continued on next page

Structure **(descthm)** continued from last page

```
{ CWSECT sect flow | SPEED velocity }
ASSMB sass nbf nbg
INLET poutlet tinlet
RADIUS r1 r2 r3 r4
{ [ POROS poros ] [ PUFR pufr ] | [ CONDF ncond (kcond(k),k=0,ncond) [ INV inv ref ] unit ] }
[ CONDC ncond (kcond(k),k=0,ncond) unit ]
[ HGAP hgap ] [ HCONV hconv ] [ TEFF wteff ]
[ CONV maxit1 maxit2 maxit3 ermaxt ermaxc ]
[ RODMESH nb1 nb2 ]
[ FORCEAVE ]
[ { BOWR | SAHA } ]
[[ SET-PARAM PNAME pvalue ]]
;
```

where

`EDIT`	keyword used to set *iprint*.
iprint	integer index used to control the printing on screen: = 0 for no print; = 1 for minimum printing; larger values produce increasing amounts of output.
`RELAX`	keyword used to set the relaxation parameter *relax*.
relax	relaxation parameter selected in the interval 0 <*relax*≤ 1 and used to update the fuel (average and surface) temperature, coolant temperature and coolant density. The updated value is taken equal to (1−*relax*) times the previous iteration value plus *relax* times the actual iteration value. The default value is *relax*= 1.
`TIME`	keyword used to specify the type of calculation (steady-state or transient) performed by the `THM:` module and the temporal parameters in case of a transient calculation. By default, a steady-state calculation is performed.
caltype	integer value set to control the type of calculation that will be performed by the `THM:` module: =0 for steady-state; =1 for transient. The default value is 0.
timestep	real value set to the time step in case of a transient calculation. The default value is 0.0.
timeiter	integer value of the current time step index, used for transient calculations. The default value is 0.
time	real value of time in second, used for transient calculations. The default value is 0.0.
`FPUISS`	keyword used to specify the fraction of the power released in fuel. The remaining fraction is assumed to be released in coolant. The default value is 0.974.
fract	real value set to the fraction (f). Power densities released in coolant and fuel are computed as

$$
\begin{aligned}
Q_{\rm cool} &= (1-f)\,\frac{V_{\rm cool}+V_{\rm fuel}}{V_{\rm cool}}\,\frac{P_{\rm mesh}}{V_{\rm mesh}} \\
Q_{\rm fuel} &= f\,\frac{V_{\rm cool}+V_{\rm fuel}}{V_{\rm fuel}}\,\frac{P_{\rm mesh}}{V_{\rm mesh}}
\end{aligned}
$$

	where V_{cool} and V_{fuel} are coolant and fuel area computed from *sass*, *nbf*, *nbg*, *r3* and *r4*. The mesh power P_{mesh} and volume V_{mesh} are recovered from *MAPFL* object.
`CRITFL`	keyword used to specify the critical heat flux.
cflux	real value set to the critical heat flux in W/m^2. The default value is 2.0×10^6 W/m^2.
`CWSECT`	keyword used to specify the core coolant section and the coolant inlet flow.
sect	real value set to the core coolant section in m^2.
flow	real value set to the coolant flow in m^3/hr. This value doesn't include the by-pass flow. The inlet coolant velocity in m/s is computed as $$V = \frac{flow}{3600\, cwsect}.$$
`SPEED`	keyword used to specify the inlet coolant velocity.
velocity	real value set to the inlet coolant velocity in m/s.
`ASSMB`	keyword used to specify the assembly characteristics.
sass	real value set to the assembly surface in m^2. This value is equal to the square of an assembly side (including the water gap).
nbf	integer value set to the number of active fuel rods in a single assembly.
nbg	integer value set to the number of active guide tubes in a single assembly.
`INLET`	keyword used to specify the outlet pressure and inlet absolute temperature.
poutlet	real value set to the outlet coolant pressure in Pa. The pressure along each channel is assumed to be constant and equal to *poutlet* in permanent conditions.
tinlet	real value set to the inlet coolant absolute temperature in K.
`RADIUS`	keyword used to set the pin-cell radii.
r1	real value set to the fuel pellet radius in m.
r2	real value set to the internal clad rod radius in m.
r3	real value set to the external clad rod radius in m.
r4	real value set to the guide tube radius in m.
`POROS`	keyword used to set the oxyde porosity of fuel. Porosity affects some built-in correlations used to represent the heat conduction phenomenon in fuel.
poros	real value set to the oxyde porosity. The default value is 0.05.
`PUFR`	keyword used to set the plutonium mass enrichment of fuel. Plutonium enrichment affects some built-in correlations used to represent the heat conduction phenomenon in fuel.
pufr	real value set to the plutonium mass enrichment. The default value is 0.0.

`CONDF`	keyword used to set the fuel thermal conductivity as a function of local fuel temperature T_{fuel}. Fuel conductivity is computed as $$\lambda_{fuel} = \sum_{k=0}^{ncond} kcond(k) * (T_{fuel})^k + \frac{inv}{T_{fuel} - ref}$$ with λ_{fuel} in $W/m/K$ and T_{fuel} in the selected unit (Kelvin or Celsius). By default, built-in models are used, taking into account oxyde porosity and plutonium mass enrichment. Note that oxyde porosity and plutonium mass enrichment are ignored if this keyword is used.
ncond	integer value set to the degree of the conductivity polynomial.
kcond	real value set to the coefficient of the conductivity polynomial. $ncond + 1$ coefficients are expected.
unit	string value set to the unit of temperature T in the conductivity function. Can be either *CELSIUS* or *KELVIN*.
`INV`	keyword used to add an inverse term in the fuel conductivity function.
inv	real value set to the coefficient in the inverse term of fuel conductivity. The default value is 0.0 (i.e. no inverse term).
ref	real value set to the reference in the inverse term of fuel conductivity.
`CONDC`	keyword used to set the clad thermal conductivity as a function of local clad temperature T_{clad}. Clad conductivity is computed with the following polynomial $$\lambda_{clad} = \sum_{k=0}^{ncond} kcond(k) * (T_{clad})^k$$ with λ_{clad} in $W/m/K$ and T_{clad} in the selected unit (Kelvin or Celsius). By default, a built-in model is used.
`HGAP`	keyword used to set the heat exchange coefficient of the gap as a constant. By default, a built-in model is used.
hgap	real value set to the constant heat exchange coefficient of the gap in $W/m^2/K$.
`HCONV`	keyword used to set the heat transfer coefficient between clad and fluid as a constant. By default, this coefficient is computed using a built-in correlation.
hconv	real value set to the constant heat transfer coefficient between clad and fluid in $W/m^2/K$.
`TEFF`	keyword used to set the weighting factor in the effective fuel temperature approximation. The effective fuel temperature is used for the cross sections interpolations on fuel temperature.
wteff	real value W_{teff} set to the weighting factor in the effective fuel temperature. The effective fuel temperature is computed as $$T_{\text{eff}}^{\text{fuel}} = W_{\text{teff}} * T_{\text{surface}}^{\text{fuel}} + (1 - W_{\text{teff}}) * T_{\text{center}}^{\text{fuel}}$$ where $0 \leq W_{\text{teff}} \leq 1$, $T_{\text{surface}}^{\text{fuel}}$ is the temperature at the surface of the fuel pellet (K), and $T_{\text{center}}^{\text{fuel}}$ is the temperature at the center of the fuel pellet (K). By default, the Rowlands weighting factor $W_{\text{teff}} = \frac{5}{9}$ is used[?].

`CONV`	keyword used to set the convergence criteria for solving the conduction and the conservation equation.
maxit1	integer value set to the maximum number of iterations for computing the conduction integral. The default value is 50.
maxit2	integer value set to the maximum number of iterations for computing the center pellet temperature. The default value is 50.
maxit3	integer value set to the maximum number of iterations for computing the coolant parameters (mass flux, pressure, enthalpy and density) in case of a transient calculation. The default value is 50.
ermaxt	real value set to the maximum temperature error in K. The default value is 1 K.
ermaxc	real value set to the maximum relative error for parameters given by the resolution of flow conservation equations (pressure, velocity and enthalpy). The default value is 10^{-3}.
`RODMESH`	keyword used to set the radial discretization of pin-cells.
nb1	integer value set to the number of discretisation points in fuel. The default value is 5.
nb2	integer value set to the number of discretisation points in the whole pin-cell (fuel+cladding). The default value is 8.
`FORCEAVE`	keyword used to force the use of the average approximation during the fuel conductivity evaluation. By default, a rectangle quadrature approximation is used.
`BOWR`	keyword used to set a subcooling model based on the Jens & Lottes correlation[?] with the Bowring model[?] (default option).
`SAHA`	keyword used to set a subcooling model based on the Saha-Zuber correlation[?]. This option is recommended for BWR applications.
`SET-PARAM`	keyword used to indicate the input (or modification) of the actual values for a parameter specified using its *PNAME*.
`PNAME`	keyword used to specify *PNAME*.
PNAME	`character*12` name of a parameter.
pvalue	single real value containing the actual parameter's values. Note that this value will not be checked for consistency by the module. It is the user responsibility to provide the valid parameter's value which should be consistent with those recorded in the multicompo or Saphyb database.

6 OPTIMIZATION MODULES

This section is related to optimization capabilities available in Donjon and based on generalized perturbation theory.[?, ?] General information about the generalized perturbation theory can be found in Sect. 5.3 of Ref. **?**.

6.1 The `DLEAK:` module

The `DLEAK:` module is used to create a delta MACROLIB (type `L_MACROLIB`) with respect to leakage information. Derivatives of leakage-related information (recovered from the input MACROLIB) are stored in the `STEP` heteroneneous list components present in the output MACROLIB. Derivatives can be taken with respect to a leakage parameter itself ($D_{g,i}$ or $\Sigma_{1,g,i}$) or relative to factor μ in $\mu D_{g,i}$ or $\mu\Sigma_{1,g,i}$. Note that factor μ is not a SPH factor because it multiplies only leakage-related parameters. One component of the `STEP` heteroneneous list is created for each value of energy group g and for each value of mixture i.

The calling specifications are:

Table 72: Structure **(DLEAK:)**

DMACRO OPTIM `:=` `DLEAK:` *MACRO* `::` **(dleak_data)**

where

DMACRO	`character*12` name of a LCM object (type `L_MACROLIB`) containing the delta MACROLIB information. *DMACRO* is created by the module. A `STEP` heteroneneous list is present in *DMACRO*.
OPTIM	`character*12` name of a second LCM object (type `L_OPTIMIZE`) created by the module. Leakage-related parameters are saved in the the control variable record `'VAR-VALUE'` of *OPTIM* object. Input data defined in Sect. **??** is also saved in *OPTIM* object.
MACRO	`character*12` name of the LCM object (type `L_MACROLIB`) containing the input MACROLIB.
(dleak_data)	structure containing the data to module `DLEAK:` (see Sect. **??**).

6.1.1 Data input for module `DLEAK:`

Table 73: Structure **(dleak_data)**

[`EDIT` *iprint*]
`TYPE` { `DIFF` | `NTOT1` }
`DELTA` { `VALUE` | `FACTOR` }
[`MIXMIN` *ibm1*] [`MIXMAX` *ibm2*]
[`GRPMIN` *ngr1*] [`GRPMAX` *ngr2*]
;

where

`EDIT`	keyword used to set *iprint*.
iprint	index used to control the printing in module `DLEAK:`.
`TYPE`	keyword used to set the leakage parameter that is differentiated.
`DIFF`	differentiation with respect to diffusion coefficients.
`NTOT1`	differentiation with respect to P_1-weighted macroscopic total cross sections.
`DELTA`	keyword used to set the type of differentiation.
`VALUE`	differentiation with respect to the leakage parameter itself.
`FACTOR`	differentiation with respect to the correction factor μ.
`MIXMIN`	keyword used to set the first mixture where leakage parameters are differentiated. By default, the first mixture index is used.
ibm1	minimum mixture index where leakage parameters are differentiated.
`MIXMAX`	keyword used to set the last mixture where leakage parameters are differentiated. By default, the total number of mixtures in *MACRO* is used.
ibm2	maximum mixture index where leakage parameters are differentiated.
`GRPMIN`	keyword used to set the first energy group where leakage parameters are differentiated. By default, the first energy group index is used.
ngr1	minimum energy group index where leakage parameters are differentiated.
`GRPMAX`	keyword used to set the last energy group where leakage parameters are differentiated. By default, the total number of energy groups in *MACRO* is used.
ngr2	maximum energy group index where leakage parameters are differentiated.

6.2 The GRAD: module

The GRAD: module is designed to perform the following tasks:

- compute the gradients of the *system characteristics* using solutions of direct or adjoint fixed source eigenvalue problems. Here, we assume an optimization problem with *nvar* control variables and with *ncst* constraints. The total number of system characteristics is therefore equal to *ncst*+1.
- define options and parameters for the different method to solve the optimization problem. The non-linear optimization problem can be solved as a converging sequence of linear optimization problems with a quadratic constraint of the form

$$\sum_{i=1}^{nvar} \omega_i \left(\Delta x_i^{(n)}\right)^2 \le \left(S^{(n)}\right)^2$$

  where ω_i is a weight defined after keyword CST-WEIGHT and $\Delta x_i^{(n)}$ is a displacement for i–th control variable at iteration (n). The initial value of radius $S^{(1)}$ is defined after keyword OUT-STEP-LIM.
- reduces the radius $S^{(n)}$ of the quadratic constraint.

The calling specifications are:

Table 74: Structure GRAD:

```
OPTIM := GRAD: [ OPTIM ] DFLUX GPT :: (grad_data)
```

where

OPTIM	character*12 name of the OPTIMIZE object (L_OPTIMIZE signature) containing the optimization informations. Object *OPTIM* must appear on the RHS to be able to updated the previous values.
DFLUX	character*12 name of the FLUX object (L_FLUX signature) containing a set of solutions of fixed-source eigenvalue problems.
GPT	character*12 name of the GPT object (L_GPT signature) containing a set of direct or adjoint sources.
(grad_data)	structure containing the data to the module GRAD: (see Sect. ??).

6.2.1 Data input for module GRAD:

Table 75: Structure grad_data

```
[ EDIT iprint ]
```

continued on next page

Structure **grad_data** continued from last page

```
[ METHOD { SIMPLEX | LEMKE | MAP | AUG-LAGRANG | PENAL-METH } ]
[ OUT-STEP-LIM sr ]
[ OUT-STEP-EPS ε_ext ] [ INN-STEP-EPS ε_inn ]
[ CST-QUAD-EPS ε_quad ]
[ { MAXIMIZE | MINIMIZE } ]
[ STEP-REDUCT { HALF | PARABOLIC } ]
[ VAR-VALUE ( control(i), i=1,nvar ) ] [ VAR-WEIGHT ( weight(i), i=1,nvar ) ]
[ VAR-VAL-MIN { ( vecmin(i), i=1,nvar ) | ALL varmin ]
[ VAR-VAL-MAX { ( vecmax(i), i=1,nvar ) | ALL varmax ]
[ FOBJ-CST-VAL ( funct(i), i=1,ncst+1 ) ]
[ CST-TYPE ( type(i), i=1,ncst ) ] [ CST-OBJ ( cstval(i), i=1,ncst ) ]
[ CST-WEIGHT ( cstw(i), i=1,ncst ) ]
;
```

where

`EDIT`	keyword used to set *iprint*.
iprint	index used to control the printing in module.
`METHOD`	keyword used to define the quasi-linear programming method. **Note:** If the general Lemke method is used, the quadratic constraint must be active. The strategy consists to proceed in two steps: • At first step, the linear programming problem (i. e., without the quadratic constraint) is solved and the control-variable displacement is computed. If this displacement is less than the radius of the quadratic constraint, the step one solution is accepted and step two is not performed. If this displacement is greater than the radius of the quadratic constraint, the step one solution is rejected and step two is performed. Step one can be solved with the SIMPLEX method or with the linear LEMKE method. • At step two, the general LEMKE method is used to find the correct solution. The general Lemke method is based on a parametric linear complementarity principle, as explained in Ref. **?**.
`SIMPLEX`	keyword used to specify that the SIMPLEX method will be used at step one and the general LEMKE method at step two.
`LEMKE`	keyword used to specify that the linear LEMKE method will be used at step one and the general LEMKE method at step two.
`MAP`	keyword used to specify that the MAP method will be used. The quadratic constraint is linearized and a converging sequence of SIMPLEX calculations is performed.
`AUG-LAGRANG`	keyword used to specify that the augmented Lagrangian method will be used.
`PENAL-METH`	keyword used to specify that the penalty method will be used.
`OUT-STEP-LIM`	keyword used to set the initial radius of the quadratic constraint (default value is *sr* = 1.0).
sr	initial radius of the quadratic constraint (real).

`OUT-STEP-EPS`	keyword used to set the tolerance of outer iteration convergence inside module `PLQ:`.
ϵ_{ext}	tolerance value (real).
`INN-STEP-EPS`	keyword used to set the tolerance used within the SIMPLEX or LEMKE method.
ϵ_{inn}	tolerance value (real).
`CST-QUAD-EPS`	keyword to set the convergence parameter *epsilon4* for the radius of the quadratic constraint inside module `GRAD:`.
ϵ_{quad}	tolerance for convergence of the radius of the quadratic constraint (real).
`MAXIMIZE`	keyword used to specify that the optimization problem will be a maximization.
`MINIMIZE`	keyword used to specify that the optimization problem will be a minimization (default).
`STEP-REDUCT`	keyword used to define the method of the reduction of the outer step.
`HALF`	keyword used to specify that the step will be reduced by a factor of 2.
`PARABOLIC`	keyword used to specify that the step will be reduced with the parabolic method.
`VAR-VALUE`	keyword to specify the values of the control variables. These values can also be set in a previous call to module `GRAD:` or set in another module.
control	array containing *nvar* real values.
`VAR-WEIGHT`	keyword to specify the values of the control variable weights in the quadratic constraint. All weights are set to 1.0 by default.
weight	array containing *nvar* real values.
`VAR-VAL-MIN`	keyword to specify the minimum values of the control variables. These values can also be set in a previous call to module `GRAD:`.
vecmin	array containing *nvar* real values.
varmin	single real value used for all control variables.
`VAR-VAL-MAX`	keyword to specify the maximum values of the control variables. These values can also be set in a previous call to module `GRAD:`.
vecmax	array containing *nvar* real values.
varmax	single real value used for all control variables.
`FOBJ-CST-VAL`	keyword to specify the value of the objective function followed by the actual values of the constraints. These values can also be set in a previous call to module `GRAD:` or set in another module.
funct	array containing *ncst*+1 real values.
`CST-TYPE`	keyword to specify the relation types of the constraints. These values can also be set in a previous call to module `GRAD:`.
type	array containing *ncst* integer values. These values are: $= -1$ for $\geq$, $= 0$ for equalily and $= 1$ for $\leq$.
`CST-OBJ`	keyword to specify the RHS values of the constraints. These values can also be set in a previous call to module `GRAD:`.

cstval	array containing *ncst* real values.
`CST-WEIGHT`	keyword to specify the weights (or penalties) of the constraints. These weights are not used with Lemke or MAP methods. These values can also be set in a previous call to module `GRAD:`.
cstw	array containing *ncst* real values.

6.3 The `PLQ:` module

The `PLQ:` module is used to solve the linear programming problem with a quadratic constraint. The gradients of the *system characteristics* are calculated with module `GRAD:`. The options and parameters for the different method to solve the optimization problem are also defined in module `GRAD:`.

The calling specifications are:

Table 76: Structure `PLQ:`

```
OPTIM := PLQ: [ OPTIM ] :: (plq_data)
```

where

OPTIM	`character*12` name of the OPTIMIZE object (`L_OPTIMIZE` signature) containing the optimization informations. Object *OPTIM* must appear on the RHS to be able to updated the previous values.
(plq_data)	structure containing the data to the module `PLQ:` (see Sect. **??**).

6.3.1 Data input for module `PLQ:`

Table 77: Structure `plq_data`

```
[ EDIT iprint ]
[ WARNING-ONLY ]
CALCUL-DX [ NO-STORE-OLD ]
[ COST-EXTRAP >> ecost << ]
[ CONV-TEST >> l_conv << ]
;
```

where

`EDIT`	keyword used to set *iprint*.
iprint	index used to control the printing in module.
`WARNING-ONLY`	keyword used to specify that only a warning will be used when no valid previous decision vectors can be recall in case of error of the mathematical programming.
`CALCUL-DX`	keyword used to specify that the new step will be calculated.
`NO-STORE-OLD`	keyword used to specify that the old value of decision variables and gradients will not be stored in the `L_OPTIMIZE/'OLD-VALUE'` directory.
`COST-EXTRAP`	keyword used to calculate the extrapolated objective constant *ecost*.

ecost	extrapolated objective constant.
`CONV-TEST`	keyword used to calculate if the external convergence has been reached.
l_{conv}	$= 1$ means that external convergence has been reached; $= 0$ otherwise.

7 PIN-POWER RECONSTRUCTION MODULES

This section is related to pin-power reconstruction capabilities available in Donjon. The corresponding theory is explained in [?, ?]

7.1 The `NAP:` module

The `NAP:` module supplies the main transport-diffusion equivalence options to DRAGON and DONJON. It can be used to perform the pin power reconstruction.[?, ?] The calling specifications are:

Table 78: Structure **(NAP:)**

```
COMPO := NAP: COMPO TRKNAM FLUNAM :: (descnap1)
MAP := NAP: MAP TRKNAM FLUNAM MATEX MACRES :: (descnap2)
GEONEW := NAP: GEOOLD COMPO :: (descnap3)
```

where

COMPO	`character*12` name of the MULTICOMPO data structure (`L_COMPO` signature) where the detailed subregion properties will be stored.
TRKNAM	`character*12` name of the read-only TRACKING data structure (`L_TRACK` signature) containing the tracking.
FLUNAM	`character*12` name of the read-only FLUXUNK data structure (`L_FLUX` signature) containing a transport solution.
MAP	`character*12` name of the MAP data structure (`L_MAP` signature) containing fuel regions description, global and local parameter information (burnup, fuel/coolant temperatures, coolant density, etc). Keyword `PPR` is expected in **(descnap2)**.
MATEX	`character*12` name of the read-only MATEX data structure (`L_MATEX` signature). The object corresponds to the heterogeneously splited geometry. Keyword `PPR` is expected in **(descnap2)**.
MACRES	`character*12` name of the read-only MACROLIB data structure (`L_MACROLIB` signature) containing a cross section for the fuel. The MACROLIB data structure must have been created with a MULTICOMPO data structure with pin level properties (transport flux, H-factor, infinite domain diffusion flux). Keyword `PPR` is expected in **(descnap2)**.
GEONEW	`character*12` name of the created GEOMETRY data structure (`L_GEOM` signature) containing the detailed core geometry definition at heterogeneous assembly level.
GEOOLD	`character*12` name of the read-only GEOMETRY data structure (`L_GEOM` signature) containing the core geometry definition with homogeneous assembly (only 1 mesh per assembly mandatory).
(descnap1)	structure containing the input data to this module to compute additional properties for subregions (see Section **??**).
(descnap2)	structure containing the input data to this module to perform pin power reconstruction (see Section **??**).

(descnap3)	structure containing the input data to this module to automatically define the core geometry with heterogeneous assembly (see Section **??**).

7.1.1 Additional properties calculations

Table 79: Structure **(descnap1)**

```
[ EDIT iprint ]
PROJECTION
STEP namedir
[ IFX ifx ]
{ [[ SET pname pvalue ]] }
;
```

where

`EDIT`	keyword used to modify the print level *iprint*.
iprint	integer index used to control the printing in module `NAP:`. =0 for no print; =1 for minimum printing (default value); larger values of *iprint* will produce increasing amounts of output.
`PROJECTION`	keyword to specify that additional properties for subregions will be computed and stored in the MULTICOMPO data-structure *COMPO*.
`STEP`	keyword to specify *namedir*.
namedir	name of the directory containing the homogenized cross sections (homogeneous or heterogeneous).
`IFX`	keyword to specify *ifx*.
ifx	number used to create the name of the flux record representing $\psi^{d,\infty}_{m,p}$. This flux represents the results of calculation in an infinite domain computed in diffusion with cross sections homogenized either homogeneously or heterogeneously. One record is associated with each type of homogenization when the MULTICOMPO is created. Thus MULTICOMPO can be "enriched" several times, each time using a different homogenization of the assembly at the end of the transport calculations. The following format is used for the flux record name (`RECNAME`): `WRITE(RECNAME,'5HFINF_,I3.3') IFX` Thus, for example for *ifx*=2, `RECNAME` would be `FINF_002`.
`SET`	keyword to specify assembly calculations at which $\psi^{d,\infty}_{m,p}$ have been calculated. Repeated as many times as there are parameters in the MULTICOMPO.
pname	name of the parameter in the MULTICOMPO.
pvalue	value of the parameter in the MULTICOMPO.

Note that in the case of heterogeneously homogenized assembly, the pin-wise projected diffusion flux is stored in mixture 1.

7.1.2 Pin power reconstruction

Table 80: Structure **(descnap2)**

```
[ EDIT iprint ]
PPR
NZASS nzass
METH GPPR ifx
[ POWER pow ]
;
```

where

`EDIT`	keyword used to modify the print level *iprint*.
iprint	index used to control the printing of this module. The *iprint* parameter is important for adjusting the amount of data that is printed by this calculation step: • *iprint*=0 results in no output; • *iprint*=1 ...
`PPR`	keyword to perform pin power reconstruction and stored results in the MAP data-structure *MAP*.
`NZASS`	keyword to specify *nzass*.
nzass	number of mesh in Z direction along assemblies.
`METH`	keyword to select the type of methodology used for pin power reconstruction.
`GPPR`	keyword to select the generalized pin power reconstruction from an heterogeneous assembly definition (several mixtures per assembly). Note that if there is only one mixture (homogeneous assembly) this method is actually the regular pin power reconstruction.
ifx	number used to create the named of the flux record representing $\psi_{m,p}^{d,\infty}$. See Sect. **??** for more details.
`POWER`	keyword used to normalize the flux. If this keyword is not used, the flux directly computed by the `FLUD:` is used to perform the pin-power reconstruction, then normalization has to be performed by the user indepently.
pow	power used to normalize the flux (MW).

7.1.3 Heterogeneous assembly geometry definition

Table 81: Structure **(descnap3)**

```
[ EDIT iprint ]
DIRGEO namedir [ MACGEO ]
MIXASS nmix (imix(i), i=1,nmix)
[ SLPITX-ASS (ispx(i), i=1,nxass) ]
[ SLPITY-ASS (ispy(i), i=1,nyass) ]
[ MAX-MIX-GEO nmxgeo ]
;
```

where

`EDIT`	keyword used to modify the print level *iprint*.
iprint	index used to control the printing of this module. The *iprint* parameter is important for adjusting the amount of data that is printed by this calculation step: • *iprint*=0 results in no output; • *iprint*=1 ...
`DIRGEO`	keyword to specify *namedir*.
namedir	name of the directory containing the heterogeneously or homogeneously homogenized cross sections.
`MACGEO`	keyword to specify that the macro-geometry stored in the *GFF* record of the macrolib will be used instead of the macro-geometry of the heterogeneously or homogeneously homogenized cross sections. Usually this other macro-geometry of the assembly corresponds to the pin-by-pin geometry. Note that another multicompo with all pin-wise properties is needed to be able to use the automatically generated geometry.
`MIXASS`	keyword to specify mixtures corresponding to assembly in the coarse geometry.
nmix	number of type of assembly.
imix	mixture number of the assemblies.
`SLPITX-ASS`	keyword to specify the mesh splitting at assembly level.
ispx	split along x-direction for each mesh of the heterogeneous assembly.
nxass	number of mesh of the heterogeneous assembly along x-direction.
`SLPITY-ASS`	keyword to specify the mesh splitting at assembly level.
ispy	split along y-direction for each mesh of the heterogeneous assembly.
nyass	number of mesh of the heterogeneous assembly along y-direction.

`MAX-MIX-GEO`	keyword to specify the number of mixtures in the original core geometry (i.e. before the core geometry is splited by the `NAP:` module). This keyword is mandatory if there is a reflector in the geometry otherwise the numbers for fuel mixtures will not match between the split core geometry (GEOMETRY) and the split fuel geometry (GEOMETRY embedded in MAP).
nmxgeo	number of mixtures in the original core geometry.

Note: The included geometry in the *COMPO* has to be unfolded, even if the transport calculations are done on a 1/8th assembly. Moreover no split can be defined in the geometry, one mesh ONLY per heterogeneous mixture is mandatory.

8 DONJON DATA STRUCTURES

A brief description of each DRAGON, DONJON and TRIVAC data structures, which can be used with DONJON code, is given in Section **??**. In this section, a detailed description of the DONJON data structures is presented.

8.1 Contents of /fmap/ data structure

A /fmap/ data structure is used to store fuel assembly (or bundle) map and fuel information such as powers, average fluxes, control zones, burnup or refueling scheme. The fuel bundle location are given in an embedded sub-directory which contains the records as a /geometry/ data structure. This object has a signature `L_MAP`; it is created using the `RESINI:` module.

8.1.1 The state-vector content

The dimensioning parameters $\mathcal{S}_i$, which are stored in the state vector for this data structure, represent:

- The number of fuel bundles per channel $N_b = \mathcal{S}_1$
- The number of fuel channels $N_{\rm ch} = \mathcal{S}_2$
- The number of combustion zones $N_{\rm comb} = \mathcal{S}_3$
- The number of energy groups $N_{\rm gr} = \mathcal{S}_4$
- The type of interpolation with respect to burnup $I_{\rm btyp} = \mathcal{S}_5$

$$I_{\rm btyp} = \begin{cases} 0 & \text{interpolation type is not provided} \\ 1 & \text{according to the time-average model} \\ 2 & \text{according to the instantaneous model} \end{cases}$$

- The number of bundle shift. $N_{\rm sht} = \mathcal{S}_6$
- The number of fuel types $N_{\rm fuel} = \mathcal{S}_7$
- The number of recorded parameters $N_{\rm parm} = \mathcal{S}_8$
- The total number of fuel bundles $N_{\rm tot} = \mathcal{S}_9$
- The number of voided reactor channels $N_{\rm void} = \mathcal{S}_{10}$
- The option with respect to the core-voiding pattern $I_{\rm void} = \mathcal{S}_{11}$

$$I_{\rm void} = \begin{cases} 0 & \text{voiding pattern not provided} \\ 1 & \text{full-core voiding pattern} \\ 2 & \text{half-core voiding pattern} \\ 3 & \text{quarter-core voiding pattern} \\ 4 & \text{checkerboard-full voiding pattern} \\ 5 & \text{checkerboard-half voiding pattern} \\ 6 & \text{checkerboard-quarter voiding pattern} \\ 7 & \text{user-defined voiding pattern} \end{cases}$$

- The type of the geometry $F_t = \mathcal{S}_{12}$

$$F_t = \begin{cases} 7 & \text{Cartesian } \textit{3-D} \text{ geometry} \\ 9 & \text{Hexagonal } \textit{3-D} \text{ geometry} \end{cases}$$

- The naval-coordinate layout used by the `SIM:` module $I_{\rm sim} = \mathcal{S}_{13}$.

  The number of assemblies along X and Y axis are given using

$$L_{\rm x} = \frac{I_{\rm sim}}{100} \quad \text{and} \quad L_{\rm y} = {\rm mod}(I_{\rm sim}, 100)$$

- The total number of assemblies $N_{ass} = \mathcal{S}_{14}$
- The number of assemblies along X direction in Cartesian geometry. The number of assemblies in the radial plane in hexagonal geometry. $N_{xa} = \mathcal{S}_{15}$
- The number of assemblies along Y direction in Cartesian geometry $N_{ya} = \mathcal{S}_{16}$
- The number of plane of the mesh along Z direction where assemblies are situated $N_{z,ass} = \mathcal{S}_{17}$

8.1.2 The main /fmap/ directory

The following records and sub-directories will be found on the first level of /fmap/ directory:

Table 82: Records and sub-directories in /fmap/ data structure

Name	Type	Condition	Units	Comment
`SIGNATURE␣␣␣`	C∗12			Signature of the /fmap/ data structure (SIGNA =`L_MAP␣␣␣␣␣␣␣`).
`STATE-VECTOR`	I(40)			Vector describing the various parameters associated with this data structure $\mathcal{S}_i$
`FLMIX␣␣␣␣␣␣␣`	I($N_{\rm ch}, N_b$)			Fuel mixture indices per bundle or assembly subdivisions for each reactor channel.
`FLMIX-INI␣␣␣`	I($N_{\rm ch}, N_b$)	$I_{\rm sim} \neq 0$		Fuel mixture indices per bundle or assembly subdivisions for each reactor channel, as defined by user in `RESINI:` module.
`S-ZONE␣␣␣␣␣␣`	C($N_{\rm ch}$) ∗ 4	$I_{\rm sim} \neq 0$		identification name corresponding to the basic naval-coordinate position of an assembly, as defined by user in `RESINI:` module..
`BMIX␣␣␣␣␣␣␣␣`	I(N_x, N_y, N_z)			Renumbered mixture indices per each fuel region over the fuel-map geometry; for the non-fuel regions these indices are set to 0.
`XNAME␣␣␣␣␣␣␣`	C(N_x) ∗ 4	$F_t = 7$		Channel identification names with respect to their horizontal position in Cartesian geometry.
`YNAME␣␣␣␣␣␣␣`	C(N_y) ∗ 4	$F_t = 7$		Channel identification names with respect to their vertical position in Cartesian geometry.

continued on next page

Records and sub-directories in /fmap/ data structure continued from last page

Name	Type	Condition	Units	Comment
`HNAME␣␣␣␣␣␣␣`	C(N_x) ∗ 8	$F_t = 9$		Channel identification names with respect to their radial position in hexagonal geometry.
`AXNAME␣␣␣␣␣␣`	C(N_{xa}) ∗ 4	$F_t = 7$		Name of the assembly on X-direction (4 character name per assembly) in Cartesian geometry
`AYNAME␣␣␣␣␣␣`	C(N_{ya}) ∗ 4	$F_t = 7$		Name of the assembly on Y-direction (4 character name per assembly) in Cartesian geometry
`ASSEMBLY␣␣␣␣`	Dir(N_{ass})			List of *assembly* directories. Each component of this list follows the specification presented in Section **??**.
`B-ZONE␣␣␣␣␣␣`	I($N_{\rm ch}$)	$N_{\rm comb} \geq 1$		Combustion-zone indices per channel.
`BURN-AVG␣␣␣␣`	R($N_{\rm comb}$)		MW d t^{-1}	Average exit burnups per combustion zone.
`BURN-INST␣␣␣`	R($N_{\rm ch}, N_b$)	$I_{\rm btyp} = 2$	MW d t^{-1}	Instantaneous burnups per bundle or assembly subdivisions for each channel.
`BURN-BEG␣␣␣␣`	R($N_{\rm ch}, N_b$)	$I_{\rm btyp} = 1$	MW d t^{-1}	Low burnup integration limits according to the time-average model.
`BURN-END␣␣␣␣`	R($N_{\rm ch}, N_b$)	$I_{\rm btyp} = 1$	MW d t^{-1}	Upper burnup integration limits according to the time-average model.
`BUND-PW␣␣␣␣␣`	R($N_{\rm ch}, N_b$)	*	kW	Bundle-powers set in `RESINI:` module or recovered from `L_POWER` object.
`BUND-PW-INI␣`	R($N_{\rm ch}, N_b$)	*	kW	Beginning-of-transient bundle-powers recovered from `L_POWER` object.
`FLUX-AV␣␣␣␣␣`	R($N_{\rm ch}, N_b, N_{\rm gr}$)	*	cm^{-2} s^{-1}	The normalized average fluxes recorded per each fuel bundle and for each energy group, recovered from `L_POWER` object.
`B-EXIT␣␣␣␣␣␣`	R(1)		MW d t^{-1}	Core-average discharge burnup.
`REF-SHIFT␣␣␣`	I($N_{\rm comb}$)			Bundle-shifts per combustion zone. A bundle-shift corresponds to the number of displaced fuel bundles during the refueling operation.
`REF-VECTOR␣␣`	I($N_{\rm comb}, N_b$)			Refueling pattern vector per combustion zone.
`REF-SCHEME␣␣`	I($N_{\rm ch}$)			Refueling scheme of each channel; it corresponds to the positive or negative bundle-shift number according to the flow direction.
`REF-RATE␣␣␣␣`	R($N_{\rm ch}$)		kg d^{-1}	Channel refueling rates.
`REF-CHAN␣␣␣␣`	R($N_{\rm ch}$)		d	Time values at which channels are refueled inside a refueling time period.
`DEPL-TIME␣␣␣`	R(1)		d	Refueling time period in days.
{*pshift*}	R($N_{\rm ch}, N_b$)	$N_{\rm sht} \geq 1$	*kW*	The power of the bundles shifted the i-th time.
{*bshift*}	R($N_{\rm ch}, N_b$)	$N_{\rm sht} \geq 1$	$MWdT^{-1}$	The burnup of the bundles shifted the i-th time.
{*ishift*}	I($N_{\rm ch}, N_b$)	$N_{\rm sht} \geq 1$		The number of shifts per bundle during refueling.
`AX-SHAPE␣␣␣␣`	R($N_{\rm ch}, N_b$)	$I_{\rm btyp} = 1$		Normalized axial power-shape values over the fuel bundles. Equal to fuel-bundle powers divided by channel powers.

continued on next page

Records and sub-directories in /fmap/ data structure continued from last page

Name	Type	Condition	Units	Comment
`EPS-AX␣␣␣␣␣␣`	R(1)	$I_{\text{btyp}} = 1$		Convergence factor for the axial power-shape calculation; it is defined as a relative error between the two successives calculations.
`GEOMAP␣␣␣␣␣␣`	Dir			Sub-directory containing the embedded *3D*-Cartesian /geometry/ of the fuel lattice.
`FUEL␣␣␣␣␣␣␣␣`	Dir(N_{fuel})			List of fuel–type sub-directories. Each component of the list is a directory containing the information relative to a single fuel type.
`{hcycle}`	Dir(N_{burn})	$I_{\text{sim}} \neq 0$		Sub-directory containing information related to a fuel cycle in a PWR. N_{burn} is the number of burnup steps used during the simulation of the cycle. These burnup steps may not be of increasing values.
`PARAM␣␣␣␣␣␣␣`	Dir(N_{parm})	$N_{\text{parm}} > 0$		List of parameter–type sub-directories. Each component of the list is a directory containing the information relative to a single parameter. The total number of sub-directories corresponds to the total number of recorded parameters N_{parm} (excluding burnups).

The contents of the `GEOMAP` sub-directory correspond to the typical contents of the /geometry/ data structure. The dimensioning parameters N_x, N_y, and N_z represent the number of volumes along the corresponding axis in the fuel-map geometry.

The shifting information records {pshift}, {bshift} and {ishift} will be composed using the following FORTRAN instructions, respectively, as

```
WRITE(pshift,'(A6,I2)') 'PSHIFT',ell

WRITE(bshift,'(A6,I2)') 'BSHIFT',ell

WRITE(ishift,'(A6,I2)') 'ISHIFT',ell
```

for $1 \leq ell \leq N_{\text{sht}}$.

Each time a bundle is shifted and stay in the reactor, its burnup and power will be saved in the records {bshift} and {pshift}. For example, {bshift i} and {pshift i} will contain all the burnups and powers of bundles that have been shifted *i*-th time.

8.1.3 The `FUEL` sub-directories

Each `FUEL` sub-directory contains the information corresponding to a single fuel type. Inside each sub-directory, the following records will be found:

Table 83: Records in `FUEL` sub-directories

Name	Type	Condition	Units	Comment
`MIX␣␣␣␣␣␣␣␣␣`	I(1)			Fuel-type mixture number.
`TOT␣␣␣␣␣␣␣␣␣`	I(1)			Total number of fuel bundles for this fuel type.
`MIX-VOID␣␣␣␣`	I(1)			Voided-cell mixture number for this fuel type.
`WEIGHT␣␣␣␣␣␣`	R(1)		kg	Fuel weight in a bundle for this fuel type.
`ENRICH␣␣␣␣␣␣`	R(1)		wt%	Fuel enrichment for this fuel type.
`POISON␣␣␣␣␣␣`	R(1)			Poison load for this fuel type.

8.1.4 The {hcycle} sub-directories

Each {hcycle} sub-directory contains the information corresponding to a single PWR fuel cycle. Inside each sub-directory, the following records will be found:

Table 84: Records in {hcycle} sub-directories

Name	Type	Condition	Units	Comment
`TIME␣␣␣␣␣␣␣␣`	R(1)		d	Depletion time corresponding to instantaneous burnup values.
`BURNAVG␣␣␣␣␣`	R(1)		MW d t^{-1}	Average burnup of the assembly.
`NAME␣␣␣␣␣␣␣␣`	$C(N_{\rm ch}) * 12$			Names of each assembly or of each quart-of assembly during a refuelling cycle. All quart-of-assembly belonging to the same assembly have the same name.
`FLMIX␣␣␣␣␣␣␣`	$I(N_{\rm ch}, N_b)$			Fuel mixture indices per assembly subdivisions for each reactor channel.
`BURN-INST␣␣␣`	$R(N_{\rm ch}, N_b)$		MW d t^{-1}	Instantaneous burnups per assembly subdivisions for each channel.
`POWER-BUND␣␣`	$R(N_{\rm ch}, N_b)$		kW	Powers per assembly subdivisions for each channel.

8.1.5 The `PARAM` sub-directories

Each `PARAM` sub-directory contains the information corresponding to a single local or global parameter (excluding burnups). Inside a such sub-directory, the following records will be found:

Table 85: Records in `PARAM` sub-directories

Name	Type	Condition	Units	Comment
`P-NAME␣␣␣␣␣␣`	C∗12			Unique identification name of this parameter. This name is user-defined; however, it is recommended to use the following pre-defined values: `C-BORE` Boron concentration; `T-FUEL` Averaged fuel temperature; `T-SURF` Surfacic fuel temperature; `T-COOL` Averaged coolant temperature; `D-COOL` Averaged coolant density; CANDU-only parameters: `T-MODE` Averaged moderator temperature; `D-MODE` Averaged moderator density
`PARKEY␣␣␣␣␣␣`	C∗12			Corresponding name of this parameter as recorded in a multi-parameter Compo file.
`P-TYPE␣␣␣␣␣␣`	I(1)			Number associated to the type of recorded parameter: *ptype* = 1 for global parameter; *ptype* = 2 for local parameter.
`P-VALUE␣␣␣␣␣`	R(1)	*ptype* = 1		Recorded single value for global parameter.
	$R(N_{\mathrm{ch}}, N_b)$	*ptype* = 2		Recorded values for local parameter per each fuel bundle for every channel.

8.1.6 The ASSEMBLY sub-directory

Table 86: Assembly type sub-directory

Name	Type	Condition	Units	Comment
`LABEL␣␣␣␣␣␣␣`	C∗8			Label of the assembly. It consists a 8 character name composed of the AYNAME and AXNAME
`PIN-POWER␣␣␣`	$R(N_{pin}^2 * N_{z,ass})$			array containing the pin power for each pin on each mesh plane of the assembly.
`ASS-POWER␣␣␣`	R(1)			total assembly power.
`SIG_F*PHI␣␣␣`	$R(N_{pin}^2 * N_{z,ass})$			array containing the integral of Σ_f times the flux for each pin on each mesh plane of the assembly.

continued on next page

Assembly type sub-directory continued from last page

Name	Type	Condition	Units	Comment
`FLUX␣␣␣␣␣␣␣␣`	R($N_{pin}^2 * N_{z,ass} * N_g$)			array containing the flux of each group for each pin on each mesh plane of the assembly.

8.2 Contents of /matex/ data structure

A /matex/ data structure is used to store several information related to the reactor extended material index and geometry. This object has a signature `L_MATEX`; it is created using the `USPLIT:` module. The information contained in this data structure can be used and updated in other DONJON modules.

8.2.1 The state-vector content

The dimensioning parameters $\mathcal{S}_i$, which are stored in the state vector for this data structure, represent:

- The number of energy groups $N_{gr} = \mathcal{S}_1$
- The maximum number of material mixtures $N_m = \mathcal{S}_2$ (N_m equals to the total number of material regions plus the number of device mixtures)
- The number of reflector types $N_r = \mathcal{S}_3$
- The number of fuel types $N_f = \mathcal{S}_4$
- The total number of mixtures indices $N_{tot} = \mathcal{S}_5$ (N_{tot} equals to the total number of mesh-splitted volumes plus the number of device mixtures)
- The type of reactor geometry $I_g = \mathcal{S}_6$ (only $I_g = 7$ for *3D*-Cartesian geometry or $I_g = 9$ for *3D*-Hexagonal geometry are allowed)
- The total number of mesh-splitted volumes $N_{el} = \mathcal{S}_7$
- The number of mesh-splitted volumes along x-axis $L_x = \mathcal{S}_8$
- The number of mesh-splitted volumes along y-axis $L_y = \mathcal{S}_9$
- The number of mesh-splitted volumes along z-axis $L_z = \mathcal{S}_{10}$

8.2.2 The /matex/ directory

The following records will be found on the /matex/ directory:

Table 87: Records in /matex/ data structure

Name	Type	Condition	Units	Comment
`SIGNATURE␣␣␣`	C∗12			Signature of the /matex/ data structure (SIGNA =`L_MATEX␣␣␣␣␣`).
`STATE-VECTOR`	I(40)			Vector describing the various parameters associated with this data structure $\mathcal{S}_i$
`RMIX␣␣␣␣␣␣␣␣`	I(N_r)			The reflector-type mixture indices, as defined in the reactor geometry.
`RTOT␣␣␣␣␣␣␣␣`	I(N_r)			The total number of reflector regions per each reflector type.
`FMIX␣␣␣␣␣␣␣␣`	I(N_f)			The fuel-type mixture indices, as defined in the reactor geometry.
`FTOT␣␣␣␣␣␣␣␣`	I(N_f)			The total number of fuel regions per each fuel type.
`MAT␣␣␣␣␣␣␣␣␣`	I(N_{tot})			The material mixture indices per each region and including the device mixtures. The fuel-type indices are set negative; the device indices are appended at the end of vector; the virtual-region indices are set to 0.
`INDEX␣␣␣␣␣␣␣`	I(N_{el})			The renumbered mixture indices. A unique number is associated with each mesh-splitted volume. The device indices are not included; the virtual-region indices are set to 0.
`MESHX␣␣␣␣␣␣␣`	R(L_x+1)			The mesh-splitted coordinates along x-axis of the reactor geometry.
`MESHY␣␣␣␣␣␣␣`	R(L_y+1)			The mesh-splitted coordinates along y-axis of the reactor geometry.
`MESHZ␣␣␣␣␣␣␣`	R(L_z+1)			The mesh-splitted coordinates along z-axis of the reactor geometry.
`H-FACTOR␣␣␣␣`	R(N_m, N_{gr})			The h-factors per each mixture and per each energy group, as recovered from the extended /macrolib/ data structure.

8.3 Contents of /device/ data structure

A /device/ data structure is used to store several information related to the reactor devices. This object has a signature `L_DEVICE`; it is created using the `DEVINI:` module. The information contained in this data structure can be used and updated in other DONJON modules which are related to the devices, namely: `LZC:`, `DSET:`, `MOVDEV:` and `NEWMAC:` modules.

8.3.1 The state-vector content

The dimensioning parameters $\mathcal{S}_i$, which are stored in the state vector for this data structure, represent:

- The type of reactor geometry $I_g = \mathcal{S}_1$ (only $I_g = 7$ for *3D*-Cartesian geometry allowed)
- The total number of rod-type devices $N_{rod} = \mathcal{S}_2$
- The total number of the rod-type groups $N_{rgrp} = \mathcal{S}_3$
- The total number of lzc-type devices $N_{lzc} = \mathcal{S}_4$
- The total number of the lzc-type groups $N_{lgrp} = \mathcal{S}_5$
- Type of control rod movement $I_{mov} = \mathcal{S}_6$ where

$$I_{\text{mov}} = \begin{cases} 1 & \text{Fading. A fraction of the fully inserted rod vanishes} \\ 2 & \text{Moving. The complete rod is moving (DONJON3-type movement).} \end{cases}$$

8.3.2 The main /device/ directory

The following records and sub-directories will be found on the first level of /device/ directory:

Table 88: Records and sub-directories in /device/ data structure

Name	Type	Condition	Units	Comment
`SIGNATURE␣␣␣`	C∗12			Signature of the /device/ data structure (SIGNA =`L_DEVICE␣␣␣␣`).
`STATE-VECTOR`	I(40)			Vector describing the various parameters associated with this data structure $\mathcal{S}_i$
`DEV_ROD␣␣␣␣␣`	Dir(N_{rod})			Sub-directories for each controller rod. A sub-directory is created for each controller rod according to the rod identification number.
`ROD_GROUP␣␣␣`	Dir(N_{rgrp})			Sub-directories for each group of rod-type devices. A sub-directory is created for each group of rod-type devices according to the rod-group identification number.

continued on next page

Records and sub-directories in /device/ data structure continued from last page

Name	Type	Condition	Units	Comment
`DEV_LZC␣␣␣␣␣`	Dir(N_{lzc})			Sub-directories for each liquid zone controller. A sub-directory is created for each liquid controller according to the liquid controller identification number.
`LZC_GROUP␣␣␣`	Dir(N_{lgrp})			Sub-directories for each group of lzc-type devices. A sub-directory is created for each group of lzc-type devices according to the lzc-group identification number.

8.3.3 The `DEV-ROD` sub-directories

Inside each `DEV-ROD` sub-directory, the following records will be found:

Table 89: Records in `DEV-ROD` sub-directories

Name	Type	Condition	Units	Comment
`ROD-ID␣␣␣␣␣␣`	I(1)			The identification number of the rod.
`ROD-NAME␣␣␣␣`	C∗12			The identification name of the rod.
`ROD-PARTS␣␣␣`	I(1)			The number of parts in the rod ($N_{\rm part} \geq 1$).
`AXIS␣␣␣␣␣␣␣␣`	I(1)			The number used to identify the rod mouvement direction: = 1 along x-axis; = 2 along y-axis; =3 along z-axis.
`FROM␣␣␣␣␣␣␣␣`	I(1)			The number used to identify the side of geometry, from which the controller rod is inserted into the reactor core along its direction of mouvement: = 1 if a rod is inserted from the highest position (e.g. from the top); = -1 if a rod is inserted from the lowest position (e.g. from the bottom).
`LENGTH␣␣␣␣␣␣`	R(2)		cm	The initial and final position coordinates of the full-inserted rod along its direction of movement. The rod length is the distance between these two coordinates.
`CORE-LIMITS␣`	R(6)		cm	The initial and final position coordinates of the full core along each Cartesian direction.
`MAX-POS␣␣␣␣␣`	R($6 \times N_{\rm part}$)		cm	The limiting *3D*-Cartesian coordinates of the full-inserted rod. This data is given for each part of the rod.

continued on next page

Records in DEV-ROD sub-directories continued from last page

Name	Type	Condition	Units	Comment
`ROD-MIX␣␣␣␣␣`	$I(2 \times N_{\text{part}})$			The rod-type mixture indices. The first number corresponds to the inserted rod position and the second to the withdrawn rod position. This data is given for each part of the rod.
`LEVEL␣␣␣␣␣␣␣`	$R(1)$	*		The actual insertion level of the controller rod. This value must be between 0.0 for the full-withdrawn rod and 1.0 for the full-inserted rod.
`SPEED␣␣␣␣␣␣␣`	$R(1)$	*	cm s^{-1}	The speed of rod movement (insertion or extraction).
`TIME␣␣␣␣␣␣␣␣`	$R(1)$	*	s	The time for the full rod insertion or extraction.
`ROD-POS␣␣␣␣␣`	$R(6 \times N_{\text{part}})$	*	cm	The actual *3D*-Cartesian coordinates of the rod. This data is given for each part of the rod.

8.3.4 The ROD-GROUP sub-directories

Inside each `ROD-GROUP` sub-directory, the following records will be found:

Table 90: Records in `ROD-GROUP` sub-directories

Name	Type	Condition	Units	Comment
`GROUP-ID␣␣␣␣`	$I(1)$			The identification number of the rod-group.
`NUM-ROD␣␣␣␣␣`	$I(1)$			The total number N_{rd} of rod-devices in the group.
`ROD-ID␣␣␣␣␣␣`	$I(N_{rd})$			An array of identification numbers of rods which belong to the same group.

8.3.5 The DEV-LZC sub-directories

Inside each `DEV-LZC` sub-directory, the following records will be found:

Table 91: Records in DEV-LZC sub-directories

Name	Type	Condition	Units	Comment
LZC-ID␣␣␣␣␣␣	I(1)			The identification number of the liquid zone controller.
MAX-POS␣␣␣␣␣	R(6)		cm	The limiting *3D*-Cartesian coordinates of the whole liquid controller, including its empty and full parts.
AXIS␣␣␣␣␣␣␣␣	I(1)			The number used to identify the water filling direction: = 1 along x-axis; = 2 along y-axis; =3 along z-axis.
HEIGHT␣␣␣␣␣␣	R(1)		cm	The water height of the full-filled controller along its direction of filling.
LEVEL␣␣␣␣␣␣␣	R(1)			The actual water level of the liquid controller device. This value must be between 0.0 for the empty state and 1.0 for the full-filled state.
EMPTY-POS␣␣␣	R(6)		cm	The actual *3D*-Cartesian coordinates of the empty-part of liquid contoller.
FULL-POS␣␣␣␣	R(6)		cm	The actual *3D*-Cartesian coordinates of the full-part of liquid contoller.
EMPTY-MIX␣␣␣	I(2)			The empty-part mixture number and the reference mixture number.
FULL-MIX␣␣␣␣	I(2)			The full-part mixture number and the reference mixture number.
RATE␣␣␣␣␣␣␣␣	R(1)		$m^3 s^{-1}$	The water filling rate.
TIME␣␣␣␣␣␣␣␣	R(1)		s	The water filling time.

8.3.6 The LZC-GROUP sub-directories

Inside each LZC-GROUP sub-directory, the following records will be found:

Table 92: Records in LZC-GROUP sub-directories

Name	Type	Condition	Units	Comment
GROUP-ID␣␣␣␣	I(1)			The identification number of the lzc-group.
NUM-LZC␣␣␣␣␣	I(1)			The total number N_{ld} of lzc-devices in the group.
LZC-ID␣␣␣␣␣␣	I(N_{ld})			An array of identification numbers of liquid controllers which belong to the same group.

8.4 Contents of a /detect/ data structure

The /detect/ data structure is used to store detector positions and responses. This object has a signature `L_DETECT`; it is created using the `DETINI:` module. The information contained in this data structure can be used and updated in other DONJON modules which are related to the detectors, namely: `DETECT:` and `DETINI:` modules.

8.4.1 The state-vector content

The dimensioning parameters $\mathcal{S}_i$, which are stored in the state vector for this data structure, represent:

- The number of energy groups. $N_g = \mathcal{S}_1$
- The total number of detectors $\sum \mathcal{I}_1 = \mathcal{S}_2$.
- Flag for hexagonal detector definition $\mathcal{S}_3 = 1$ for hexagonal detector definition, $= 0$ otherwise.

The dimensioning parameters for a specific detector type, which are stored in the vector $\mathcal{I}_i$, represents:

- The number of detectors of type { name_type } $\mathcal{I}_1$.
- The number of delayed responses + 2, $\mathcal{I}_2$.

8.4.2 The main /detect/ directory

The following records and sub-directories will be found on the first level of /detect/ directory:

Table 93: Records and sub-directories in /detect/ data structure

Name	Type	Condition	Units	Comment
`SIGNATURE␣␣␣`	C∗12			Signature of the /detect/ data structure (SIGNA =`L_DETECT␣␣␣␣`).
`STATE-VECTOR`	I(40)			Vector describing the various parameters associated with this data structure $\mathcal{S}_i$
{*/name_type/*}	Dir			Detector type sub-directory contains informations for each detector of this type.

8.4.3 The /name_type/ sub-directories

Inside each /name_type/ sub-directory, the following records will be found:

Table 94: Records in /name_type/ sub-directories

Name	Type	Condition	Units	Comment
`INFORMATIONS`	I(2)			Record containing describing the various parameters associated with a detector type $\mathcal{I}_i$.
`INV-CONST␣␣␣`	R($\mathcal{I}_2 - 2$)		s^{-1}	The inverse time constant of the delayed detector responses.
`FRACTION␣␣␣␣`	R($\mathcal{I}_2 - 1$)			The delayed and prompt fractions of the detector responses.
`SPECTRAL␣␣␣␣`	R($\mathcal{I}_2 - 1$)			The energy spectrum of the detector.
{*/name_detect/*}	Dir			Detector information sub-directory

8.4.4 The /name_detect/ sub-directories

Inside each /name_detect/ sub-directory, the following records will be found:

Table 95: Records in /name_detect/ sub-directories

Name	Type	Condition	Units	Comment
`NHEX␣␣␣␣␣␣␣␣`	I($nhex$)	$hex = 1$		The numbers of affected hexagons in the first X-Y plane.
`POSITION␣␣␣␣`	R(6)		cm	The coordinates of the detector.
`RESPON␣␣␣␣␣␣`	R($\mathcal{I}_2$)			The responses of the detector.

8.5 Contents of /power/ data structure

A /power/ data structure is used to store the information related to the powers and fluxes over the reactor core. This object has a signature `L_POWER`; it is created using the `FLPOW:` module. The reactor fluxes and powers are recorded using several data formats.

8.5.1 The state-vector content

The dimensioning parameters $\mathcal{S}_i$, which are stored in the state vector for this data structure, represent:

- The number of energy groups $N_{gr} = \mathcal{S}_1$
- The total number of mesh-splitted volumes $N_{el} = \mathcal{S}_2$
- The number of mesh-splitted volumes along x-axis $L_x = \mathcal{S}_3$
- The number of mesh-splitted volumes along y-axis $L_y = \mathcal{S}_4$
- The number of mesh-splitted volumes along z-axis $L_z = \mathcal{S}_5$
- The number of reactor channels $N_{ch} = \mathcal{S}_6$
- The number of bundles per channel $N_b = \mathcal{S}_7$

8.5.2 The /power/ directory

The following records will be found on the /power/ directory:

Table 96: Records in /power/ data structure

Name	Type	Condition	Units	Comment
`SIGNATURE␣␣␣`	C∗12			Signature of the /power/ data structure (SIGNA =`L_POWER␣␣␣␣␣`).
`STATE-VECTOR`	I(40)			Vector describing the various parameters associated with this data structure $\mathcal{S}_i$
`PTOT␣␣␣␣␣␣␣␣`	D(1)		MW	The total reactor power.
`VTOT␣␣␣␣␣␣␣␣`	D(1)		cm^3	The total reactor volume.
`NORM␣␣␣␣␣␣␣␣`	D(1)			The flux normalization factor.
`FLMIX␣␣␣␣␣␣␣`	I(N_{ch}, N_b)			Fuel mixture indices per fuel bundle.
`FLUX␣␣␣␣␣␣␣␣`	R(N_{el}, N_{gr})		cm^{-2} s^{-1}	The normalized fluxes over the whole reactor geometry, recorded per each mesh-splitted volume and per each energy group. The flux values over the virtual regions are set to 0.
`VOLU-BUND␣␣␣`	R(N_{ch}, N_b)		cm^2	The volume of each fuel bundle.
`FLUX-BUND␣␣␣`	R(N_{ch}, N_b, N_{gr})		cm^{-2} s^{-1}	The normalized average fluxes recorded per each fuel bundle and per each energy group.

continued on next page

Records in /power/ data structure continued from last page

Name	Type	Condition	Units	Comment
`FLUX-DISTR␣␣`	$\mathrm{R}(L_x, L_y, L_z, N_{gr})$		$\mathrm{cm}^{-2}\ \mathrm{s}^{-1}$	The normalized flux distribution over the whole reactor geometry, recorded per each X-Y-Z planes and per each energy group.
`FLUX-RATIO␣␣`	$\mathrm{R}(L_x, L_y, L_z, N_{gr} - 1)$			The fluxes ratios with respect to the thermal energy-group fluxes.
`POWER-BUND␣␣`	$\mathrm{R}(N_{ch}, N_b)$		*kW*	The bundle powers.
`POWER-CHAN␣␣`	$\mathrm{R}(N_{ch})$		*kW*	The channel powers.
`POWER-DISTR␣`	$\mathrm{R}(L_x, L_y, L_z)$		*W*	The power distribution over the reactor core, recorded per each X-Y-Z planes. The power values over the non-fuel regions are set to 0.
`PMAX-CHAN␣␣␣`	R(1)		*kW*	The maximum channel power.
`PMAX-BUND␣␣␣`	R(1)		*kW*	The maximum bundle power.
`FORM-CHAN␣␣␣`	R(1)			The radial power-form factor, defined as maximum-to-average channel power in core.
`FORM-BUND␣␣␣`	R(1)			The overall power-form factor, defined as maximum-to-average bundle power in core.
`K-EFFECTIVE␣`	R(1)			The effective multiplication factor, recovered from the /flux/ data structure.

All stored fluxes are normalized either to the given total reactor power or using the previously recorded normalization factor. The recorded values of the maximum channel and bundle powers, the channel and bundle power-form factors, and the effective multiplication factor, can be used as power and criticity constraints for the optimization and fuel management purposes.

8.6 Contents of /history/ data structure

This data structure contains the information required to ensure a smooth coupling of DRAGON with DONJON when a history based full reactor calculation is to be performed.

8.6.1 The main directory

The following records and sub-directories will be found in the first level of a /history/ directory:

Table 97: Main records and sub-directories in /history/

Name	Type	Condition	Units	Comment
`SIGNATURE␣␣␣`	C∗12			parameter SIGNA containing the signature of the data structure
`STATE-VECTOR`	I(40)			array $\mathcal{S}_i^h$ containing various parameters that are required to describe this data structure
`BUNDLELENGTH`	R(1)		cm	parameter L_z containing the fuel bundle length
`NAMEGLOBAL␣␣`	C($\mathcal{S}_1^h$)∗12			array $\mathcal{G}_j$ containing the names of the global parameters
`PARAMGLOBAL␣`	R($\mathcal{S}_1^h$)			array G_j containing the value of the global parameters
`NAMELOCAL␣␣␣`	C($\mathcal{S}_2^h$)∗12			array $\mathcal{L}_j$ containing the names of the local parameters
`CELLID␣␣␣␣␣␣`	I($\mathcal{S}_3^h, \mathcal{S}_4^h$)			array $C_{i,j}$ containing an identification number associated with bundle i and channel j
`FUELID␣␣␣␣␣␣`	I($\mathcal{S}_3^h, \mathcal{S}_4^h$)			array $F_{i,j}$ containing the fuel type associated with bundle i and channel j
{*/FUELDIR/*}	Dir			list of sub-directories FUEL$_{i,j}$ that contain the properties associated with the fuel type $F_{i,j}$
{*/CELLDIR/*}	Dir			list of sub-directories CELL$_{i,j}$ that contain the properties associated with the cell $C_{i,j}$

The signature for this data structure is SIGNA=`L_HISTORY␣␣␣`. The array $\mathcal{S}_i^h$ contains the following information:

- $\mathcal{S}_1^h = N_g$ contains the number of global parameters.
- $\mathcal{S}_2^h = N_l$ contains the number of local parameters.
- $\mathcal{S}_3^h = N_b$ contains the number of bundles per channel.
- $\mathcal{S}_4^h = N_c$ contains the number of channels in the core.
- $\mathcal{S}_5^h = N_s$ contains the number of bundle shift.
- $\mathcal{S}_6^h = T_s$ contains the type of depletion solution used.
- $\mathcal{S}_7^h = T_b$ contains the type of burnup considered.

- $\mathcal{S}_8^h = N_I$ contains the number of isotopes.
- $\mathcal{S}_9^h = G$ contains the number of transport groups.
- $\mathcal{S}_{10}^h = N_r$ contains the number of regions.
- $\mathcal{S}_{11}^h = N_F$ contains the number of fuel types.

The fuel directory name $\texttt{FUEL}_{i,j}$ associated with fuel type $F_{i,j}$ is composed using the following FORTRAN instruction:

```
WRITE(FUEL,'(A4,I8.8)') 'FUEL', F_{i,j}
```

This directory will contain the initial isotopic content of this fuel type. The cell directory name $\texttt{CELL}_{i,j}$ associated with $C_{i,j}$ is composed using the following FORTRAN instruction:

```
WRITE(CELL,'(A4,I8.8)') 'CELL', C_{i,j}
```

This directory will contain the value of the local parameters associated with cell $C_{i,j}$ as well as the current isotopic content of this cell.

The identification number $C_{i,j}$ associated with channel j and bundle i can be seen as the serial number of the bundle located at a position in space identified by (i, j). It is automatically managed by the `HST:` module.[?] For a fresh core $C_{i,j} = n$ where n represents the cell order definition in the input file. Upon refueling, some bundles in channel k of the core are displaced from region (l, k) to (m, k), new bundles are introduced at location (l, k) and old bundles removed from location (m, k). If one assumes that C^{NEW} and C^{OLD} represents the value of C after and before refueling then we will have:

$$\begin{aligned} C_{m.k}^{\mathrm{NEW}} &= C_{l,k}^{\mathrm{OLD}} \\ C_{l,k}^{\mathrm{NEW}} &= C_{m,k}^{\mathrm{FRESH}} \end{aligned}$$

where $C_{m,k}^{\mathrm{FRESH}}$ represent a fresh fuel cell. The local parameters and burnup power density of the fuel cell previously located at (m, k) are preserved and the fresh fuel isotopic densities is that provided in $F_{m,k}$, the fuel type associated with $C_{m,k}^{\mathrm{FRESH}}$.

8.6.2 The fuel type sub-directory

Each fuel sub-directory $\texttt{FUEL}_{i,j}$ contains the following information

Table 98: Fuel type sub-directory

Name	Type	Condition	Units	Comment
`FUELDEN-INIT`	R(2)			array containing the initial density of heavy element in the fuel ρ_f in g/cm^3 and the initial linear density of heavy element in the fuel m_f in g/cm.
`ISOTOPESUSED`	C(N_I)∗12			array containing the name of isotopes used in this fuel type
`ISOTOPESMIX␣`	I(N_I)			array containing the mixture associated with each isotopes in this fuel type
`ISOTOPESDENS`	R(N_I)		(cm b)$^{-1}$	array ρ_i containing the density of each isotopes

8.6.3 The cell type sub-directory

Each cell isotopic sub-directory $\mathsf{CELL}_{i,j}$ contains the following information

Table 99: Cell sub-directory

Name	Type	Condition	Units	Comment
`FUELDEN-INIT`	R(2)			array containing the initial density of heavy element in the fuel ρ_f in g/cm^3 and the initial linear density of heavy element in the fuel m_f in g/cm.
`PARAMLOCALBR`	R(N_l)			array V_l^B containing the value of the local parameters before refueling
`PARAMLOCALAR`	R(N_l)			array V_l^A containing the value of the local parameters after refueling
`PARAMBURNTBR`	R(2)			array containing the depletion time T^B in days and the burnup power rate P^B in kW/kg before refueling
`PARAMBURNTAR`	R(2)			array containing the depletion time T^A in days and the burnup power rate P^A in kW/kg after refueling
`DEPL-PARAM␣␣`	R(3)			array containing the time step T in days, the burnup B in kWd/kg and the irradiation w in n/kb currently reached by the fuel in this cell
`ISOTOPESDENS`	R(N_I)		(cm b)$^{-1}$	array ρ_i containing the density of each isotopes

8.7 Contents of /thm/ data structure

This data structure contains the thermal-hydraulics information required in a multi-physics calculation

8.7.1 The main /thm/ directory

The following records and sub-directories will be found in the first level of a /thm/ directory:

Table 100: Main records and sub-directories in /thm/

Name	Type	Condition	Units	Comment
`SIGNATURE␣␣␣`	C∗12			parameter SIGNA containing the signature of the data structure
`STATE-VECTOR`	I(40)			array $\mathcal{S}_i^{th}$ containing various integer parameters that are required to describe this data structure
`REAL-PARAM␣␣`	R(40)			array $\mathcal{R}_i^{th}$ containing various floating-point parameters that are required to describe this data structure
`KCONDF␣␣␣␣␣␣`	R($\mathcal{S}_{16}^{th}+3$)	$\mathcal{S}_{12}^{th}\neq 0$		coefficients of the user-defined correlation for the fuel thermal conductivity
`UCONDF␣␣␣␣␣␣`	C12	$\mathcal{S}_{12}^{th}\neq 0$		string variable set to `KELVIN` or to `CELSIUS`
`KCONDC␣␣␣␣␣␣`	R($\mathcal{S}_{17}^{th}+3$)	$\mathcal{S}_{13}^{th}\neq 0$		coefficients of the user-defined correlation for the clad thermal conductivity
`UCONDC␣␣␣␣␣␣`	C12	$\mathcal{S}_{13}^{th}\neq 0$		string variable set to `KELVIN` or to `CELSIUS`
`ERROR-T-FUEL`	R(1)		K	absolute error in fuel temperature
`ERROR-D-COOL`	R(1)		g/cc	absolute error in coolant density
`ERROR-T-COOL`	R(1)		K	absolute error in coolant temperature
`MIN-T-FUEL␣␣`	R(1)		K	minimum fuel temperature
`MIN-D-COOL␣␣`	R(1)		g/cc	minimum coolant density
`MIN-T-COOL␣␣`	R(1)		K	minimum coolant temperature
`MAX-T-FUEL␣␣`	R(1)		K	maximum fuel temperature
`MAX-D-COOL␣␣`	R(1)		g/cc	maximum coolant density
`MAX-T-COOL␣␣`	R(1)		K	maximum coolant temperature
`HISTORY-DATA`	Dir			sub-directory containing the historical values taken by the thermal-hydraulics parameters (mass flux, density, pressure, enthalpy, temperature) in the coolant and in the fuel rod for the whole geometry

The signature for this data structure is SIGNA=`L_THM`. The array $\mathcal{S}_i^h$ contains the following information:

- $\mathcal{S}_1^{th}$ contains the number of active fuel rods.
- $\mathcal{S}_2^{th}$ contains the number of guide tubes.
- $\mathcal{S}_3^{th}$ contains the maximum number of iterations for computing the conduction integral.
- $\mathcal{S}_4^{th}$ contains the maximum number of iterations for computing the center pellet temperature.

- $\mathcal{S}_5^{th}$ contains the maximum number of iterations for computing the coolant parameters (velocity, pressure, enthapy, density) in case of a transient calculation.
- $\mathcal{S}_6^{th}$ contains the number of discretisation points in fuel.
- $\mathcal{S}_7^{th}$ contains the number of total discretisation points in the whole fuel rod (fuel+cladding).
- $\mathcal{S}_8^{th}$ contains the integer setting the type of calculation (steady-state or transient) performed by the `THM:` module.
- $\mathcal{S}_9^{th}$ contains the current time index.
- $\mathcal{S}_{10}^{th}$ flag to set the gap correlation:

$$\mathcal{S}_{10}^{th} = \begin{cases} 0 & \text{built-in correlation is used} \\ 1 & \text{set the heat exchange coefficient of the gap as a user-defined constant.} \end{cases}$$

- $\mathcal{S}_{11}^{th}$ flag to set the heat transfer coefficient between the clad and fluid:

$$\mathcal{S}_{11}^{th} = \begin{cases} 0 & \text{built-in correlation is used} \\ 1 & \text{set the heat exchange coefficient between the clad and fluid as a user-defined constant.} \end{cases}$$

- $\mathcal{S}_{12}^{th}$ flag to set the fuel thermal conductivity:

$$\mathcal{S}_{12}^{th} = \begin{cases} 0 & \text{built-in correlation is used} \\ 1 & \text{set the fuel thermal conductivity as a function of a simple user-defined correlation.} \end{cases}$$

- $\mathcal{S}_{13}^{th}$ flag to set the clad thermal conductivity:

$$\mathcal{S}_{13}^{th} = \begin{cases} 0 & \text{built-in correlation is used} \\ 1 & \text{set the clad thermal conductivity as a function of a simple user-defined correlation.} \end{cases}$$

- $\mathcal{S}_{14}^{th}$ type of approximation used during the fuel conductivity evaluation:

$$\mathcal{S}_{14}^{th} = \begin{cases} 0 & \text{use a rectangle quadrature approximation} \\ 1 & \text{use an average approximation.} \end{cases}$$

- $\mathcal{S}_{15}^{th}$ type of subcooling model:

$$\mathcal{S}_{15}^{th} = \begin{cases} 0 & \text{use the Jens-Lottes correlation and Bowring's model} \\ 1 & \text{use the Saha-Zuber subcooling model.} \end{cases}$$

- $\mathcal{S}_{16}^{th}$ contains the number of terms in the user-defined correlation for the fuel thermal confuctivity (if $\mathcal{S}_{12}^{th} = 1$).
- $\mathcal{S}_{17}^{th}$ contains the number of terms in the user-defined correlation for the clad thermal confuctivity (if $\mathcal{S}_{13}^{th} = 1$).

The array $\mathcal{R}_i^{th}$ contains the following information:

- $\mathcal{R}_1^{th}$ contains the current time step in s.
- $\mathcal{R}_2^{th}$ contains the fraction of reactor power released in fuel.
- $\mathcal{R}_3^{th}$ contains the critical heat flux in W/m^2.

- $\mathcal{R}^{th}_{4}$ contains the inlet coolant velocity in m/s.
- $\mathcal{R}^{th}_{5}$ contains the outlet coolant pressure in Pa.
- $\mathcal{R}^{th}_{6}$ contains the inlet coolant temperature in K.
- $\mathcal{R}^{th}_{7}$ contains the Plutonium mass fraction in fuel.
- $\mathcal{R}^{th}_{8}$ contains the fuel porosity.
- $\mathcal{R}^{th}_{9}$ contains the fuel pellet radius
- $\mathcal{R}^{th}_{10}$ contains the internal clad rod radius in m.
- $\mathcal{R}^{th}_{11}$ contains the external clad rod radius in m.
- $\mathcal{R}^{th}_{12}$ contains the guide tube radius in m.
- $\mathcal{R}^{th}_{13}$ contains the assembly surface in m^2.
- $\mathcal{R}^{th}_{14}$ contains the temperature maximum absolute error (in K) allowed in the solution of the conduction equations.
- $\mathcal{R}^{th}_{15}$ contains the maximum relative error allowed in the matrix resolution of the conservation equations of the coolant.
- $\mathcal{R}^{th}_{16}$ contains the relaxation parameter for the multiphysics parameters (temperature of fuel and coolant and density of coolant).
- $\mathcal{R}^{th}_{17}$ contains the time in s.
- $\mathcal{R}^{th}_{18}$ contains the heat transfer coefficient of the gap (if $\mathcal{S}^{th}_{10} = 1$).
- $\mathcal{R}^{th}_{19}$ contains the heat transfer coefficient between the clad and fluid (if $\mathcal{S}^{th}_{11} = 1$).
- $\mathcal{R}^{th}_{20}$ contains the surface temperature weighting factor of effective fuel temperature for the Rowlands approximation.

8.7.2 The `HISTORY-DATA` *sub-directory*

In the `HISTORY-DATA` directory, the following sub-directories will be found:

Table 101: Sub-directories in `HISTORY-DATA` directory

Name	Type	Condition	Units	Comment
`STATIC-PARAM`	Dir(N_{ch})			sub-directory containing all the values of the thermal-hydraulics parameters computed by the `THM:` module in steady-state conditions and sorted channel by channel. Each channel is identified by an integer *numc* that can take values between 1 and 9999. For example, the first channel is identified by the string character "`CHANNEL 0001`".
`TIMESTEP`*numt*	Dir(N_{ch})			sub-directories containing all the values of the thermal-hydraulics parameters computed by the `THM:` module in transient conditions at a given time index *numt* and sorted channel by channel. *numt* can take values between 1 and 9999.

In each of the N_{ch} `CHANNEL` *numc* sub-directories, the following records will be found:

Table 102: Records in each `CHANNEL` directory

Name	Type	Condition	Units	Comment
`VINLET␣␣␣␣␣␣`	R(1)		$m.s^{-1}$	inlet velocity
`TINLET␣␣␣␣␣␣`	R(1)		K	inlet temperature
`PINLET␣␣␣␣␣␣`	R(1)		Pa	inlet pressure
`VELOCITIES␣␣`	R(N_b)		$m.s^{-1}$	velocity in each of the N_b bundles of the channel numbered *numc*
`PRESSURE␣␣␣␣`	R(N_b)		Pa	pressure in each bundle of the channel
`ENTHALPY␣␣␣␣`	R(N_b)		$J.kg^{-1}$	enthalpy in each bundle of the channel
`DENSITY␣␣␣␣␣`	R(N_b)		$kg.m^{-3}$	density in each bundle of the channel
`LIQUID-DENS␣`	R(N_b)		$kg.m^{-3}$	density of liquid phase in each bundle of the channel
`TEMPERATURES`	R(N_b, N_{dtot})		K	distribution of the temperature in the fuel-pin for each bundle of the channel
`CENTER-TEMPS`	R(N_b)		K	center fuel pellet temperature in each bundle of the channel

8.8 Contents of a /optimize/ data structure

The /optimize/ specification is used to store the optimization variables and functions values and definitions, limits and options.

In any case, the signature variable for this data structure must be `SIGNA=L_OPTIMIZE␣␣`. The dimensioning parameters for this data structure, which are stored in the state vector $\mathcal{S}^o_i$, represents:

- The number of decision variables $N_{var} = \mathcal{S}^o_1$.
- The number of constraints $N_{cst} = \mathcal{S}^o_2$.
- The type of optimization $\mathcal{S}^o_3$, where

$$\mathcal{S}^o_3 = \begin{cases} 1 & \text{minimization} \\ -1 & \text{maximization} \end{cases}$$

- The result of a test for external convergence of the quadratic constraint $\mathcal{S}^o_4$, where

$$\mathcal{S}^o_4 = \begin{cases} 0 & \text{not converged} \\ 1 & \text{converged} \end{cases}$$

- The number of iterations relative to the quadratic constraint (S^o_5).
- The type of reduction for the radius if the quadratic constraint ($\mathcal{S}^o_6$), where

$$\mathcal{S}^o_6 = \begin{cases} 1 & \text{half} \\ 2 & \text{parabolic} \end{cases}$$

- The number of inner iterations S^o_7.
- The number of outer iterations S^o_8.
- The resolution's method for the linear problem with quadratic constraint ($\mathcal{S}^o_9$), where

$$\mathcal{S}^o_9 = \begin{cases} 1 & \text{SIMPLEX/LEMKE} \\ 2 & \text{LEMKE/LEMKE} \\ 3 & \text{MAP} \\ 4 & \text{Augmented Lagragian} \\ 5 & \text{Penalty Method} \end{cases}$$

- The number of outer iterations without step-back S^o_{10}.
- S^o_{11} (not used).
- S^o_{12} (not used).
- A flag for unsuccessful resolution in module `PLQ:` S^o_{13}, where

$$\mathcal{S}^o_{13} = \begin{cases} 0 & \text{successful at last iteration} \\ \geq 1 & \text{number of iteration with unsuccessful resolution.} \end{cases}$$

Table 103: Main records and sub-directories in /optimize/

Name	Type	Condition	Units	Comment
`SIGNATURE␣␣␣`	C∗12			Signature of the data structure (SIGNA)
`STATE-VECTOR`	I(40)			Vector describing the various parameters associated with data structure $\mathcal{S}_i^o$.
`DLEAK-STATE␣`	I(40)	*		Vector describing the various parameters associated with data structure $\mathcal{S}_i^g$. This array is available if the OPTIMIZE object has been created using module `DLEAK:`.
`VAR-VALUE␣␣␣`	R(N_{var})			The values of the decision variables
`VAR-MAX-VAL␣`	R(N_{var})			The maximum values of the decision variables can be.
`VAR-MIN-VAL␣`	R(N_{var})			The minimum values of the decision variables can be.
`VAR-WEIGHT␣␣`	R(N_{var})			The weight of the decision variables w_i in the quadratic constraint.
`CST-OBJ␣␣␣␣␣`	R(N_{cst})			The limit value of the contraints. The units depends with the type of the constraint type.
`CST-TYPE␣␣␣␣`	I(N_{cst})			The type of the contraints: =-1 for $\geq$; =0 for =; =1 for $\leq$.
`CST-WEIGHT␣␣`	R(N_{cst})			The weight of the constraint η_j and γ_j for the duals and meta-heuristic methods.
`FOBJ-CST-VAL`	R($N_{cst}+1$)			The actual values of the objective function (first value) and the contraints (the following values). The number of the constraints are assigned in the order they have been defined.
`OPT-PARAM-R␣`	R(40)			The different limits and values for the iterative calculations of the optimization problem.
`GRADIENT␣␣␣␣`	R($N_{var}, N_{cst}+1$)			The gradients of the objective function and the constraints. The gradients of the objective for all the decision variables are in first position, then follow the gradients of the constraints.
`OLD-VALUE␣␣␣`	Dir			Directory containing differents informations of the previous iterations. the values of the decision variables, the objective function, the constraints and the gradients of these functions for the previous external iteration. This repertory will be created by the module `QLP:` unless it is specified to not do.

The array `OPT-PARAM-R` contains real values related with the different limits and values for the iterative calculations of the optimization problem:

1st	S	initial radius of the quadratic constraint (default: 1.0).
2nd	δ	initial size of the hypercube for MAP method. (default: 0.1).
3rd	ε_{ext}	limit for external convergence (default: 10^{-4}).
4th	ε_{int}	limit for internal convergence (default: 10^{-4}).
5th	$\varepsilon_{\text{quad}}$	limit for convergence of the quadratic constraint (default: 10^{-4}).

The other value of the record are not used and set to 0.0.

The optional array `DLEAK-STATE` contains integer values related to the definition of mixture and group indices in module `DLEAK:`.

- The number of energy groups in macrolib $\mathcal{S}_1^g$.
- The number of mixtures in macrolib $\mathcal{S}_2^g$.
- The type of leakage parameters $\mathcal{S}_3^g$, where

$$\mathcal{S}_3^g = \begin{cases} 1 & \text{use diffusion coefficients} \\ 2 & \text{use } P_1\text{-weighted macroscopic total cross sections.} \end{cases}$$

- The type of control variables $\mathcal{S}_4^g$, where

$$\mathcal{S}_4^g = \begin{cases} 1 & \text{use leakage parameter itself} \\ 2 & \text{use a correction factor.} \end{cases}$$

- The minimum group index $\mathcal{S}_5^g$, with $1 \le \mathcal{S}_5^g \le \mathcal{S}_1^g$.
- The maximum group index $\mathcal{S}_6^g$, with $\mathcal{S}_5^g \le \mathcal{S}_6^g \le \mathcal{S}_2^g$.
- The minimum mixture index $\mathcal{S}_7^g$, with $1 \le \mathcal{S}_7^g \le \mathcal{S}_2^g$.
- The maximum mixture index $\mathcal{S}_8^g$, with $\mathcal{S}_7^g \le \mathcal{S}_8^g \le \mathcal{S}_2^g$.

8.8.1 The sub-directory /OLD-VALUE/ in /optimize/

Table 104: Main records and sub-directories in //OLD-VALUE//

Name	Type	Condition	Units	Comment
`VARVALUE␣␣␣`	$\text{R}(N_{var})$			The values of the decision variables of the last valid iteration.
`FOBJ-CST-VAL`	$\text{R}(N_{cst}+1)$			The values of the objective function and the contraints of the last valid iteration.
`GRADIENT␣␣␣␣`	$\text{R}(N_{var}, N_{cst}+1)$			The gradients of the objective function and the constraints of the last valid iteration.
`VAR-VALUE2␣␣`	$\text{R}(N_{var})$			The values of the decision variables of the second-last valid iteration.
`BEST-VAR␣␣␣␣`	$\text{R}(N_{var})$			The values of the decision variables corresponding to the best valid solution ever found.
`BEST-FCT␣␣␣␣`	$\text{R}(1)$			The value of the objective function corresponding to the best valid solution ever found.

8.9 Contents of /d2p_info/ data structure

A /d2p_info/ data structure is used to store cross sections and Saphyb information such as keff, kinf, assembly discontinuity factors. This object has a signature `L_INFO`; it is created using the `D2P:` module.

8.9.1 The state-vector content

The dimensioning parameters $\mathcal{S}_i$, which are stored in the state vector for this data structure, represent:

- The number of energy groups $N_{\rm gr} = \mathcal{S}_1$
- The number of pkey for the generation of the PMAXS file $N_{\rm pk} = \mathcal{S}_2$
- The number of cross sections recovered using the `SCR:` module $N_{\rm xs} = \mathcal{S}_3$
- The number of points with respect to burnup $N_{\rm bu} = \mathcal{S}_4$
- The type of branching calculation requested for the generation of PMAXS tree $I_{\rm grid} = \mathcal{S}_5$

$$I_{\rm grid} = \begin{cases} 0 & \text{default meshing} \\ 1 & \text{according to the saphyb content} \\ 2 & \text{user defined with global otpion} \\ 3 & \text{user defined with add otpion} \\ 4 & \text{user defined with add new otpion} \end{cases}$$

- The number of control rod composition $N_{\rm comp} = \mathcal{S}_6$
- The number of delay neutron groups $N_{\rm del} = \mathcal{S}_7$
- The number of columns in assembly $N_{\rm cols} = \mathcal{S}_8$
- The number of rows in assembly $N_{\rm rows} = \mathcal{S}_9$
- The computed part of asembly assembly $I_{\rm part} = \mathcal{S}_{10}$

$$I_{\rm part} = \begin{cases} 0 & \text{whole} \\ 1 & \text{half} \\ 2 & \text{quarter} \\ 3 & \text{eighth} \end{cases}$$

- The number of sides in assembly. $N_{\rm surf} = \mathcal{S}_{11}$
- The number of corners in assembly $N_{\rm corn} = \mathcal{S}_{12}$
- The number of assembly discontinuity factors per energy groups $N_{\rm adf} = \mathcal{S}_{13}$
- The number of ADF-type boundary flux edits. $N_{\rm type} = \mathcal{S}_{14}$
- The number of corner discontinuity factors per energy groups. $N_{\rm cdf} = \mathcal{S}_{15}$
- The number of group-wise form functions per energy groups in compued part of assembly. $N_{\rm gff} = \mathcal{S}_{16}$
- The number of pin on each side of the assembly. $N_{\rm pin} = \mathcal{S}_{17}$

The following records and sub-directories will be found on the first level of /d2p_info/ directory:

Table 105: Records and sub-directories in /d2p_info/ data structure

Name	Type	Condition	Units	Comment
`SIGNATURE␣␣␣`	C∗12			Signature of the /d2p_info/ data structure (SIGNA =`L_INFO␣␣␣␣␣␣`).
`STATE-VECTOR`	I(40)			Vector describing the various parameters associated with this data structure
`BARR_INFO␣␣␣`	I($N_{\rm comp}$)			Meaning of control rod in the saphyb object. N_{comp} is the number of composition for control rods (including the unrodded case)
`SAPHYB_INFO␣`	Dir	$iph \geq 1$		Information related to the saphyb content.
`HELIOS_HEAD␣`	Dir	$iph \geq 1$		Information related to the header of output `.DRA` file
`GENPMAXS_INP`	Dir	$iph \geq 2$		Information related to the GenPMAXS input file.
`BRANCH_INFO␣`	Dir	$iph \geq 3$		Information related to the current branch calculation.
`TH_DATA␣␣␣␣␣`	Dir($N_{\rm bu}$)	$iph \geq 2$		Information related to the invariant T/H data block. Each componant of the list is a directory containing TH data for a single burnup point.

Table 106: Records and sub-directories in /SAPHYB_INFO/

Name	Type	Condition	Units	Comment
`PKEY_INFO␣␣␣`	Dir(6)			Each component of the list is a directory containing information for all possible state variables.
`STATE_VAR␣␣␣`	C($N_{\rm pk}$) ∗ 12			Name of state variables (`PKEY`).
`ISOTOPES␣␣␣␣`	C(4) ∗ 12			Reference name of isotopes in the saphyb object for depletion chain, recovery of fission yield and number densities. 1) Xe135 2) I135 3) Sm 149 4) Pm149.
/svname/	R(N_k)			Values for each state variable specified by `PKEY`$_k$, N_k is the number of value taken for the kth PKEY.

continued on next page

Records and sub-directories in /SAPHYB_INFO/ continued from last page

Name	Type	Condition	Units	Comment
`PKIDX␣␣␣␣␣␣␣`	I($N_{\rm pk}$)			Correspondance of indices between SAP/MCO and GenPMAXS state variables
`ADF_TYPE␣␣␣␣`	C∗3	ladf		Type of ADF recovered (`DRA`, `GET` or `SEL`).
`HADF␣␣␣␣␣␣␣␣`	C($N_{\rm adf}$) ∗ 8	ladf		Name of the ADF-type boundary flux edit to be recovered from Multicompo
`CDF_TYPE␣␣␣␣`	C∗3	lcdf		Type of CDF recovered (only `DRA`).
`HCDF␣␣␣␣␣␣␣␣`	C($N_{\rm adf}$) ∗ 8	lcdf		Name of the CDF-type boundary flux edit to be recovered from Multicompo.
`GFF_TYPE␣␣␣␣`	C∗3	lgff		Type of GFF recovered (only `DRA`).
`YLD_OPT␣␣␣␣␣`	C∗3	lyld		Type of fission yield to be recovered (`FIX`,`MAN` or `DEF`).
`YLD_FIX␣␣␣␣␣`	R(3)	lyld		User defined values for fission yields (1:I, 2:XE, 3:PM)
`YLD_LOC␣␣␣␣␣`	R(6)	lyld		Values for state parameter on which fission yields are calculated.

Each component of the list `PKEY_INFO` is a directory containing for all possible state variables (1=`BARR`, 2=`DMOD`, 3=`CBOR`, 4=`TCOM`, 5=`TMOD`, 6=`BURN`). Inside each groupwise directory, the following records will be found:

Table 107: Records in /PKEY_INFO/

Name	Type	Condition	Units	Comment
`LFLAG␣␣␣␣␣␣␣`	I(1)			Indicates if the corresponding state variable can be found in the `SAP` (or `MCO`) object.
`NAME␣␣␣␣␣␣␣␣`	C*12	lflag		Name of the state parameter in the `SAP` (or `MCO`) object. For `BARR` and `BURN` state variables, this record exists even if `LFLAG` is *false.*

Table 108: Records and sub-directories in /HELIOS_HEAD/

Name	Type	Condition	Units	Comment
`FILE_CONT_1␣`	R(2)			Set of data for `FILE_CONT_1` block in `DRA` file : Heavy metal Density(`HM_Dens`), Bypass Density(`ByPass`).
`FILE_CONT_2␣`	R(8)			Set of data for `FILE_CONT_2` block in `DRA` file : Lower Energy Limits of Neutron Groups.
`FILE_CONT_3␣`	R(7)			Set of data for `FILE_CONT_3` block in `DRA` file : Volume of coolant (`VCool`), moderator (`VModr`), control rods (`VCnRd`), fuel(`VFuel`), cladding (`VClad`), channels (`VChan`), water (`VWatR`).
`FILE_CONT_4␣`	R(3)			Set of data for `FILE_CONT_4` block in `DRA` file : Cell Pitch and X,Y Position of First Cell.
`XS_CONT␣␣␣␣␣`	R(1)			Set of data for `XS_CONT` block in `DRA` file : (`VFCM`).

Table 109: Records and sub-directories in /GENPMAXS_INP/

Name	Type	Condition	Units	Comment
`FLAG␣␣␣␣␣␣␣␣`	I(1)			Indicates the end of a branch calculation.
`JOB_TIT␣␣␣␣␣`	C∗16			Name of final PMAXS file.
`FILE_NAME␣␣␣`	C∗12			Name of output `DRA` file
`DERIVATIVE␣␣`	C∗4			Set of data for `FILE_CONT_3` block in `DRA` file : Volume of coolant (`VCool`), moderator (`VModr`), control rods (`VCnRd`), fuel(`VFuel`), cladding (`VClad`), channels (`VChan`), water (`VWatR`).
`VERSION␣␣␣␣␣`	R(1)			Version of PARCS used.
`COMMENT␣␣␣␣␣`	C∗40			Free comment.
`JOB_OPT␣␣␣␣␣`	$C(14) * 1$			Options for GenPMAXS running (see [?]).
`DAT_SRC␣␣␣␣␣`	R(5)			Data source information (see [?]).

Each component of the list `TH_DATA` is a directory containing TH data for a single burnup point. Inside each groupwise directory, the following records associated will be found:

Table 110: Records in sub-directory /TH_DATA/

Name	Type	Condition	Units	Comment
`CHI␣␣␣␣␣␣␣␣␣`	R(N_{gr})	*lchi*		The steady-state energy spectrum of the neutron emitted by fission χ.
`OVERV␣␣␣␣␣␣␣`	R(N_{gr})	*linv*	s.cm^{-1}	The average of the inverse neutron velocity.
`LAMBDA␣␣␣␣␣␣`	R(N_{del})	*lamb*	s^{-1}	Decay constant of delayed neutron.
`BETA␣␣␣␣␣␣␣␣`	R(N_{del})	*lbet*		Effective delayed neutron fraction.
`YLDPm␣␣␣␣␣␣␣`	R(1)	*lyld*		Effective yield value of Pm-149
`YLDXe␣␣␣␣␣␣␣`	R(1)	*lyld*		Effective yield value of Xe-135
`YLDI␣␣␣␣␣␣␣␣`	R(1)	*lyld*		Effective yield value of I-135

Table 111: Records and sub-directories in sub-directory /BRANCH_INFO/

Name	Type	Condition	Units	Comment
`BRANCH_IT␣␣␣`	I(1)			Index of the current branch type calculation.
`BRANCH␣␣␣␣␣␣`	C∗4			Current BRANCH type calculation.
`BRANCH_NB␣␣␣`	I(1)			Number of branches to be created in PMAXS files.
`REF_STATE␣␣␣`	R($N_{\mathrm{pk}}-1$)			Values of state variables for reference state, burnup excepted.
`HST_STATE␣␣␣`	R($N_{\mathrm{pk}}-1$)			Values of state variables for history state, burnup excepted.
`STATE_INDEX␣`	I(N_{pk})			Current index value for each state variables.
`BRANCH_INDEX`	I(N_{pk})			Index of current branch calculation.
`STATE␣␣␣␣␣␣␣`	R(N_{pk})			State variable values for the current branch calculation.
`REWIND␣␣␣␣␣␣`	I(1)			Indicates the end of a branch calculation (`REWIND`=1).
`STOP␣␣␣␣␣␣␣␣`	I(1)			Indicates the end of calculation (`STOP`=1).
`CROSS_SECT␣␣`	Dir(N_{bu})			Each component of the list is a directory containing cross sections for all burnup point of a specific branch(cross sections are overwrited after each branch calcualtion).
`DIVERS␣␣␣␣␣␣`	Dir(N_{bu})	$I_{grid} \leq 1$		Each component of the list is a directory containing the `DIVERS` data block recovered from the saphyb object(inforamtions are overwrited after each branch calcualtion).

Each component of the list `CROSS_SECT` is a directory containing cross sections for all burnup point of a specific branch. Cross sections for a single burnup points are filled seqentially during the branching calculation. Inside each groupwise directory, the following sub-directories will be found:

Table 112: Sub-directories in /CROSS_SECT/

Name	Type	Condition	Units	Comment
`MACROLIB_XS␣`	Dir			Directory for macroscopic cross sections for a given calculated branch (cross sections are overwrited after each branch calculation).
`MICROLIB_XS␣`	Dir()	*lxes*		Directory for microscopic cross sections for a given calculated branch (cross sections are overwrited after each branch calculation).

Table 113: Records in the sub-directory /MACROLIB_XS/

Name	Type	Condition	Units	Comment
`XTR␣␣␣␣␣␣␣␣␣`	R($N_{\rm gr}$)		cm^{-1}	The transport cross section Σ_f^g: $\Sigma_t^g = 1/(3 \cdot D_g)$.
`ABSORPTION␣␣`	R($N_{\rm gr}$)		cm^{-1}	The absorption cross section Σ_a^g: $\Sigma_a^g = \Sigma_{tot}^g - \sum_{g'} \Sigma_s^{g \to g'}$.
`KAPPA_FI␣␣␣␣`	R($N_{\rm gr}$)		cm^{-1}	The product of Σ_f^g, the fission cross section with the energy emitted by this reaction κ : $\kappa\Sigma_f^g$.
`X_NU_FI␣␣␣␣␣`	R($N_{\rm gr}$)		cm^{-1}	The product of Σ_f^g, the fission cross section with the averaged number of fissionemitted delayed neutron ν : $\nu\Sigma_f^g$.
`SCAT␣␣␣␣␣␣␣␣`	R($N_{\rm gr} * N_{\rm gr}$)		cm^{-1}	Scattering cross section elements from group g to g' : $\Sigma_s^{g \to g'}$.
`DET␣␣␣␣␣␣␣␣␣`	R($N_{\rm gr}$)	*ldet*		The detector response parameter, it is product of cross section and local flux ratio.
`SFI␣␣␣␣␣␣␣␣␣`	R($N_{\rm gr}$)	*lxes*	cm^{-1}	The fission cross section : Σ_f^g.
`ADF␣␣␣␣␣␣␣␣␣`	R($N_{\rm adf}, N_{\rm gr}$)	*ladf*		Assembly Discontinuity Factors.
`CDF␣␣␣␣␣␣␣␣␣`	R($N_{\rm cdf}, N_{\rm gr}$)	*lcdf*		Corner Discontinuity Factors.
`GFF␣␣␣␣␣␣␣␣␣`	R($N_{\rm pin} * N_{\rm pin}, N_{\rm gr}$)	*lgff*		Group-wise Form Function.

Table 114: Records in the sub-directory /MICROLIB_XS/

Name	Type	Condition	Units	Comment
`XENG␣␣␣␣␣␣␣␣`	$\mathrm{R}(N_{\mathrm{gr}})$		cm^{-2}	The microscopic absorption cross section of Xenon: $\sigma^g_{a,Xe} = \sigma^g_{tot,Xe} - \sum_{g'} \Sigma^{g\to g'}_{s,Xe}$.
`SMNG␣␣␣␣␣␣␣␣`	$\mathrm{R}(N_{\mathrm{gr}})$		cm^{-2}	The microscopic absorption cross section of Samarium: $\sigma^g_{a,Sm} = \sigma^g_{tot,Sm} - \sum_{g'} \Sigma^{g\to g'}_{s,Sm}$.
`XEND␣␣␣␣␣␣␣␣`	$\mathrm{R}(N_{\mathrm{gr}})$		$(\mathrm{b.cm})^{-1}$	The Xenon number density: n_{Xe}.
`SMND␣␣␣␣␣␣␣␣`	$\mathrm{R}(N_{\mathrm{gr}})$		$(\mathrm{b.cm})^{-1}$	The Samarium number density: n_{Sm}.

Each component of the list `DIVERS` is a directory containing informations for all burnup points of a specific branch recovered from the `DIVERS` block of the saphyb object. Inside each groupwise directory, the following records will be found:

Table 115: Records in the sub-directory /DIVERS/

Name	Type	Condition	Units	Comment
`B2␣␣␣␣␣␣␣␣␣␣`	R(1)		cm^{-2}	Critical buckling.
`KEFF␣␣␣␣␣␣␣␣`	R(1)			Effective multiplication factor.
`KINF␣␣␣␣␣␣␣␣`	R(1)			Infinite multiplication factor.

9 EXAMPLES

The following examples of input files represent a typical core modeling using DONJON. The main characteristics of a simplified design for the ACR-700 benchmark core model are given below.

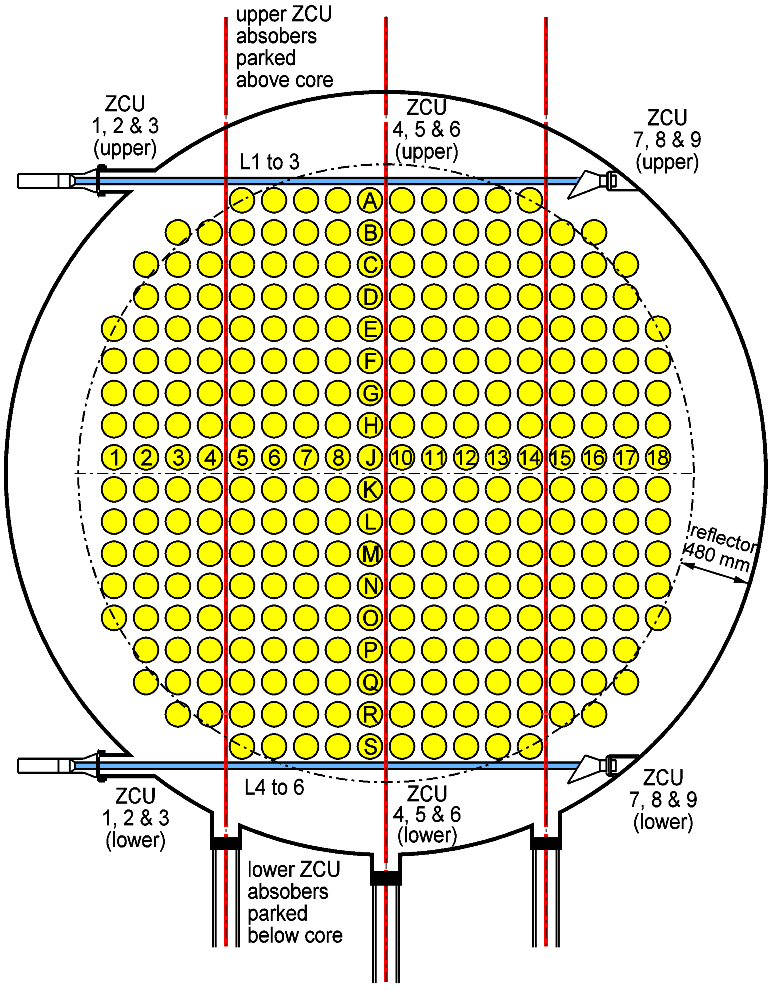

Figure 12: Face View of ACR Benchmark Core Model (292 Channels)

- Number of reactor channels 292
- Number of fuel bundles per channel 12
- Core radius 260 cm
- Core length 594.36 cm
- Lattice pitch 22 cm
- Reactor thermal power 1800 MW(th)