UK Atomic Energy Authority

UKAEA-CCFE-RE(21)03
February 2021

Editors:
Greg Bailey
David Foster
Priti Kanth
Mark Gilbert

The FISPACT-II API User Manual

Email: publicationsmanager@ukaea.uk

The FISPACT-II API User Manual

Editors:
Greg Bailey
David Foster
Priti Kanth
Mark Gilbert

February 2021

UK Atomic Energy Authority
Culham Science Centre
Abingdon
Oxfordshire
OX14 3DB

Contact

UK Atomic Energy Authority
Culham Science Centre
Abingdon
Oxfordshire
OX14 3DB
United Kingdom

Telephone: +44 (0)1235-466884
email: admin@fispact.ukaea.uk
website: http://fispact.ukaea.uk

Disclaimer

Neither the authors nor the United Kingdom Atomic Energy Authority accept responsibility for consequences arising from any errors either in the present documentation or the FISPACT-II code, or for reliance upon the information contained in the data or its completeness or accuracy.

Acknowledgement

This work was funded by the UK EPSRC under grant EP/P012450/1

CCFE is the fusion research arm of the United Kingdom Atomic Energy Authority.

CCFE is certified to ISO 9001 and ISO 14001.

Executive Summary

This document is the User Manual for the FISPACT-IIVersion 5.0 API.

Contents

List of Figures

List of Tables

This page has been left intentionally blank.

1 Introduction

FISPACT-II is an inventory simulation code used to study the nuclide evolution and subsequent radiological quantities of a material composition during and after irradiation. FISPACT-II is capable of using modern nuclear cross section and decay data libraries in calculations of neutron, charged particle and photonic irradiation. Previous versions of FISPACT-II have been supplied as an executable, requiring file based inputs, to be called from the command line. The latest edition of FISPACT-II also includes a complete API to allow the code to be used in a variety of complex and adaptable ways.

This document gives a detailed overview of how to perform an inventory simulation with the FISPACT-II API. Exact details of the associated methods are provided in the API code reference sheets included alongside this manual.

1.1 The API approach

The traditional and commonplace method of configuring Multi-physics and Monte Carlo (MC) codes, such as FISPACT-II, is to create and process an 'input' file, or series of input files. Typically these inputs are text based, and do not conform to any general standard or schema between different codes. Whilst this may be simple and easy for humans to read and process, it becomes difficult for clients of the codebase to process complex jobs, causes issues when trying to scale simulations, and comes with a performance cost due to serialisation.

An Application Programming Interface (API) using an object design, can avoid these issues and allow users to develop complex simulations and interface with other codes. In computer programming, an API can be considered as the suite of functions, commands, subroutines, protocols, objects and/or tools for building software and interacting with an application. Generally, it is the set of clearly defined methods, or interfaces, of communication among various components. An API has been developed for FISPACT-II to provide users with a straight forward method of performing multiple complex simulations by interfacing with the FISPACT-II engine in a programmatic fashion. The API is available for Fortran, C, C++ and Python.

1.2 Applications of Fispact-II API

The traditional command line driven version of FISPACT-II provides rapid inventory simulations but when studying complex systems, where many calculations are required, it can be cumbersome. For a complete activation calculation FISPACT-II requires a minimum of 3 files (an INPUT file, a FLUXES file and a FILES file) and while some calculations may share some files producing the required inputs for many simulations can be time consuming or require a user to develop their own script-based solutions.

The FISPACT-II API removes this problem. A single script can be used to execute a range of inventory simulation and produce as many, or as few, outputs as a user desires. The API approach can also aid with the performance overheads of running many calculations, because the nuclear data can be loaded

in to memory once whereas each traditional calculation would require the nuclear data to be loaded individually.

The API provides the user a with a series of straight forward methods to control several aspects of an activation calculations which are not available or require additional steps with the standard execution. This includes complete control over elemental data and/or nuclear data, the ability to chain calculations together and direct access to calculated outputs. But, in this beta release, some functionality is missing. Particularly the decay chain analysis.

2 API Architecture

The FISPACT-II API is broken down into five modular and independent data objects, which map loosely to the conventional input and output files.

- **INPUT DATA** - The input object tracking all user input state, such as flux definition, irradiation schedule and run settings. It contains data conventionally defined in the input file (*.i*) and *fluxes* file.
- **OUTPUT DATA** - The output object, contains all output data associated with a FISPACT-II simulation, such as gamma spectrum and inventories. It contains data conventionally defined in the output files, such as the main *.out*, *.tab*, *.gra* and *.json* files.
- **NUCLEAR DATA** - The nuclear data object, provides accessors to reaction and decay data. It does not map to a conventional FISPACT-II input or output file.
- **NUCLEAR DATA READER** - The nuclear data reader object, provides accessors to file paths for loading the nuclear data in ENDF and EAF formats. It contains data conventionally defined in the *files* file.
- **MONITOR** - The error handling object. A dynamic data structure which contains data conventionally defined in the *.log* file.

The basic structure of the API is demonstrated in figure 1. Here *FispactProcess* in the engine which performers the collapse and solves the rate equations which provide the transmutation results. Alongside these a number of utilities and tools have also been implemented to aid an enhance functionality.

- **ELEMENTAL DATA** - A set of methods for interfacing and altering the elemental data traditionally hard coded in FISPACT-II.
- **GROUP CONVERT** - Routines for converting the group structure of a flux spectrum. Analogous to the functionality of the GRPCONVERT keyword.
- **UTILITIES** - A number of methods which aid API usage such give radionuclide's ZAI or proton number from its name and the energies bins for nuclear data group structure.
- **CONTROL CONSTANTS** - Values are defined for interfacing with API routines and providing common data structures.

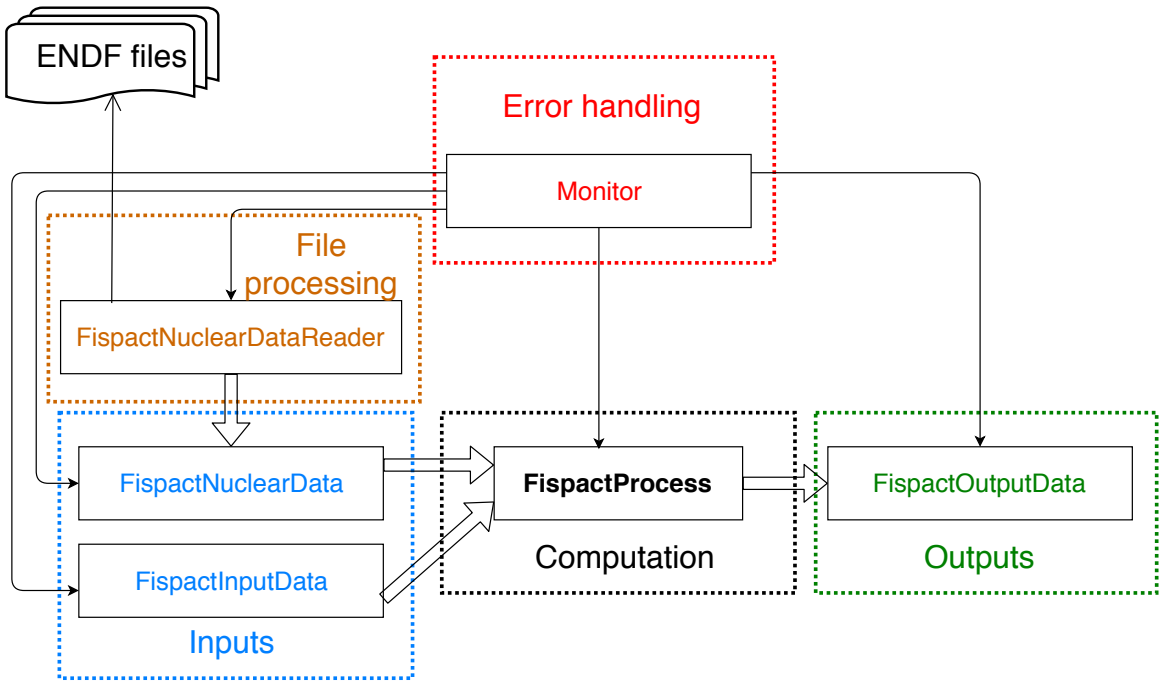

Figure 1: The high level architecture of the API.

3 Libraries and Compiling

The FISPACT-II API has been provided as a complied libraries file (alongside headers for the C and C++ variants) so that it can be included in a users project in a similar manner to other code libraries, such as the liner algebra suite LAPACK.

3.1 Fortran

For static libraries use the following, where *FC* is the Fortran compiler of your choosing.

```
${FC} -static-libstdc++ /path/to/fispactapi.f90 -L /path/to/fispact/libs/ \
      -lfispactapi -lfispact -lmonitor -ljsonfortran -lstdc++ \
       main.f90 \
       -o main
```

When cross compiling, for example if using Intel compiled libraries with a GNU compiler for the program, or vice versa, ensure that static libraries for that compiler are included, otherwise link issues will arise.

For shared libraries use the following, where *FC* is the Fortran compiler of your choosing.

```
${FC} -static-libstdc++ /path/to/fispactapi.f90 -L /path/to/fispact/libs/ \
```

```
      -lfispactapi -lmonitor \
       main.f90 \
       -o main
```

The latter is simpler to compile and only needs to link against the API level libraries. It should be noted that when compiling on MacOS systems using the native CLang compiler linking errors can be encountered due. These can be overcome by pointing to the CLang libraries at compile time or using a third party tool to handle the linking, such as CMake.

Once compiled the program can be run via:

```
main /path/to/nuclear_data
```

if the nuclear data path is to read from the command line. If the path has been hard coded the option should be omitted.

3.1.1 Fortran KINDs

The Fortran API includes a number of definitions of Fortran KINDs which are expected for types input into the API methods. The Fortran API makes use of 4 byte integers and 8 byte real, these are defined in table 1. These are implemented into any Fortran API script as standard KIND definitions and user defined equivalent an be also be used.

Table 1: Fortran API KIND parameters

Type	KIND	Use in code
integer	selected_int_kind(9)	integer(ki4), 1_ki4
real	selected_real_kind(15)	real(kr8), 1.0_kr8

3.2 C

For static libraries use the following, where *CC* is the C compiler of your choosing.

```
${CC} -static-libstdc++ -I /path/to/fispact/c/headers -L /path/to/fispact/libs/ \
      -lfispactapi -lfispact -lmonitor -ljsonfortran -lstdc++ \
       main.c \
       -o main
```

When cross compiling, for example if using Intel compiled libraries with a GNU compiler for the program, or vice versa, ensure that static libraries for that compiler are included, otherwise link issues will arise. When compiling on MacOS issues may arise as MacOS does not support dynamic linking, as such the Fortran libraries used to compile the API libraries need to be provided.

For shared libraries use the following, where *CC* is the C compiler of your choosing.

```
${CC} -static-libstdc++ -I /path/to/fispact/c/headers -L /path/to/fispact/libs/ \
      -lfispactapi -lmonitor \
       main.c \
       -o main
```

The latter is again simpler to compile and only needs to link against the API level libraries.

Once compiled the program can be run via:

```
main /path/to/nuclear_data
```

if the nuclear data path is to read from the command line. If the path has been hard coded the option should be omitted.

3.3 C++

For static libraries use the following, where *CXX* is the C++ compiler of your choosing.

```
${CXX} -static-libstdc++ -std=c++14 \
      -I /path/to/fispact/c/headers -I /path/to/fispact/cpp/headers \
      -I /path/to/spdlog/include \
      -L /path/to/fispact/libs/ \
      -lfispactapi -lfispact -lmonitor -ljsonfortran -lstdc++ \
       main.cpp \
       -o main
```

Again, when cross compiling, for example if using Intel compiled libraries with a GNU compiler for the program, or vice versa, ensure that static libraries for that compiler are included, otherwise link issues will arise. Note that the C++ headers use the C headers and hence both must be included when compiling. The FISPACT-II C++ API makes use of the spdlog headers, so these must also be included as part of the compilation.

For shared libraries use the following, where *CXX* is the C++ compiler of your choosing.

```
${CXX} -static-libstdc++ -std=c++14 \
      -I /path/to/fispact/c/headers -I /path/to/fispact/cpp/headers \
      -L /path/to/fispact/libs/ \
      -lfispactapi -lmonitor \
       main.cpp \
       -o main
```

The latter is again simpler to compile and only needs to link against the API level libraries.

Once compiled the program can be run via:

```
main /path/to/nuclear_data
```

if the nuclear data path is to read from the command line. If the path has been hard coded the option should be omitted.

3.4 Python

The Python bindings do not require the user to compile anything, and simply require the single *pyfispact* shared library. Static libraries are not possible with Python. It is then simple to import the library, via:

```
#!/usr/bin/env python3
import pyfispact as pf
....
```

Note that the *pyfispact.so* library file must be placed on the users python path in order to successfully import. If not the sys module can be used to append a python path in teh script itself, for example:

```
#!/usr/bin/env python3
import sys
sys.path.append(path/to/pyfispact.so)
import pyfispact as pf
....
```

Using a script is simple via:

```
python script.py
```

4 Including and Initialising the API

Before the the functionality of the FISPACT-II API can be used the objects, classes and the global data (default elemental abundances etc) are required to be initialised. It should be noted that while the examples shown assume that a complete inventory simulation is to be performed, an user only needs to initialise what is required for their use case.

For example, if a script is only to probe the nuclear data available and not perform a simulation only the base initialise and nuclear data object need to be included. Likewise if only utility methods and tools, such as the group convert methods (see section 10.1), are being implement only the base initialise need to be completed.

4.1 Fortran

The Fortran API is included as a module and a number of subroutines are called to initialise the required objects and global data. Below the input, output and nuclear data related objects are initialised alongside the monitor (error handling).

```
program API_example
    ! import fispact api library
    use fispactapi
    implicit none
    ! define types and variables
    ! required for complete API functionality
        ...
    type(c_ptr) :: monitor
    type(c_ptr) :: input_data
    type(c_ptr) :: output_data
    type(c_ptr) :: nuclear_data
    type(c_ptr) :: nuclear_data_reader
        ...
    ! initialse objects and global data
    monitor = MonitorCreate()
    input_data = FispactInputDataCreate(monitor)
    output_data = FispactOutputDataCreate(monitor)
    nuclear_data = FispactNuclearDataCreate(monitor)
    nuclear_data_reader = FispactNuclearDataReaderCreate(monitor)
    call MonitorSetVerbosityLevel(monitor, MONITOR_SEVERITY_TRACE)

    ! initialise static and global data
    call FispactInitialise(monitor)
        ...
```

4.2 C

When using the API in C the relevant headers are included and the required classes initialised in similar manner to that seen in the Fortran example above.

```
    #include "fispactapi.h"
        ...
    MONITOR_PTR_VALUE monitor;
    FISPACT_INPUT_PTR_VALUE input_data;
    FISPACT_OUTPUT_PTR_VALUE output_data;
    FISPACT_ND_PTR_VALUE nuclear_data;
    FISPACT_ND_READER_PTR_VALUE nd_reader;

    // initialise
    monitor = MonitorCreate();
    FispactInitialise(monitor);
    input_data = FispactInputDataCreate(monitor);
    output_data = FispactOutputDataCreate(monitor);
    nuclear_data = FispactNuclearDataCreate(monitor);
    nd_reader = FispactNuclearDataReaderCreate(monitor);

    // set the minimum verbosity default to lowest level - trace
    MonitorSetVerbosityLevel(monitor, MONITOR_SEVERITY_TRACE);
    ...
```

4.3 C++

Making use of the API in C++ required a suite of header files to be included. For course only the header files which are relevant to the API's which will be used need to be included.

```
#include "common.hpp"
#include "monitor.hpp"
#include "exceptions.hpp"

#include "fispactnuclеardata.hpp"
#include "fispactinputdata.hpp"
#include "fispactoutputdata.hpp"
#include "fispactcompute.hpp"
#include "fispactgroupstructures.hpp"
#include "fispactgroupconvert.hpp"
#include "fispactelementaldata.hpp"
#include "fispactutil.hpp"

namespace fp = fispact;
...
        fp::FispactMonitor monitor(run_name + ".log");
        fp::FispactMonitor::CMonitor& mpp = monitor.native(); // c++ native type

        // fispact data types
        fp::NuclearData nd(monitor);
        fp::io::NuclearDataReader nd_reader(monitor);
        fp::InputData input(monitor);
        fp::OutputData output(monitor);

        // set the minimum verbosity default to lowest level - trace
        mpp.setVerbosityLevel(fp::severity::level::trace);

        fp::GlobalInitialise(monitor);
...
```

4.4 Python

When using Python the initialisation is performed in a similar manner to the other languages. Unlike the other languages the monitor and the initialise methods need to be done at start of the script. The other initialisation can be done as and when required.

```
import pyfispact as pf

m = pf.Monitor()
pf.initialise(m)
...
nd = pf.NuclearData(m)
ndr = pf.io.NuclearDataReader(m)
ip = pf.InputData(m)
o = pf.OutputData(m)
...
```

5 Nuclear Data

The APIs which cover the nuclear data handling are split into two objects: the NuclearData object and the NuclearDataReader object. As names suggest the NuclearDataReader handles the reading of files and fills the NuclearData object itself. The NuclearData object is then passed to the *FispactProcess* (which runs the FISPACT-II engine, detailed in a later section) and any other methods which require access to the nuclear data. This separation allows the FISPACT-II API to be adaptable: if a user has nuclear data that they wish to use but is not in a format known to FISPACT-II, the user can write there own readers to fill the NuclearData object.

A major advantage the API approach has over the traditional methods is that once loaded the nuclear data can be used multiple times. For example an API script can pass a NuclearData object to several inventory simulations, streamlining calculations which could require many FISPACT-II runs. It should also be noted that multiple NuclearData objects can be filled with different data. This could be used to perform the same simulation with different nuclear data to assess sensitivities for example.

Once loaded, the nuclear data can be probed with a set APIs from the NuclearData object. These give the user access to any data stored by FISPACT-II via series of getter methods. Also any of the loaded data can be altered via series of setter methods, these allow the user to alter the nuclear data they are using for specific nuclides if desired. This section will detail various methods available to each object and examples as to there use. A full breakdown of the objects and the associated routine is given the relevant code reference sheets.

5.1 Nuclear Data keys and constants

The nuclear data objects make use of a series of integer keys to determine which file units should be used, which projectile has been selected or which decay spectra is desired. The tables in this section detail these keys and there use is explained in the forthcoming sections on the FISPACT-II API nuclear data objects. Table 2 contains the file keys for the nuclear data reader object (these align with those used in the FILES file required with traditional command line use of FISPACT-II). Table 3 contains the integer keys for projectile selection. Table 4 shows the integer keys for decay spectra retrieval (these mirror the identifiers used for decay data in the ENDF6 file format).

Table 2: Nuclear Reader file keys

Data	Fortran, C, C++
Nuclide index	FISPACT_ND_IND_NUC_KEY
Hazards	FISPACT_ND_HAZARDS_KEY
Absorp (γ Attenuation)	FISPACT_ND_ABSORP_KEY
Clarence	FISPACT_ND_CLEAR_KEY
A2 (Transport)	FISPACT_ND_A2DATA_KEY
Enbins	FISPACT_ND_ENBINS_KEY
Decay (EAF)	FISPACT_ND_DECAY_KEY
Deacy (ENDF)	FISPACT_ND_DK_ENDF_KEY
Probability Table	FISPACT_ND_PROB_TAB_KEY
Links between fissionable nuclides and yields	FISPACT_ND_ASSCFY_KEY
Fission Yield (EAF)	FISPACT_ND_FISSYLD_KEY
Instantaneous Fission Yield	FISPACT_ND_FY_ENDF_KEY
Spontaneous Fission Yield	FISPACT_ND_SF_ENDF_KEY
Spontaneous Fission Yield	FISPACT_ND_SP_ENDF_KEY
ENDF merge	FISPACT_ND_XS_EXTRA_KEY
Cross Sections (EAF)	FISPACT_ND_CROSSEC_KEY
Cross Sections Uncertainty (EAF)	FISPACT_ND_CROSSUNC_KEY
Cross Sections (ENDF)	FISPACT_ND_XS_ENDF_KEY
Cross Sections (ENDF binary)	FISPACT_ND_XS_ENDFB_KEY

Data	Python
Nuclide index	pyfisapct.io.ND_IND_NUC_KEY()
Hazards	pyfisapct.io.ND_HAZARDS_KEY()
Absorp (γ Attenuation)	pyfisapct.io.ND_ABSORP_KEY()
Clarence	pyfisapct.io.ND_CLEAR_KEY()
A2 (Transport)	pyfisapct.io.ND_A2DATA_KEY()
Enbins	pyfisapct.io.ND_ENBINS_KEY()
Decay (EAF)	pyfisapct.io.ND_DECAY_KEY()
Deacy (ENDF)	pyfisapct.io.ND_DK_ENDF_KEY()
Probability Table	pyfisapct.io.ND_PROB_TAB_KEY()
Links between fissionable nuclides and yields	pyfisapct.io.ND_ASSCFY_KEY()
Fission Yield (EAF)	pyfisapct.io.ND_FISSYLD_KEY()
Instantaneous Fission Yield	pyfisapct.io.ND_FY_ENDF_KEY()
Spontaneous Fission Yield	pyfisapct.io.ND_SF_ENDF_KEY()
Spontaneous Fission Yield	pyfisapct.io.ND_SP_ENDF_KEY()
ENDF merge	pyfisapct.io.ND_XS_EXTRA_KEY()
Cross Sections (EAF)	pyfisapct.io.ND_CROSSEC_KEY()
Cross Sections Uncertainty (EAF)	pyfisapct.io.ND_CROSSUNC_KEY()
Cross Sections (ENDF)	pyfisapct.io.ND_XS_ENDF_KEY()
Cross Sections (ENDF binary)	pyfisapct.io.ND_XS_ENDFB_KEY()

Table 3: Nuclear Data projectile keys

Projectile	**Fortran, C, C++**
Neutron	FISPACT_PROJECTILE_TYPE_NEUTRON
Deuteron	FISPACT_PROJECTILE_TYPE_DEUTERON
Proton	FISPACT_PROJECTILE_TYPE_PROTON
Alpha	FISPACT_PROJECTILE_TYPE_ALPHA
Gamma	FISPACT_PROJECTILE_TYPE_GAMMA
Triton	FISPACT_PROJECTILE_TYPE_TRITON
Helion	FISPACT_PROJECTILE_TYPE_HELION
Data	**Python**
Neutron	pyfispact.PROJECTILE_NEUTRON()
Deuteron	pyfispact.PROJECTILE_DEUTERON()
Proton	pyfispact.PROJECTILE_PROTON()
Alpha	pyfispact.PROJECTILE_ALPHA()
Gamma	pyfispact.PROJECTILE_GAMMA()
Triton	pyfispact.PROJECTILE_TRITON()
Helion	pyfispact.PROJECTILE_HELION()

Table 4: Nuclear Data decay spectrum keys

Spectrum Type	**Fortran, C, C++**
Gamma	FISPACT_SPECTRUM_TYPE_GAMMA
Beta minus	FISPACT_SPECTRUM_TYPE_BETA
Beta plus/Electron Capture	FISPACT_SPECTRUM_TYPE_EC
Unknown	FISPACT_SPECTRUM_TYPE_UNKNOWN
Alpha	FISPACT_SPECTRUM_TYPE_ALPHA
Neutron	FISPACT_SPECTRUM_TYPE_NEUTRON
Spontaneous Fission	FISPACT_SPECTRUM_TYPE_SF
Proton	FISPACT_SPECTRUM_TYPE_PROTON
Discrete Electrons	FISPACT_SPECTRUM_TYPE_ELECTRON
X ray	FISPACT_SPECTRUM_TYPE_X_RAY
Data	**Python**
Gamma	pyfispact.SPECTRUM_TYPE_GAMMA()
Beta minus	pyfispact.SPECTRUM_TYPE_BETA()
Beta plus/Electron Capture	pyfispact.SPECTRUM_TYPE_EC()
Unknown	pyfispact.SPECTRUM_TYPE_UNKNOWN()
Alpha	pyfispact.SPECTRUM_TYPE_ALPHA()
Neutron	pyfispact.SPECTRUM_TYPE_NEUTRON()
Spontaneous Fission	pyfispact.SPECTRUM_TYPE_SF()
Proton	pyfispact.SPECTRUM_TYPE_PROTON()
Discrete Electrons	pyfispact.SPECTRUM_TYPE_ELECTRON()
X ray	pyfispact.SPECTRUM_TYPE_X_RAY()

5.2 Reading Nuclear Data

As with the command line version of FISPACT-II, the API requires that the nuclear data is read in and stored in memory for an inventory calculation to be performed. To read in the nuclear data a user needs to provide the paths to the data. These paths are the same as those given in the FILES file used with the traditional FISPACT-II method and the API uses the standard nuclear data provided with FISPACT-II.

Regardless of which language is being used, the paths to each type (cross-sections, decay, fission yields etc) of data needs to be set in the NuclearDataReader object. Each type of nuclear data has a key associated with it (detailed in table 2) which will allow the API to determine what is being loaded. After all paths have been set the data is loaded by calling the *load* method. The projectile type can be set in the NuclearData object using the keys given in table 3, by default FISPACT-II assumes neutron irradiation.

The NuclearDataReader allows a callback routine to be defined. This will print to the terminal details and the progress of the data load. This allows the functionality of the MONITOR keyword to be reproduced. Examples of these will also be included in the examples given below, which detail the loading of the TENDL2019 cross section data, the decay2020 decay data with fission yields and radiological data from decay2012 associated files. In the following examples *nuclear_data_path* is the path to the FISPACT-II nuclear data folder on a users system.

5.2.1 Fortran

```
    ...
! callback pointers
type(c_ptr) :: c_state_ptr
type(c_ptr) :: c_nuclide_ptr
type(fispact_nuclear_data_reader_callback_t), pointer :: reader_state_native_ptr
type(fispact_process_callback_t), pointer :: process_state_native_ptr
character(:), allocatable :: lastkey
integer(ki4) :: status
    ...
! load Nuclear Data
! set projectile
call  FispactNuclearDataSetProjectile(nuclear_data, monitor, &
                                      & FISPACT_PROJECTILE_TYPE_NEUTRON)
! loading TENDL2019 and Decay2020 alongside hazards and radiological data
! from Decay2012 files
! set paths to the nuclear data, same paths and keys as those found in a FILES file
! TENDL2019 load, using the decay2020 index file
call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
    & FISPACT_ND_IND_NUC_KEY, &
    & f_c_string(trim(nd_base_path)//"/decay2020/tendl19_decay2020_index"))
call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
    & FISPACT_ND_XS_ENDF_KEY, &
    & f_c_string(trim(nd_base_path)//"/TENDL2019data/gendf-1102"))
call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
    & FISPACT_ND_PROB_TAB_KEY, &
```

```
    & f_c_string(trim(nd_base_path)//"/TENDL2019data/tp-1102-294"))
! fission yeild data
call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
    & FISPACT_ND_FY_ENDF_KEY, &
    & f_c_string(trim(nd_base_path)//"/GEFY61data/gefy61_nfy"))
call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
    & FISPACT_ND_SF_ENDF_KEY, &
    & f_c_string(trim(nd_base_path)//"/GEFY61data/gefy61_nfy"))
! Deacy2020 decay data
call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
    & FISPACT_ND_DK_ENDF_KEY, &
    & f_c_string(trim(nd_base_path)//"/decay2020/decay_2020"))
! attenuation data required for dose rate calculation
call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
    & FISPACT_ND_ABSORP_KEY, &
    & f_c_string(trim(nd_base_path)//"/decay/abs_2012"))
! Hazards and radiological data from decay2012 files
call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
    & FISPACT_ND_HAZARDS_KEY, &
    & f_c_string(trim(nd_base_path)//"/decay/hazards_2012"))
call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
    & FISPACT_ND_CLEAR_KEY, &
    & f_c_string(trim(nd_base_path)//"/decay/clear_2012"))
call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
    & FISPACT_ND_A2DATA_KEY, &
    & f_c_string(trim(nd_base_path)//"/decay/a2_2012"))

! allocate our callback for reader
allocate(reader_state_native_ptr, stat=status)
c_state_ptr = c_loc(reader_state_native_ptr)

! load the nuclear data and cleanup reader, we no longer need it
lastkey = ""
call FispactNuclearDataReaderLoad(nuclear_data_reader, nuclear_data, &
                                  & monitor, c_state_ptr, c_callback_load)
deallocate(reader_state_native_ptr)
    ...
    ! callback for nuclear data reader
    subroutine c_callback_load(c_state) bind(C)
        use, intrinsic :: iso_fortran_env, only : stdout=>output_unit
        type(c_ptr), intent(in), value :: c_state

        type(fispact_nuclear_data_reader_callback_t), pointer :: ptr
        character(:), allocatable :: fkey

        call c_f_pointer(c_state, ptr)

        fkey = trim(c_f_string(ptr%key, FISPACT_MAX_KEY_LENGTH-1))
        if(fkey /= trim(lastkey))then
            write(stdout, '(A)') achar(13)//"  reading "//fkey &
            & //" data from: "// &
            & trim(c_f_string(ptr%path, FISPACT_MAX_PATH_LENGTH-1))//achar(13)
            flush(stdout)
        end if

        ! simple progress bar - could use a better library for this
```

```
            write(stdout, '(A3, I4.1, A1, I4.1, A2)', advance='NO') &
            & achar(13)//" [", ptr%index, "/", ptr%total, "]"//achar(13)
            flush(stdout)
            lastkey = fkey
        end subroutine c_callback_load
        ...
```

The NuclearDataReader object contains methods to allows the reading of cumulative and spontaneous fission yields. For these to be included, flags need to be set in the NuclearDataReader object (with the relevant included methods) before the 'load' is called.

```
        ...
    ! read cumulative fission yield. 1 = True
    call FispactNuclearDataReaderSetFissionYieldCumulativeOption(&
                        & nuclear_data_reader, monitor, 1_ki4)
    ! read spontaneous fission yield. 1 = True
    call FispactNuclearDataReaderSetReadSpontaneousFission(&
                                        & nuclear_data_reader, monitor, 1_ki4)
        ...
```

5.2.2 C

```
        ...
// callback for nuclear data load
void load_callback(FISPACT_NUCLEAR_DATA_LOAD_CALLBACK_DATA* data){
    printf("\33[2K\r%s: %s [%i/%i]", data->key, data->path, data->index, data->total );
    fflush(stdout);
}
        ...
    // loading TENDL2019 and Decay2020 alongside hazards and radiological
    // data from Decay2012 files
    // set projectile
    FispactNuclearDataSetProjectile(nuclear_data, monitor,
                                    FISPACT_PROJECTILE_TYPE_NEUTRON)

    // set paths to the nuclear data, same patsh and keys as those
    // found in a FILES file
    char path[FISPACT_MAX_PATH_LENGTH];
    strcpy(path, nuclear_data_path);
    // TENDL2019 load, using the decay2020 index file
    strcat(path, "/decay2020/tendl19_decay2020_index");
    FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_IND_NUC_KEY, path);
    strcpy(path, nuclear_data_path);
    strcat(path, "/TENDL2019data/gendf-1102");
    FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_XS_ENDF_KEY, path);
    strcpy(path, nuclear_data_path);
    strcat(path, "/TENDL2019data/tp-1102-194");
    FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_PROB_TAB_KEY, path);
    // fission yeild data
    strcpy(path, nuclear_data_path);
    strcat(path, "/GEFY61data/gefy61_nfy");
```

```
FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_FY_ENDF_KEY, path);
strcpy(path, nuclear_data_path);
strcat(path, "/GEFY61data/gefy61_sfy");
FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_SF_ENDF_KEY, path);
// decay2020 decay data
strcpy(path, nuclear_data_path);
strcat(path, "/decay2020/decay_2020");
FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_DK_ENDF_KEY, path);
// attenuation data required for dose rate calculation
strcpy(path, nuclear_data_path);
strcat(path, "/decay/abs_2012");
FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_ABSORP_KEY, path);
//  Hazards and radiological data from decay2012 files
strcpy(path, nuclear_data_path);
strcat(path, "/decay/hazards_2012");
FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_HAZARDS_KEY, path);
strcpy(path, nuclear_data_path);
strcat(path, "/decay/clear_2012");
FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_CLEAR_KEY, path);
strcpy(path, nuclear_data_path);
strcat(path, "/decay/a2_2012");
FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_A2DATA_KEY, path);
// load nuclear data
FISPACT_NUCLEAR_DATA_LOAD_CALLBACK_DATA load_data;
FispactNuclearDataReaderLoad(nd_reader, nuclear_data, monitor,
                                    &load_data, load_callback);
    ...
```

The NuclearDataReader object contains methods to allows the reading of cumulative and spontaneous fission yields. For these to be included, flags need to be set in the NuclearDataReader object (with the relevant included methods) before the 'load' is called.

```
    ...
//  read cumulative fission yield. 1 = True
FispactNuclearDataReaderSetFissionYieldCumulativeOption(nd_reader, monitor, 1);
//  read spontaneous fission yield. 1 = True
FispactNuclearDataReaderSetReadSpontaneousFission(nd_reader, monitor, 1);
    ...
```

5.2.3 C++

```
    ...
// loading TENDL2019 and Decay2020 alongside hazards and radiological
// data from Decay2012 files
// set projectile
nd.setProjectile(FISPACT_PROJECTILE_TYPE_NEUTRON);
// set paths to the nuclear data, same patsh and keys as those
// found in a FILES file
// TENDL2019 load, using the decay2020 index file
nd_reader.setPath(FISPACT_ND_IND_NUC_KEY, nd_base_path +
                                    "/decay2020/tendl19_decay2020_index");
```

```
    nd_reader.setPath(FISPACT_ND_XS_ENDF_KEY, nd_base_path +
                                        "/TENDL2019data/gendf-1102");
    nd_reader.setPath(FISPACT_ND_PROB_TAB_KEY, nd_base_path +
                                        "/TENDL2019data/tp-1102-194");
    // fission yeild data
    nd_reader.setPath(FISPACT_ND_FY_ENDF_KEY, nd_base_path + "/GEFY61data/gefy61_nfy");
    nd_reader.setPath(FISPACT_ND_SF_ENDF_KEY, nd_base_path + "/GEFY61data/gefy61_sfy");
    // decay2020 decay data
    nd_reader.setPath(FISPACT_ND_DK_ENDF_KEY, nd_base_path + "/decay2020/decay_2020");
    // attenuation data required for dose rate calculation
    nd_reader.setPath(FISPACT_ND_ABSORP_KEY, nd_base_path + "/decay/abs_2012");
    //  Hazards and radiological data from decay2012 files
    nd_reader.setPath(FISPACT_ND_HAZARDS_KEY, nd_base_path + "/decay/hazards_2012");
    nd_reader.setPath(FISPACT_ND_CLEAR_KEY, nd_base_path + "/decay/clear_2012");
    nd_reader.setPath(FISPACT_ND_A2DATA_KEY, nd_base_path + "/decay/a2_2012");
    // load nuclear data
    nd_reader.load(nd, &load_callback);
        ...
```

The NuclearDataReader object contains methods to allows the reading of cumulative and spontaneous fission yields. For these to be included, flags need to be set in the NuclearDataReader object (with the relevant included methods) before the 'load' is called.

```
        ...
    // read cumulative fission yield
    nd_reader.setReadFissionYieldCumulative(true);
    // read spontaneous fission yield
    nd_reader.setReadSF(true);
        ...
```

5.2.4 Python

```
        ...
# load nuclear data
def load_nuclear_data(nd_base_path):
    # loading TENDL2019 and Decay2020 alongside hazards and radiological
    # data from Decay2012 files
    # initialise nuclear data object
    nd = pf.NuclearData(m)
    # initialise nuclear data reader object
    ndr = pf.io.NuclearDataReader(m)
    #  set projectile to neutron
    nd.setprojectile(pf.PROJECTILE_NEUTRON())
    # set paths to the nuclear data, same patsh and keys as those
    # found in a FILES file
    # TENDL2019 load, using the decay2020 index file
    ndr.setpath(pf.io.ND_IND_NUC_KEY(),   os.path.join(nd_base_path,
                'decay2020', 'tendl19_decay2020_index'))
    ndr.setpath(pf.io.ND_XS_ENDF_KEY(),   os.path.join(nd_base_path,
                'TENDL2019data', 'gendf-1102'))
    ndr.setpath(pf.io.ND_PROB_TAB_KEY(), os.path.join(nd_base_path,
```

```
                'TENDL2019data', 'tp-1102-194'))
    # fission yeild data
    ndr.setpath(pf.io.ND_FY_ENDF_KEY(),   os.path.join(nd_base_path,
                'GEFY61data', 'gefy61_nfy'))
    ndr.setpath(pf.io.ND_SF_ENDF_KEY(),   os.path.join(nd_base_path,
                'GEFY61data', 'gefy61_sfy'))
    # Deacy2020 decay data
    ndr.setpath(pf.io.ND_DK_ENDF_KEY(),   os.path.join(nd_base_path,
                'decay2020', 'decay_2020'))
    # Hazards and radiological data from decay2012 files
    ndr.setpath(pf.io.ND_HAZARDS_KEY(),   os.path.join(nd_base_path,
                'decay', 'hazards_2012'))
    ndr.setpath(pf.io.ND_CLEAR_KEY(),     os.path.join(nd_base_path,
                'decay', 'clear_2012'))
    ndr.setpath(pf.io.ND_A2DATA_KEY(),    os.path.join(nd_base_path,
                'decay', 'a2_2012'))
    # attenuation data required for dose rate calculation
    ndr.setpath(pf.io.ND_ABSORP_KEY(),    os.path.join(nd_base_path,
                'decay', 'abs_2012'))
    ndr.load(nd, op=loadfunc)
    return nd

# nuclear data call back
def loadfunc(k, p, index, total):
    print(" [{}/{}]  Reading {}: {}".format(index, total, k, p), end="\r")
    sys.stdout.write("\033[K")
        ...
```

The NuclearDataReader object contains methods to allows the reading of cumulative and spontaneous fission yields. For these to be included, flags need to be set in the NuclearDataReader object (with the relevant included methods) before the 'load' is called.

```
        ...
    # read cumulative fission yield
    ndr.setreadcumfy(True)
    # read spontaneous fission yield
    ndr.setreadsf(True)
        ...
```

5.3 Probing Cross section data

In an analogous fashion to the *extract_xs_endf* methods associated with standard FISPACT-II approach, the FISPACT-II API allows all of the cross sections read by FISPACT-II to be extracted and studied. This can be achieved by requesting the cross sections values for a given nuclide ZAI and MT number. As well as being able to retrieve data with the getter methods, data can be stored with setter methods. This can allow sensitivity of inventory results to changes and individual cross sections to be studied.

It should be noted the FISPACT-II uses internal integer indexing for stored values. As such users need to provide the indexes to the API methods, other methods are provided which can provide a user with

the indexes of interest. In the example function given below the group wise cross sections for a given MT number and a given target ZAI are extracted from the NuclearData object.

Here *groups* is the number of bin in the cross section data, *mt* is the MT number of the desired reaction and *targetzai* is the ZAI number of the target nuclide

5.3.1 Fortran

```
! subroutine which extracts a reaction xs from nuclear data
subroutine extract_xs_from_nucleardata(monitor,nuclear_data,targetzai,mt,xs,groups)
    type(c_ptr), intent(in) :: monitor
    type(c_ptr), intent(in) :: nuclear_data
    integer(ki4), intent(in) :: targetzai
    integer(ki4), intent(in) :: mt
    integer(ki4), intent(in) :: groups
    real(kr8), intent(out) :: xs(0:groups-1)

    integer(ki4) :: nrofzais
    integer(ki4) :: nrofMTs
    integer(ki4) :: nrofbins
    integer(ki4) :: diff
    integer(ki4) :: channelMT
    integer(ki4) :: i, j
    integer(ki4), allocatable :: nd_zais(:)
    real(kr8), allocatable :: xs_data(:)

    ! default values
    xs = 0.0_kr8

    ! number of nucludes with reaction data
    nrofzais = FispactNuclearDataGetNuclideZaiSize(nuclear_data, monitor)
    ! alloacte arrays
    allocate(nd_zais(0:nrofzais-1))
    ! fill array with zai's from reaction data
    call FispactNuclearDataGetNuclideZais(nuclear_data, monitor, &
                                & nrofzais, nd_zais(0:nrofzais-1))

    ! loop over nd zais to find request zai
    do i = 0, nrofzais-1
        if (nd_zais(i) == targetzai) then
            ! get MT's for zai
            nrofMTs = FispactNuclearDataGetReactionNrOfChannels(nuclear_data, &
                        &monitor, i)
            ! for EAF style data
            ! nrofMTs = FispactNuclearDataGetReactionNrOfChannelsEAF(nuclear_data,&
            !                 &monitor, i)
            ! find requested MT
            do j = 0 , nrofMTs-1
                channelMT = FispactNuclearDataGetReactionMT(nuclear_data, &
                            &monitor, i, j)
                ! for EAF style data
                ! channelMT = FispactNuclearDataGetReactionMTEAF(nuclear_data, &
                !                 &monitor, i, j)
```

```
                if ( channelMT == mt) then
                    ! get number of bins with data
                    nrofbins = FispactNuclearDataGetReactionXSNrOfBins(&
                                & nuclear_data,monitor, i, j)
                    ! for EAF style data
                    ! nrofbins = FispactNuclearDataGetReactionXSNrOfBinsEAF(&
                    !               &nuclear_data,monitor, i, j)
                    allocate(xs_data(0:nrofbins-1))
                    ! get xs
                    call FispactNuclearDataGetReactionXS(nuclear_data, monitor, i,&
                                &j, nrofbins, xs_data)
                    ! for EAF style data
                    ! call FispactNuclearDataGetReactionXSEAF(nuclear_data, &
                    !           & monitor, i, j, nrofbins, xs)
                    exit
                end if
            end do
            exit
        end if
    end do

    ! need to account for threshold reactions whcih will only store nonzero data
    if (nrofbins /= groups) then
        diff = groups - nrofbins
        do i = diff, groups-1
            xs(i) = xs_data(i-diff)
        end do
    else
        xs = xs_data
    end if

    ! deallocate arrays
    deallocate(nd_zais)
end subroutine extract_xs_from_nucleardata
```

5.3.2 C

```
//  extacting xs for nuclear data
//
int nrofbins = 0;
double xs[groups];
memset(xs, 0, sizeof(xs));
// number of nuclides with reaction data
int nrofzais = FispactNuclearDataGetNuclideZaiSize(nuclear_data, monitor);
// define arrays
int nd_zais[nrofzais];
// get decay sizes from decay data
FispactNuclearDataGetNuclideZais(nuclear_data, monitor, nrofzais, nd_zais);
for (int i = 0; i < nrofzais+1; ++i) {
    if (nd_zais[i] == targetzai) {
        // get MT's for zai
        int nrofMTs = FispactNuclearDataGetReactionNrOfChannels(nuclear_data,
                    monitor, i);
```

```
            for (int j = 0; j < nrofMTs+1; ++j) {
                // search for requested MT
                int channelMT = FispactNuclearDataGetReactionMT(nuclear_data,
                                monitor, i, j);
                if ( channelMT == mt) {
                    // get number of bins with data
                    nrofbins = FispactNuclearDataGetReactionXSNrOfBins(
                                nuclear_data, monitor, i, j);
                    // get xs data
                    double xs_data[nrofbins];
                    FispactNuclearDataGetReactionXS(nuclear_data, monitor, i,
                                j, nrofbins, xs_data);

                    // account for threshold reactions not having data in all bins
                    if ( nrofbins != groups ) {
                        int diff = groups - nrofbins;
                        for (int k = diff; k < groups+1; ++k) {
                            xs[i] = xs_data[k-diff];
                        }
                    }
                    else {
                        memcpy(xs, xs_data, sizeof(xs));
                    }
                    break;
                }
            }
            break;
        }
    }
```

5.3.3 C++

```
std::vector<double> xs;
    std::fill(xs.begin(), xs.end(), 0.0);

    // get reaction sizes from decay data
    std::vector<int>  nd_zais = nd.getNuclideZais();
    // number of nuclides with reaction data
    int nrofzais = nd_zais.size();
    for (int i = 0; i < nrofzais+1; ++i) {
        if (nd_zais[i] == targetzai) {
            // get MT's for zai
            int nrofMTs = nd.getReactionNrOfChannels(i);
            for (int j = 0; j < nrofMTs+1; ++j) {
                // search for requested MT
                int channelMT = nd.getReactionMT(i, j);
                if ( channelMT == mt) {
                    // get xs data
                    std::vector<double> xs_data = nd.getReactionXS( i, j);
                    // get number of bins with data
                    nrofbins = xs_data.size();
                    // accout for threshold reactions not having data in all bins
```

```
                    if ( nrofbins != groups ) {
                        int diff = groups - nrofbins;
                        for (int k = diff; k < groups+1; ++k) {
                            xs[i] = xs_data[i-diff];
                        }
                    }
                    else{
                        xs.assign(xs_data.begin(), xs_data.end());
                    }
                    break;
                }
            }
            break;
        }
    }
```

5.3.4 Python

```
# extract half life from nd - generic routine
def extract_xs_from_nucleardata(nd,targetzai,mt,groups):
    # get zai of all nuclides with reaction data
    nd_zais = nd.getnuclidezais()
    # get zai index
    zindx = nd_zais.index(targetzai)
    # get number of MTs
    nrofMTs = nd.getreactionnrofchannels(zindx)
    for i in range(0, nrofMTs-1):
        # get MT for channel index
        channelMT = nd.getreactionMT(zindx, i)
        if channelMT == mt:
            # extract xs data
            xs = nd.getreactionxs(zindx, i)
            break
    # acconitng for possible threshold reactions
    if len(xs) != groups:
        diffsize = groups - len(xs)
        xs = [0.0]*diffsize + xs
    return xs
```

5.4 Compressing Cross section data

The loading of the cross section data can require a significant amount of time when performing FISPACT-II calculations. To remove this overhead FISPACT-II allows the cross section data to be compressed into a binary format for swifter reading. The NuclearDataReader can read such binaries, a user simply needs to include the correct key when loading the data (see table 2). The compressed binary can also be produced using API methods, as such is possible to replicate the functionality of the *compress_xs_endf* tool and keyword. Like the *compress_xs_endf* keyword the reading of uncertainties

and co-variances needs to be set before the nuclear data is loaded for them to be included in the binary.

5.4.1 Fortran

```
type(fispact_nuclear_data_index_callback_t), pointer :: writer_state_native_ptr
...

    ! allocate our callback for writer
    allocate(writer_state_native_ptr, stat=status)
    c_state_ptr = c_loc(writer_state_native_ptr)
    lastzai = 0_ki4
    call FispactNuclearDataWriteBinary(nuclear_data, monitor, &
                    &f_c_string(trim("tal19-n.bin")),&
                    & c_state_ptr, c_callback_write)
    deallocate(writer_state_native_ptr)

...
    ! callback for nuclear data reader
    subroutine c_callback_write(c_state) bind(C)
        use, intrinsic :: iso_fortran_env, only : stdout=>output_unit
        type(c_ptr), intent(in), value :: c_state
        type(fispact_nuclear_data_index_callback_t), pointer :: ptr

        call c_f_pointer(c_state, ptr)

        if (ptr%index == 1) then
            write(stdout, '(A)') achar(13)//" writing nuclear "//&
                                &"data to binary"//achar(13)
            flush(stdout)
        end if

        ! simple progress bar - could use a better library for this
        write(stdout, '(A3, I4.1, A1, I4.1, A2, I7, A1)', advance='NO')&
                                & achar(13)//" [",ptr%index, "/" &
                                &, ptr%total, "] ",ptr%zai,""//achar(13)
        flush(stdout)
    end subroutine c_callback_write
```

5.4.2 C

```
// callback for nuclear data write
void write_callback(FISPACT_NUCLEAR_DATA_INDEX_CALLBACK_DATA* data){
    printf("\33[2K\rwriting binary: %i [%i/%i]", data->zai, data->index, data->total );
    fflush(stdout);
}

...
    // compress nulear data to binary
    FISPACT_NUCLEAR_DATA_INDEX_CALLBACK_DATA write_data;
```

```
    FispactNuclearDataWriteBinary(nuclear_data, monitor, "tal19-n.bin",
                                  &write_data, write_callback);
```

5.4.3 C++

```
// nuclear data load callback
void write_callback(int zai, int i, int t){
    std::cout << "\33[2K\r writing binary " << zai << ": " << " [" << i << "/" << t << "]" << std
}
...
    // write nuclear data to binary
    fp::io::ToBinaryFile( nd, monitor, "tal19-n.bin", &write_callback);
```

5.4.4 Python

```
print("Writing binary data...")
# write reading nuclear data to binary
pf.io.to_binary_file(nd, m, "tal19-n.bin", op=writefunc)

# callback function for binary writing
def writefunc(zai, index, total):
    print(" [{}/{}] {}".format(index, total,
        pf.util.nuclide_from_zai(m, zai)), end="\r")
    sys.stdout.write("\033[K")
```

5.5 Probing decay data

In order to study the decay data with standard FISPACT-II a user had to make use of the PRINTLIB keyword or make single nuclide inventories to extract spectra. The FISPACT-II API allows the decay data (branching ratios, decay spectra etc) to studied directly without running a complete FISPACT-II calculation. Much like the cross section data, the decay data can be altered by the user. This could be used to update a given nuclides decay information or perform some sensitivity study.

In the examples below the halflife of a given nuclide is extracted from a FISPACT-II API NuclearData object, here *zai* can be the zai of any nuclide and -1 is returned if the nuclide is not found. These functions can be adapted to extract any relevant data by replacing the methods within the *if* statement.

5.5.1 Fortran

```
    ! function for extracting half life from nuclear data
    function extract_halflife_from_nuc(monitor, nuclear_data, zai) result(halflife)
        type(c_ptr), intent(in) :: monitor
        type(c_ptr), intent(in) :: nuclear_data
```

```
        integer(ki4), intent(in) :: zai

        integer(ki4) :: nrofzais
        integer(ki4) :: i
        real(kr8) :: halflife
        integer(ki4), allocatable :: decay_zais(:)

        ! set default half life value
        halflife = -1.0_kr8
        ! number of nucludes with decay data
        nrofzais = FispactNuclearDataGetDecayDataSize(nuclear_data, monitor)
        ! alloacte arrays
        allocate(decay_zais(0:nrofzais-1))
        ! fill array with zai's from decay data
        call FispactNuclearDataGetDecayZais(nuclear_data, monitor, &
                                    & nrofzais, decay_zais(0:nrofzais-1))
        ! loop over decay zais to find request zai
        do i = 0, nrofzais-1
            if (decay_zais(i) == zai) then
                ! get halflife for found zai
                halflife = FispactNuclearDataGetDecayHalfLife(nuclear_data, monitor, i)
                exit
            end if
        end do
        ! deallocate arrays
        deallocate(decay_zais)
    end function extract_halflife_from_nuc
```

5.5.2 C

```
// function for extracting half life from nuclear data
double extract_halflife_from_nuc(MONITOR_PTR_VALUE monitor,
                                 FISPACT_ND_PTR_VALUE nuclear_data, int zai)
{
    // number of nuclides with decay data
    int nrofzais = FispactNuclearDataGetDecayDataSize(nuclear_data, monitor);
    // define arrays
    int decay_zais[nrofzais];
    // get decay sizes from decay data
    FispactNuclearDataGetDecayZais(nuclear_data, monitor, nrofzais, decay_zais);
    for (int i = 0; i < nrofzais+1; ++i) {
        if (decay_zais[i] == zai) {
            double halflife = FispactNuclearDataGetDecayHalfLife(nuclear_data,
                                                                 monitor, i);
            return halflife;
        }
    }
    return -1.0;
}
```

5.5.3 C++

```
// function for extracting half life from nuclear data
double extract_halflife_from_nuc( const fp::NuclearData& nd, int zai){
    // number of nuclides with decay data
    int nrofzais = nd.getDecayDataSize();
    // get decay sizes from decay data
    std::vector<int> decay_zais = nd.getDecayZais();
    for (int i = 0; i < nrofzais+1; ++i) {
        if (decay_zais[i] == zai) {
            double halflife = nd.getDecayHalfLife(i);
            return halflife;
        }
    }
    return -1.0;
}
```

5.5.4 Python

```
# extract half life from nd - generic routine
def extract_halflife_from_nuc(nd, nuclide_zai):
    halflife = -1
    # get zai of all nuclides with decay data
    decay_zais = nd.getdecayzais()
    for i in range(len(decay_zais)):
        if decay_zais[i] == nuclide_zai:
            # get half life and unc for nuclide if found
            halflife = nd.getdecayhalflife(i)
            break
    return halflife
```

6 Input Data

The InputData object contains the data and flags which define a the precise irradiation and functionality a user wishes when performing an inventory simulation with the FISPACT-II API. For a full calculations these must include the flux spectrum, an initial inventory and the irradiation schedule. Other possible inputs can be whether to include fission, how γ-dose rates should be calculated and what tolerances to use with FISPACT-II's numerical solver.

The InputData object contains a number of setter methods, many of which will be detailed in this section. These are accompanied by getter methods so that set inputs, or defaults, can be probed by the user. A complete overview of all available methods is given in the relevant code reference sheets.

6.1 Input data keys, Conversion factors and Group Structures

To aid with setting the required inputs, the FISPACT-II API includes a series of keys and constants. These will assist the user with inputting the correct data. The relevant integer input data keys are shown in table 5 and the time conversion factors for the irradiation schedule in table 6. The arrays/vectors/lists which contain the bin boundaries of common corr section group structures are presented in table 7.

Table 5: Input Data object keys

Key	**Fortran, C, C++(only dose rate keys for C++)**
Fission yield defaults	FISPACT_INPUT_DATA_INCLUDE_FISYIELD_DEFAULT
Fission yield include	FISPACT_INPUT_DATA_INCLUDE_FISYIELD_LIST
Fission yield exclude	FISPACT_INPUT_DATA_EXCLUDE_FISYIELD_ALL
Fission yield exclude all	FISPACT_INPUT_DATA_EXCLUDE_FISYIELD_LIST
Slab Dose rate	FISPACT_INPUT_DATA_DOSE_TYPE_SLAB
Point Dose rate	FISPACT_INPUT_DATA_DOSE_TYPE_POINT
Quantity	**Python**
Slab Dose rate	pyfisact.DOSE_SLAB()
Point Dose rate	pyfisact.DOSE_POINT()

Table 6: FISPACT-II time conversion factors

Conversion to seconds	**Fortran, C, C++**
Minutes	FISPACT_MIN_TO_SEC
Hour	FISPACT_HOUR_TO_SEC
Day	FISPACT_DAY_TO_SEC
Month	FISPACT_MONTH_TO_SEC
Year	FISPACT_YEAR_TO_SEC
Conversion to seconds	**Python**
Minutes	pyfisact.util.MIN_TO_SEC()
Hour	pyfisact.util.HOUR_TO_SEC()
Day	pyfisact.util.DAY_TO_SEC()
Month	pyfisact.util.MONTH_TO_SEC()
Year	pyfisact.util.YEAR_TO_SEC()

Table 7: Group structure boundaries known to FISPACT-II. These are zero order arrays of length group+1

	Fortran, C	
Groups	**Descending Energy**	**Ascending Energy**
66	FISPACT_GROUP_66	N/A
69	FISPACT_GROUP_69	N/A
100	FISPACT_GROUP_100	N/A
142	FISPACT_GROUP_142	N/A
162	FISPACT_GROUP_162	N/A
172	FISPACT_GROUP_172	N/A
175	FISPACT_GROUP_175	N/A
211	FISPACT_GROUP_211	N/A
315	FISPACT_GROUP_315	N/A
351	FISPACT_GROUP_351	N/A
586	FISPACT_GROUP_586	N/A
616	FISPACT_GROUP_616	N/A
689	FISPACT_GROUP_689	N/A
709	FISPACT_GROUP_709	N/A
1102	FISPACT_GROUP_1102	N/A
	C++	
Groups	**Descending Energy**	**Ascending Energy**
66	fispact::groups::g66()	fispact::groups::G66()
69	fispact::groups::g69()	fispact::groups::G69()
100	fispact::groups::g100()	fispact::groups::G100()
142	fispact::groups::g142()	fispact::groups::G142()
162	fispact::groups::g162()	fispact::groups::G162()
172	fispact::groups::g172()	fispact::groups::G172()
175	fispact::groups::g175()	fispact::groups::G175()
211	fispact::groups::g211()	fispact::groups::G211()
315	fispact::groups::g315()	fispact::groups::G315()
351	fispact::groups::g351()	fispact::groups::G351()
586	fispact::groups::g586()	fispact::groups::G586()
616	fispact::groups::g616()	fispact::groups::G616()
689	fispact::groups::g689()	fispact::groups::G689()
709	fispact::groups::g709()	fispact::groups::G709()
1102	fispact::groups::g1102()	fispact::groups::G1102()
	Python	
Groups	**Descending Energy**	**Ascending Energy**
66	pyfispact.groups.g66()	pyfispact.groups.G66()
69	pyfispact.groups.g69()	pyfispact.groups.G69()
100	pyfispact.groups.g100()	pyfispact.groups.G100()
142	pyfispact.groups.g142()	pyfispact.groups.G142()
162	pyfispact.groups.g162()	pyfispact.groups.G162()
172	pyfispact.groups.g172()	pyfispact.groups.G172()
175	pyfispact.groups.g175()	pyfispact.groups.G175()
211	pyfispact.groups.g211()	pyfispact.groups.G211()
315	pyfispact.groups.g315()	pyfispact.groups.G315()
351	pyfispact.groups.g351()	pyfispact.groups.G351()
586	pyfispact.groups.g586()	pyfispact.groups.G586()
616	pyfispact.groups.g616()	pyfispact.groups.G616()
689	pyfispact.groups.g689()	pyfispact.groups.G689()
709	pyfispact.groups.g709()	pyfispact.groups.G709()
1102	pyfispact.groups.g1102()	pyfispact.groups.G1102()

6.2 Flux Spectra

Unlike the command line version of FISPACT-II the API routines expect the flux in ascending energy groups, this is to match the typical outputs from particle transport codes. The flux spectra group structure must match that of the read in nuclear data and the group bounds must also be provided to the setter methods. The FISPACT-II API includes data which correspond to the group structures used by the nuclear data supplied with FISPACT-II (see table 7). Other flux related quantities can also be set such as a name (if you are performing calculations with multiple spectra) or a wall loading.

6.2.1 Fortran

```
...
    real(kr8) :: flux_values(0:708)
    flux_values = [ ... ]
...
    call FispactInputDataSetFlux(input_data, monitor, 709_ki4, &
                        & FISPACT_GROUP_709(709:0:-1), flux_values(0:708))
    call FispactInputDataSetFluxWallLoading(input_data, monitor, 1.0_kr8)
    call FispactInputDataSetFluxName(input_data, monitor, f_c_string("709 dummy flux"))
...
```

6.2.2 C

```
...
    // set flux - defined in ascending order
    const int group_len = 709;
    double flux_values[709] = { ... }
...
    // reverse the bounds to get ascending order
    double bounds[710];
    memcpy(bounds, FISPACT_GROUP_709, (group_len+1)*sizeof(double));
    reverse_range(bounds, 0, group_len);

    FispactInputDataSetFlux(input_data, monitor, group_len, bounds, flux_values);
    FispactInputDataSetFluxWallLoading(input_data, monitor, 1.0);
    FispactInputDataSetFluxName(input_data, monitor, "709 dummy flux");
```

6.2.3 C++

```
...
        // set flux - defined in ascending order
        std::vector<double> flux_values = { ... }
...
        input.setFlux(fispact::groups::G709(), flux_values);
        input.setFluxWallLoading(1.0);
        input.setFluxName("709 dummy flux");
```

```
...
```

6.2.4 Python

```
flux = [ ... ]
ip.setflux(pf.groups.G709(), flux)
ip.setfluxwallloading(1.0)
ip.setfluxname("709 dummy flux")
```

6.3 Initial Inventory

In the same manner as in a traditional FISPACT-II input file, the initial inventory can be defined either as elements (MASS keyword equivalence) or a nuclides (FUEL keyword equivalence). If the material is set via elements the nuclide make up will be determined by the elemental abundance data available to FISPACT-II, this is loaded during the initialise step discussed earlier. A user can modify the elemental data via a set of APIs which are discussed later (see section 10.2), these can allow the nuclide make up of specific elements to be altered while retaining other nuclides standard natural abundances.

When setting by mass, the total mass (in kg) of material needs to be input. The mass setter method takes two array/vector/list objects: one of atomic numbers and the other of percentages. The atomic number of a given element can be determined via provided methods (see section 10.3). The GetMass methods will return the inventory which has been set.

When using the fuel methods an array of nuclides can be input with the set methods and individual nuclides can be set with append methods. In both cases nuclides are input by their integer ZAI number (utility methods are provided to determine these, see section 10.3) and the number of atoms required. The density (in gcm^{-3}) of the material should also be provided. The nuclides which have been set can be retrieved using the GetFuel methods.

6.3.1 Fortran

6.3.1.1 By elements, MASS

```
    ...
real(kr8) :: percent(0:3)
integer(ki4) :: atomic_numbers(0:3)
    ...
!
! material details
!
! set total mass (kg)
call FispactInputDataSetMassTotal(input_data, monitor, 1.0E-3_kr8)
! set material my weight percentage, MASS keyword
```

```
! atomic numbers for elements to be included, using utilities to convert
! from element names
atomic_numbers = [ FispactGetAtomicNumberFromElementName(monitor, 'Ni'),&
                   &FispactGetAtomicNumberFromElementName(monitor, 'Mn'),&
                   &FispactGetAtomicNumberFromElementName(monitor, 'Fe'),&
                    FispactGetAtomicNumberFromElementName(monitor, 'Cr') ]
! percentge of each element
percent = [ 75.82_kr8, 0.39_kr8, 7.82_kr8, 15.97_kr8 ]
! input materials
call FispactInputDataSetMass(input_data, monitor, size(percent), &
                             & atomic_numbers, percent)
    ...
```

6.3.1.2 By nuclides, FUEL

```
    ...
! set density (g/cm3)
call FispactInputDataSetDensity(input_data, monitor,  19.0_kr8)
! input materials
call FispactInputDataSetFuel(input_data, monitor, 1_ki4, &
       & [FispactGetZai(monitor, f_c_string("U235"))], [1.0E+24_kr8])
    ...
```

6.3.2 C

6.3.2.1 By elements, MASS

```
    ...
// set total mass (kg)
FispactInputDataSetMassTotal(input_data, monitor, 1.0E-3);
// set material my weight percentage, MASS keyword
// atomic numbers for elements to be included, using utilities
// to convert from element names
int atomic_numbers[4] = { FispactGetAtomicNumberFromElementName(monitor, "Ni"),
                  FispactGetAtomicNumberFromElementName(monitor, "Mn"),
                  FispactGetAtomicNumberFromElementName(monitor, "Fe"),
                  FispactGetAtomicNumberFromElementName(monitor, "Cr") };
// percentge of each element
double percent[4] = { 75.82, 0.39, 7.82, 15.97 };
// input materials
FispactInputDataSetMass(input_data, monitor, 4, atomic_numbers, percent);
    ...
```

6.3.2.2 By nuclides, FUEL

```
    ...
// set density (g/cm3)
FispactInputDataSetDensity(input_data, monitor, 19.0);
// input materials
int nuclides[1] = {FispactGetZai(monitor, "U235")};
double atoms[1] = {1.0E+24};
FispactInputDataSetFuel(input_data, monitor, 1, nuclides, atoms);
    ...
```

6.3.3 C++

6.3.3.1 By elements, MASS

```
    ...
//
// set material details
//
// set total mass (kg)
input.setMassTotal(1.0E-3);
// set material my weight percentage, MASS keyword
// atomic numbers for elements to be included, using utilities
// to convert from element names
std::vector<int> atomic_numbers = {
                    fp::util::GetAtomicNumberFromElementName(monitor, "Ni"),
                    fp::util::GetAtomicNumberFromElementName(monitor, "Mn"),
                    fp::util::GetAtomicNumberFromElementName(monitor, "Fe"),
                    fp::util::GetAtomicNumberFromElementName(monitor, "Cr") };
// percentge of each element
std::vector<double>  percent = { 75.82, 0.39, 7.82, 15.97 };
// input materials
input.setMass(atomic_numbers, percent);;
    ...
```

6.3.3.2 By nuclides, FUEL

```
    ...
// set density (g/cm3)
input.setDensity(19.0);
//  set material by nuclides FUEL keyword
// input materials
std::vector<int> nuclides = {  fp::util::GetZai(monitor, "U235") };
std::vector<double> atoms = {1.0E+24};
input.setFuel( nuclides, atoms );
    ...
```

6.3.4 Python

6.3.4.1 By elements, MASS

```
    ...
# set total mass (kg)
ip.setmasstotal(1.0E-3)
# set material my weight percentage, MASS keyword
# atomic numbers for elements to be included, using utilities to convert
# from element names
atomtic_numbers = [ pf.util.z_from_element(m, 'Ni'),
                    pf.util.z_from_element(m, 'Mn'),
                    pf.util.z_from_element(m, 'Fe'),
                    pf.util.z_from_element(m, 'Cr') ]
# percentge of each element
percent = [ 75.82, 0.39, 7.82, 15.97 ]
# input materials
ip.setmass(  atomtic_numbers, percent )
    ...
```

6.3.4.2 By nuclides, FUEL

```
    ...
# set density (g/cm3)
ip.setdensity(19.0)
# set material by nuclides FUEL keyword
#              nuclide                                  atoms
ip.setfuel( [pf.util.zai_from_name(m, "U235")], [1.0E+24])
    ...
```

6.4 Irradiation Schedule

The irradiation schedule needs to be defined by the user. This is defined by a array/vector/list of time steps (in seconds, conversion factors included as part of the API are detailed in table 6) and an array/vector/list of total energy-integrated projectile flux values (in particles s^{-1} cm^{-1}). If pure decay steps are required the integrated projectile flux should be 0 for those times. The complete irradiation schedule can be input vis the setter methods or individual time steps can be input via the append methods.

6.4.1 Fortran

```
...
!
! set irradiation schedule
!
```

```
! 5 min irradiation
call FispactInputDataSetSchedule(input_data, monitor, 1_ki4, &
                                 & [5.0_kr8*FISPACT_MIN_TO_SEC], [1.116E+10_kr8])
! cooling steps
cooltimes = [36.0_kr8, 15.0_kr8, 16.0_kr8, 15.0_kr8, 15.0_kr8, 26.0_kr8, &
            & 33.0_kr8, 36.0_kr8, 53.0_kr8, 66.0_kr8, 66.0_kr8, 97.0_kr8, &
            & 127.0_kr8, 126.0_kr8, 187.0_kr8, 246.0_kr8, 244.0_kr8, &
            & 246.0_kr8, 428.0_kr8, 606.0_kr8, 607.0_kr8 ]
do i=0,size(cooltimes)-1
    call FispactInputDataAppendSchedule(input_data, monitor, cooltimes(i), 0.0_kr8)
end do
...
```

6.4.2 C

```
...
//
// set irradiation schedule
//
// 5 min irradiation
double irradiationtime[1] = {5.0*FISPACT_MIN_TO_SEC};
double fluxamp[1] = {1.116E+10};
FispactInputDataSetSchedule(input_data, monitor, 1, irradiationtime, fluxamp);
// cooling steps
double cooltimes[21] = {36.0, 15.0, 16.0, 15.0, 15.0, 26.0, 33.0, 36.0,
            53.0, 66.0, 66.0, 97.0, 127.0, 126.0, 187.0, 246.0,
            244.0, 246.0, 428.0, 606.0, 607.0 };

for (int i = 0; i < 20; ++i) {
     FispactInputDataAppendSchedule(input_data, monitor, cooltimes[i], 0.0);
}
...
```

6.4.3 C++

```
...
//
// set irradiation schedule
//
// 5 min irradiation
std::vector<double> irradiationtime = {5.0*FISPACT_MIN_TO_SEC};
std::vector<double> fluxamp = {1.116E+10};
input.setSchedule(irradiationtime, fluxamp);
// cooling steps
std::vector<double> cooltimes = {36.0, 15.0, 16.0, 15.0, 15.0, 26.0, 33.0, 36.0,
            53.0, 66.0, 66.0, 97.0, 127.0, 126.0, 187.0, 246.0,
            244.0, 246.0, 428.0, 606.0, 607.0 };

for (int i = 0; i < 20; ++i) {
    input.appendSchedule(cooltimes[i], 0.0);
```

```
}
...
```

6.4.4 Python

```
...
#
# set irradiation schedule
#
# 5 min irradiation
ip.appendschedule(5.0*pf.util.MIN_TO_SEC(), 1.116E+10)
# cooling steps
cooltimes = [36, 15, 16, 15, 15, 26, 33, 36, 53, 66, 66,
             97, 127, 126, 187, 246, 244, 246, 428, 606, 607 ]
for i in cooltimes:
    ip.appendschedule(i, 0.0)
...
```

6.5 Gamma Spectrum and Dose Rates

A user can specify the group structure of the output gamma spectrum (GGBINS keyword) and what type dose rate FISPACT-II will output (DOSE keyword). The gamma spectrum bins should be provided in ascending order in eV. By default the FISPACT-II API will output gamma spectra in the standard FISPACT-II 24-group structure. Also by default, FISPACT-II includes xray lines in the calculation of gamma spectra, these can be excluded via the (ExcludexXrays) methods (this is equivalent to the NOXRAY keyword functionality).

The dose rate type is determined by the integer input keys, see table 5. By default slab dose rates are assumed by FISPACT-II but if a point dose rate is chosen the distance from the point at which the dose rate should be calculated can also be provided. The smallest distance that FISPACT-II can accept is 0.3m, anything less than this will be treated as 0.3m for the purposes of the dose rate calculation.

In the examples in this section it is specified that point dose rates at 1m should be calculated and the output should use a custom gamma group structure.

6.5.1 Fortran

```
...
!
! set gamma dose and spectrum inputs
!
! set dose rate type and options
call  FispactInputDataSetDoseType(input_data, monitor, &
                & FISPACT_INPUT_DATA_DOSE_TYPE_POINT )
call FispactInputDataSetDoseDistance(input_data, monitor, 1.0)
```

```
! set output gamma boundries in eV
gbins = [ ... ]
call FispactInputDataSetGammaEnergyBounds(input_data, monitor, &
                & size(gbins), gbins)
! do not include xrays in gamma spec
call FispactInputDataSetExcludeXrays(input_data, monitor, 1_ki4)
...
```

6.5.2 C

```
...
//
// set gamma dose and spectrum inputs
//
// set dose rate type and options
FispactInputDataSetDoseType(input_data, monitor, &
                & FISPACT_INPUT_DATA_DOSE_TYPE_POINT );
FispactInputDataSetDoseDistance(input_data, monitor, 1.0);
// set output gamma boundries in eV
double gbins[...] = { ... };
FispactInputDataSetGammaEnergyBounds(input_data, monitor, ... , gbins);
// do not include xarys in gamma spec
FispactInputDataSetExcludeXrays(input_data, monitor, 1);
...
```

6.5.3 C++

```
...
//
// set gamma dose and spectrum inputs
//
// set dose rate type and options
input.setDoseType(FISPACT_INPUT_DATA_DOSE_TYPE_POINT);
input.setDosePointDistance(1.0);
// set output gamma boundries in eV
std::vector<double> gbins = { ... };
input.setGammaEnergyBounds(gbins);
input.setExcludeXrays(true);
...
```

6.5.4 Python

```
...
#
# set gamma dose and spectrum inputs
#
```

```
# set dose rate type and options
ip.setdosetype(pf.DOSE_POINT())
ip.setdosepointdistance(1.0)
# set output gamma boundries in eV
gbins = [ ... ]
ip.setgammabounds(gbins)
ip.setexcludexrays(True)
...
```

6.6 Including Fission

In an analogous fashion to the USEFISSION keyword present in traditional FISPACT-II, for fission reactions to be included in a FISPACT-II API driven inventory simulation a flag must be set in the InputData object. Also a user needs to specify which nuclides should undergo fission and which nuclides to exclude. If cumulative and spontaneous fission yields were read in as part of the NuclearDataReader they will be included here. In the examples fission will be set to on and only ^{235}U can fission.

6.6.1 Fortran

The interpretation of the fission yield list is determined by which option is set. The option keys are shown in table 5.

```
...
!
! set fission inputs
!
! use fission
call FispactInputDataSetEnableUseFission(input_data, monitor, 1_ki4)
! set which nuclides to use for fission
! set fission include option and nuclides
call FispactInputDataSetFissionYieldOption(input_data, monitor, &
        & FISPACT_INPUT_DATA_INCLUDE_FISYIELD_LIST)
call FispactInputDataSetFissionYieldList(input_data, monitor, &
        & 1_ki4, [ FispactGetZai(monitor, f_c_string("U235")) ] )
...
```

6.6.2 C

The interpretation of the fission yield list is determined by which option is set. The option keys are shown in table 5.

```
...
//
// set fission inputs
//
```

```
// use fission
FispactInputDataSetEnableUseFission(input_data, monitor, 1);
// set which nuclides to use for fission
// set fission include option and nuclides
FispactInputDataSetFissionYieldOption(input_data, monitor, &
                & FISPACT_INPUT_DATA_INCLUDE_FISYIELD_LIST);
FispactInputDataSetFissionYieldList(input_data, monitor, &
                & 1, {FispactGetZai(monitor, "U235")} );
//
...
```

6.6.3 C++

A user must use the correct method to include or exclude nuclide from fission calculations.

```
...
std::vector<int> nuclides = {  fp::util::GetZai(monitor, "U235") };
...
//
// set fission inputs
//
// use fission
input.setUseFission(true);
// set which nuclides to use for fission
input.setFissionYieldIncludes( nuclides  );
...
```

6.6.4 Python

A user must use the correct method to include or exclude nuclide from fission calculations.

```
...
#
# set fission inputs
#
# use fission
ip.setusefission(True)
# set which nuclides to use for fission
ip.setfissionyieldincludes( [pf.util.zai_from_name(m, 'U235')]  )
...
```

6.7 Tolerances

Which nuclides, and how many atoms thereof, appear in an inventory is a result of the numerical solutions which FISPACT-II determines and the API includes methods which can adjust the required

parameters to suit a users needs. These methods are equivalent to those used in the input file driven version of FISPACT-II via the TOLERANCE and MIND keywords.

In the API the AtomsThreshold methods set the minimum number of atoms which must be present for a nuclide to be present in the OutputData object. The SolverTolerence methods allow a user to control the relative and absolute tolerance of the LSODES solver FISPACT-II uses. If neither of these methods are included the FISPACT-II defaults will be used.

6.7.1 Fortran

```
...
! set atoms threshold, MIND keyword
call FispactInputDataSetAtomsThreshold(input_data, monitor, 1.0e3_kr8)
! set solver tolerance:  relative, absolute
call FispactInputDataSetSolverTolerance(input_data, monitor, 2.0E-3,  1.0E2 )
...
```

6.7.2 C

```
...
// set atoms threshold, MIND keyword
FispactInputDataSetAtomsThreshold(input_data, monitor, 1.0e3);
// set solver tolerance:  relative, absolute
FispactInputDataSetSolverTolerance(input_data, monitor, 2.0E-3,  1.0E2 );
...
```

6.7.3 C++

```
...
// set atoms threshold, MIND keyword
input.setAtomsThreshold(1.0e3);
// set solver tolerance:  relative, absolute
input.setSolverTolerance( 2.0E-3,  1.0E2 );
...
```

6.7.4 Python

```
...
# set atoms threshold, MIND keyword
ip.setatomsthreshold(1.0e3)
# set solver tolerance:  relative, absolute
ip.setsolvertolerance( 2.0E-3,  1.0E2 )
...
```

7 Compute Inventory

Once the Nuclear Data and Input Data have been set the FISPACT-II engine can be called and the inventory evolved according to the irradiation schedule defined. This is done via the *fispactprocess* methods which takes in the Nuclear Data and Input Data in and fills the Output Data object with the evolved inventory. The engine called here collapses the cross sections with the provided flux spectrum and solves the rate equations to evolve the inventory. Like the NuclearDataReader the *fispactprocess* methods can take a call back routine as an argument so that progress of the solver can be tracked.

7.1 Fortran

```
...
    ! run fispact
    allocate(process_state_native_ptr, stat=status)
    c_state_ptr = c_loc(process_state_native_ptr)
    call FispactProcess(input_data, nuclear_data, output_data, monitor, &
                    & c_state_ptr, c_callback_process)
    deallocate(process_state_native_ptr)
...
        ! callback for process
        subroutine c_callback_process(c_state) bind(C)
            type(c_ptr), intent(in), value :: c_state

            type(fispact_process_callback_t), pointer :: ptr

            call c_f_pointer(c_state, ptr)
            write(*, '(A2, I2.1, A1, I2.1, A1, A, A, A2)') &
                    & " [", ptr%index, "/", ptr%total, "]", " processing ",
                    & trim(c_f_string(ptr%process, FISPACT_MAX_PATH_LENGTH))

        end subroutine c_callback_process
...
```

7.2 C

```
...
void process_callback(FISPACT_COMPUTE_CALLBACK_DATA* data){
    printf("\33[2K\r%s: [%i/%i]", data->process, data->index, data->total );
    fflush(stdout);
}
...
    FISPACT_COMPUTE_CALLBACK_DATA compute_data;
    FispactProcess(input_data, nuclear_data, output_data, monitor,

...
```

7.3 C++

```
...
void process_callback(std::string process_name, int i, int t){
    std::cout << "\33[2K\r [" << i << "/" << t << "] " << process_name << std::flush;
}
...
fp::Process(input, nd, output, monitor, process_callback);
...
```

7.4 Python

```
...
# callback for compute
def computefunc(p, index, total):
    print(" [{}/{}]  processing {}".format(index, total, p))
...
pf.process(i, nd, o, m, op=computefunc)
...
```

8 Output Data

The Output data object contains the complete FISPACT-II nuclide inventories as populated during an irradiation calculation. This is equivalent to the data contained within the traditional FISPACT-II output file. A significant advantage of using the API over the traditional approach is the user has complete control over which outputs are printed. While the API calculates all output data the user can specify what is extracted from the output data and save that using standard file writing methods. A complete inventory may be saved in a JSON output if desired.

8.1 Output data structure and keys

8.1.1 Output Data structure

The Output data is stored in a dictionary like struct with at each time step containing a list of all nuclides present, with there associated radiological properties, and a set of totals defined by the nuclide content at that time step. The structure is described in below.

```
{
  "type": "object",
  "required": [],
  "properties": {
    "run_data": {
```

```
    "type": "object",
    "required": [],
    "properties": {
      "timestamp": {
        "type": "string",
         "unit": ""
      },
      "run_name": {
        "type": "string",
         "unit": ""
      },
      "flux_name": {
        "type": "string",
         "unit": ""
      }
    }
  },
  "inventory_data": {
    "type": "array",
    "items": {
      "type": "object",
      "required": [],
      "properties": {
        "irradiation_time": {
          "type": "number",
          "unit": "s"
        },
        "cooling_time": {
          "type": "number",
          "unit": "s"
        },
        "flux": {
          "type": "number",
          "unit": "cm-2 s-1"
        },
        "total_activity": {
          "type": "number",
          "unit": "Bq"
        },
        "total_mass": {
          "type": "number",
          "unit": "kq"
        },
        "total_heat": {
          "type": "number",
          "unit": "kW"
        },
        "alpha_heat": {
          "type": "number",
          "unit": "kW"
        },
        "beta_heat": {
          "type": "number",
          "unit": "kW"
        },
        "gamma_heat": {
```

```
      "type": "number",
      "unit": "kW"
    },
    "ingestion_dose": {
      "type": "number",
      "unit": "Sv kg-1"
    },
    "inhalation_dose": {
      "type": "number",
      "unit": "Sv kg-1"
    },
    "dose_rate": {
      "type": "object",
      "required": [],
      "properties": {
        "type": {
          "type": "string",
          "unit": ""
        },
        "distance": {
          "type": "number",
          "unit": "m"
        },
        "mass": {
          "type": "number",
          "unit": "kg"
        },
        "dose": {
          "type": "number",
          "unit": "Sv hr-1"
        }
      }
    },
    "nuclides": {
      "type": "array",
      "items": {
        "type": "object",
        "required": [],
        "properties": {
          "element": {
            "type": "string",
            "unit": ""
          },
          "isotope": {
            "type": "number",
            "unit": ""
          },
          "state": {
            "type": "string",
            "unit": ""
          },
          "half_life": {
            "type": "number",
            "unit": "s"
          },
          "zai": {
```

```
            "type": "number",
            "unit": ""
          },
          "atoms": {
            "type": "number",
            "unit": ""
          },
          "grams": {
            "type": "number",
            "unit": "g"
          },
          "activity": {
            "type": "number",
            "unit": "Bq"
          },
          "heat": {
            "type": "number",
            "unit": "kW"
          },
          "alpha_heat": {
            "type": "number",
            "unit": "kW"
          },
          "beta_heat": {
            "type": "number",
            "unit": "kW"
          },
          "gamma_heat": {
            "type": "number",
            "unit": "kW"
          },
          "dose": {
            "type": "number",
            "unit": "Sv hr-1"
          },
          "ingestion": {
            "type": "number",
            "unit": "Sv"
          },
          "inhalation": {
            "type": "number",
            "unit": "Sv"
          }
        }
      }
    },
    "gamma_spectrum": {
      "type": "object",
      "required": [],
      "properties": {
        "boundaries": {
          "type": "array",
          "items": {
            "type": "number",
            "unit": "MeV"
          }
```

```
            },
            "values": {
              "type": "array",
              "items": {
                "type": "number",
                "unit": "MeV s-1"
              }
            }
          }
        }
      }
    }
  }
}
```

8.1.2 Output Data keys

The methods which probe the Output data object are generic and as such make use of a number of integer keys to determine which values are desired. These integer keys are global parameters within the FISPACT-II API and are detailed in table 8.1.2.

Table 8: Output Data object keys

Quantity	Fortran, C, C++
Irradiation Time	FISPACT_OUTPUT_DATA_INVENTORY_IRRAD_TIME
Cooling Time	FISPACT_OUTPUT_DATA_INVENTORY_COOL_TIME
Total Activity	FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_ACTIVITY
Alpha Activity	FISPACT_OUTPUT_DATA_INVENTORY_ALPHA_ACTIVITY
Beta Activity	FISPACT_OUTPUT_DATA_INVENTORY_BETA_ACTIVITY
Gamma Activity	FISPACT_OUTPUT_DATA_INVENTORY_GAMMA_ACTIVITY
Total Heat	FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_HEAT
Alpha Heat	FISPACT_OUTPUT_DATA_INVENTORY_ALPHA_HEAT
Beta Heat	FISPACT_OUTPUT_DATA_INVENTORY_BETA_HEAT
Gamma Heat	FISPACT_OUTPUT_DATA_INVENTORY_GAMMA_HEAT
Total Mass	FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_MASS
Total Atoms	FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_ATOMS
Ingestion	FISPACT_OUTPUT_DATA_INVENTORY_INGESTION
Inhalation	FISPACT_OUTPUT_DATA_INVENTORY_INHALATION
Flux Amplitude	FISPACT_OUTPUT_DATA_INVENTORY_FLUX_AMP
Quantity	**Python**
Irradiation Time	pyfisact.INVENTORY_IRRAD_TIME()
Cooling Time	pyfisact.INVENTORY_COOL_TIME()
Total Activity	pyfisact.INVENTORY_TOTAL_ACTIVITY()
Alpha Activity	pyfisact.INVENTORY_ALPHA_ACTIVITY()
Beta Activity	pyfisact.INVENTORY_BETA_ACTIVITY()
Gamma Activity	pyfisact.INVENTORY_GAMMA_ACTIVITY()
Total Heat	pyfisact.INVENTORY_TOTAL_HEAT()
Alpha Heat	pyfisact.INVENTORY_ALPHA_HEAT()
Beta Heat	pyfisact.INVENTORY_BETA_HEAT()
Gamma Heat	pyfisact.INVENTORY_GAMMA_HEAT()
Total Mass	pyfisact.INVENTORY_TOTAL_MASS()
Total Atoms	pyfisact.INVENTORY_TOTAL_ATOMS()
Ingestion	pyfisact.INVENTORY_INGESTION()
Inhalation	pyfisact.INVENTORY_INHALATION()
Flux Amplitude	pyfisact.INVENTORY_FLUX_AMP()

8.2 Probing the inventory data

When probing the FISPACT-II API output data object which irradiation time step is to be probed needs to be decided. This is refereed to as a the inventory index. The number of available indices is determined by the input irradiation schedule but output data methods are provided to retrieve this information from the Output data. A user can specify a given inventory index or iterate over all indices to extract the data at all time steps. Once the desired inventory index has been determined methods which make use of the integer inventory keys can be implemented to retrieve the requested data.

8.2.1 Inventory values

At each time step total inventory quantities can be extracted. In the examples below the total time elapsed and the total inventory decay heating are printed to screen.

8.2.1.1 Fortran

```
    ...
real(kr8) :: coolingtime, irradtime
real(kr8) :: time, mass
real(kr8) :: total_heat
    ...
! print heating after irradiation
write(6,*) ""
write(6,*) " TIME(yrs) ", "  HEAT(kW/kg)"
time = 0.0_kr8
do i = 0, FispactOutputDataGetNrOfInventoryEntries(output_data, monitor)-1
    ! timesteps where flux amp = 0
    coolingtime = FispactOutputDataGetInventoryValueByKey(output_data, monitor, &
                            & i, FISPACT_OUTPUT_DATA_INVENTORY_COOL_TIME)
    ! timesteps where flux amp /= 0
    irradtime = FispactOutputDataGetInventoryValueByKey(output_data, monitor, &
                            & i, FISPACT_OUTPUT_DATA_INVENTORY_IRRAD_TIME)
    ! do not print initial inventory heating
    if (coolingtime > 0.0_kr8 .or. irradtime > 0.0_kr8 ) then
        time = time + coolingtime
        ! get total heating from inventory
        total_heat = FispactOutputDataGetInventoryValueByKey(output_data, monitor, &
                            & i, FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_HEAT)
        ! get total mass from inventory
        mass = FispactOutputDataGetInventoryValueByKey(output_data, monitor, &
                            & i, FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_MASS)
        write(6,"(2E12.4)") time/FISPACT_YEAR_TO_SEC, total_heat/mass
    end if
end do
    ...
```

8.2.1.2 C

```
...
// heating after irradiation
printf("\n\n");
printf( "  TIME(yrs)    HEAT(kW/kg) \n" );
double time = 0.0;
int nrofentries = FispactOutputDataGetNrOfInventoryEntries(output_data, monitor);
for (int i = 0; i < nrofentries+1; ++i) {
    // timesteps where flux amp = 0
    double coolingtime = FispactOutputDataGetInventoryValueByKey(output_data,
                    monitor, i, FISPACT_OUTPUT_DATA_INVENTORY_COOL_TIME);
    // timesteps where flux amp /= 0
```

```
        double irradtime = FispactOutputDataGetInventoryValueByKey(output_data,
                           monitor, i, FISPACT_OUTPUT_DATA_INVENTORY_IRRAD_TIME);
        // do not print initial inventory heating
        if ( (coolingtime > 0.0) || (irradtime > 0.0) ) {
            time += coolingtime/FISPACT_YEAR_TO_SEC;
            // get total heta from inventory
            double total_heat = FispactOutputDataGetInventoryValueByKey(output_data,
                    monitor, i, FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_HEAT);
            // get total mass from inventory
            double mass = FispactOutputDataGetInventoryValueByKey(output_data, monitor,
                    i, FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_MASS);
            double heatpermass = total_heat/mass;
            printf("%12.4E %12.4E \n", time, heatpermass);
        }
    }
    ...
```

8.2.1.3 C++

```
    ...
    // heating after irradiation
    double time = 0.0;
    std::cout <<  "  \n\n";
    std::cout <<  std::setw(15) << "TIME(yrs)" << std::setw(16) << "HEAT(kW/kg)\n";
    for(int i=0; i< output.getNrOfInventoryEntries(); ++i){
        // timesteps where flux amp = 0
        double coolingtime = output.getInventoryValue( i,
                                        FISPACT_OUTPUT_DATA_INVENTORY_COOL_TIME);
        // timesteps where flux amp /= 0
        double irradtime = output.getInventoryValue(i,
                                        FISPACT_OUTPUT_DATA_INVENTORY_IRRAD_TIME);
        // do not print initial inventory heating
        if ( (coolingtime > 0.0) || (irradtime > 0.0) ) {
            time += coolingtime/FISPACT_YEAR_TO_SEC;
            // get total heta from inventory
            double total_heat = output.getInventoryValue( i,
                                        FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_HEAT);
            // get total mass from inventory
            double mass = output.getInventoryValue(i,
                                            FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_MASS);
            std::cout << std::setw(15) << time << std::setw(15)
                      << total_heat/mass << "\n";
        }
    }
    ...
```

8.2.1.4 Python

```
    ...
# extract heat from output
```

```
def print_heat_after_irrad(o):
    # start at 2nd index to avoid initial data input
    time = 0
    print(" ")
    print(" TIME(yrs) ", "  HEAT(kW/kg)")
    for i in range(o.getnrofinventoryentries() ):
        # get cooling time
        coolingtime = o.getinventoryvalue(i, pf.INVENTORY_COOL_TIME())
        irradtime = o.getinventoryvalue(i, pf.INVENTORY_IRRAD_TIME())
        if coolingtime > 0.0 or irradtime > 0.0:
            ## sum time
            time += coolingtime
            # get total heating from inventory
            total_heat = o.getinventoryvalue(i, pf.INVENTORY_TOTAL_HEAT())
            # get total mass from inventory
            mass = o.getinventoryvalue(i, pf.INVENTORY_TOTAL_MASS())
            print(" {:.4E}   {:.4E} ".format(time/pf.util.YEAR_TO_SEC(), total_heat/mass) )
    ...
```

8.2.2 Nuclide information

Often information on individual nuclide, or nuclides, is required and such information is extractable form the API OutputData object. Much like when probing the inventory totals, the time step at which the nuclide information is desired need to be provided. Methods are included to check if a given nuclide is present at a given time step so that the nuclide list only needs to be searched if necessary. Once the presence of a given nuclide has been determined its index in the nuclide list (at the chosen time step) needs to be found, methods are provided for this as well as the size of the nuclide list. From this the nuclide data struct for a given nuclide can be pointed to and the relevant data extracted. It should be noted that the data here is not per unit mass and a user will have to perform a manual conversion if this is desired.

In the examples a function is shown which will find the activity for a given nuclide at a given inventory index. These functions are generic and could be simply modified to output any quantity in the output nuclide data struct.

8.2.2.1 Fortran

A user must allocate a pointer which will populate the nuclide data type. This prevents data being duplicated, causing an over use of memory.

```
    ...
    ! extract nuclide activity at a given timestep
    function extract_activity(output_data, monitor, inv_index, zai) result(activity)
        type(c_ptr), intent(in) :: monitor
        type(c_ptr), intent(in) :: output_data
        integer(ki4), intent(in) :: inv_index, zai
        type(c_ptr) :: c_nuclide_ptr
```

```
        type(fispact_inventory_nuclide_data_t), pointer :: nuclide_native_ptr
        real(kr8) :: activity
        integer(ki4) :: status, check, nuc_index

        allocate(nuclide_native_ptr, stat=status)
        c_nuclide_ptr = c_loc(nuclide_native_ptr)
        ! set default output
        activity = -1.0_kr8
        ! check if nuclide is in inventory
        check = FispactOutputDataFindInventoryNuclideExists(output_data, monitor,&
            & inv_index, zai)
        if (check == 1_ki4) then
            ! if nuclide is found, find activity
            nuc_index = FispactOutputDataFindInventoryNuclideIndex(output_data,&
                & monitor, inv_index, zai)
            call FispactOutputDataGetInventoryNuclideAtIndex(output_data, monitor,&
            & inv_index, nuc_index, c_nuclide_ptr)
            ! if nuclide is found, output activity
            activity = nuclide_native_ptr%activity
        end if

        deallocate(nuclide_native_ptr)
    end function extract_activity_from_nuc
    ...
```

8.2.2.2 C

```
    ...
// function for extracting activity from output data nuclide
double extract_activity_from_nuc(FISPACT_OUTPUT_PTR_VALUE output_data,
                                 MONITOR_PTR_VALUE monitor,
                                 int inv_index,
                                 int zai)
{
    // check if nuclide is in inventory
    int check = FispactOutputDataFindInventoryNuclideExists(output_data, monitor,
                                                            inv_index, zai);
    if (check == 1) {
        // if nuclide is found, find activity
        int nuc_index = FispactOutputDataFindInventoryNuclideIndex(output_data,
                                    monitor, inv_index, zai);
        FISPACT_OUTPUT_DATA_INVENTORY_NUCLIDE nuclideobj;
        FispactOutputDataGetInventoryNuclideAtIndex(output_data, monitor,
                    inv_index, nuc_index, &nuclideobj);
        // if nuclide is found, output activity
        double activity = nuclideobj.activity;
        return activity;
    }
    return -1.0;
}
    ...
```

8.2.2.3 C++

```
    ...
//  extract activity - generic routine
double extract_activity_from_nuc(fp::OutputData& output,
                                 int inv_index,
                                 int nuclide_zai) {
    // check if nuclide present at a given step
    if (output.findInventoryExists(inv_index, nuclide_zai) ) {
        // get index of nuclide in inventory nuclide list
        int nuclide_index = output.findInventoryIndex(inv_index, nuclide_zai);
        // get nuclide list at given step
        std::vector<fp::OutputNuclideData> nuclide_list =
                                    output.getInventoryNuclides(inv_index);

        // get heating value for that nuclide from inventory
        double activity = nuclide_list[nuclide_index].activity;
        return activity;
    }
    return -1.0;
}
    ...
```

8.2.2.4 Python

```
    ...
# extract activity - generic routine
def extract_activity_from_nuc(o, inv_index, nuclide_zai):
    activity = -1.0
    # check if nuclide present at a given step
    if (o.findinventoryexists(inv_index, nuclide_zai)):
        # get index of nuclide in inventory nuclide list
        nuclide_index = o.findinventoryindex(inv_index, nuclide_zai)
        # get nuclide list at given step
        nuclide_list = o.getinventorynuclides(inv_index)
        # get heating value for that nuclide from inventory
        activity = nuclide_list[nuclide_index].activity
    return activity
    ...
```

8.2.3 Dominant Contributions

Output data methods which sort, in ascending order, the nuclides in the inventory according to to there contribution to a given inventory total quantity are available. The inventory quantity is determined by a given key, see table 8.1.2. For example, requesting a sorted inventory according to total activity would return a the nuclides in that inventory in ascending order by nuclide activity.

In the examples below the 20 nuclides which produce the greatest decay heating are output for the first inventory index. Also output in these examples are the nuclide in questions half-life's extracted from the NuclearData object using the approaches shown in section 5.5.

8.2.3.1 Fortran

```
...
character(FISPACT_NUCLIDE_NAME_LENGTH) :: nuclidename
real(kr8) :: heating, mass
real(kr8), allocatable :: dom_heat(:)
integer(ki4), allocatable :: dom_zai(:)
...
! print dominatants heats and half-life after irradiation
write(6,*) ""
write(6,*) " DOMINANT NUCLIDES AFTER IRRADIATION"
write(6,*) " HALFLIFE(yrs) ", "HEAT(kW/kg)", " NUCLIDE"
! mass at this time step
mass = FispactOutputDataGetInventoryValueByKey(output_data, monitor, &
                        & 1_ki4, FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_MASS)
! number of nuclides in inventory at time step
nrofnuclides = FispactOutputDataGetInventoryNrOfNuclides(output_data, &
                                                        & monitor, 1_ki4)
! allocate arrays for dominant zais  and avalues
allocate(  dom_heat(0:nrofnuclides-1), dom_zai(0:nrofnuclides-1)    )
! extract dominants
call FispactOutputDataGetInventorySortedByKey(output_data, monitor, 1_ki4, &
                                    & FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_HEAT, &
                                    &nrofnuclides, dom_zai, dom_heat )
do i = nrofnuclides-21, nrofnuclides-1, 1
    heating = dom_heat(i) / mass
    halflife = extract_halflife_from_nuc(monitor, nuclear_data, dom_zai(i))
    if (halflife /= -1.0_kr8) then
        call FispactGetNuclideName(monitor, dom_zai(i), nuclidename)
        write(6,"(E12.4, 2x,E12.4, 4x, 6A)") halflife/FISPACT_YEAR_TO_SEC, &
                    & heating, nuclidename
    end if
end do
...
```

8.2.3.2 C

```
...
// print dominants heats and halflife after irradiation
printf("\n\n");
printf("  DOMINANT NUCLIDES AFTER IRRADIATION\n" );
printf( "  TIME(yrs)     HEAT(kW/kg) NUCLIDE\n" );
//  mass at this timestep
double mass = FispactOutputDataGetInventoryValueByKey(output_data, monitor,
                    1, FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_MASS);
// number of nuclides at his timestep
```

```
int nrofnuclides = FispactOutputDataGetInventoryNrOfNuclides(output_data,
                                                             monitor, 1);
// define arrays
double dom_heat[nrofnuclides];
int dom_zai[nrofnuclides];
// extract dominants
FispactOutputDataGetInventorySortedByKey(output_data, monitor, 1,
                                FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_HEAT,
                                nrofnuclides, dom_zai, dom_heat );

for (int i = (nrofnuclides-21); i < (nrofnuclides); ++i) {
    double heating = dom_heat[i] / mass;
    double halflife = extract_halflife_from_nuc(monitor, nuclear_data, dom_zai[i]);
    if (halflife != -1.0) {
        char nuclidename[FISPACT_NUCLIDE_NAME_LENGTH];
        FispactGetNuclideName(monitor, dom_zai[i], nuclidename);
        printf( "%12.4E %12.4E %6s \n", halflife/FISPACT_YEAR_TO_SEC,
                heating, nuclidename);
    }
}
...
```

8.2.3.3 C++

```
...
// print dominants heats and halflife after irradiation
std::cout <<"\n\n";
std::cout <<"  DOMINANT NUCLIDES AFTER IRRADIATION\n";
std::cout << std::setw(15) << "TIME(yrs)" << std::setw(15) << "HEAT(kW/kg)"
          << std::setw(16) <<  "NUCLIDE\n";
//  mass at this timestep
double mass = output.getInventoryValue(1,
                            FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_MASS);
// sort inventory by total heat
std::pair<std::vector<int>, std::vector<double>>  dom_sort =
        output.getSortedInventory(1, FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_HEAT);
std::vector<int> dom_zai = std::get<0>(dom_sort);
std::vector<double> dom_heat = std::get<1>(dom_sort);
// number of nuclides at his timestep
int nrofnuclides = dom_zai.size();
for (int i = (nrofnuclides-21); i < nrofnuclides; ++i) {
    double heating = dom_heat[i] / mass;
    double halflife = extract_halflife_from_nuc(nd, dom_zai[i]);
    if (halflife != -1.0) {
        std::string nuclidename = fp::util::GetNuclideName(monitor, dom_zai[i] );
        std::cout << std::setw(15) << halflife/FISPACT_YEAR_TO_SEC
                  << std::setw(15) << heating
                  << std::setw(15) << nuclidename << "\n";
    }
}
...
```

8.2.3.4 Python

```
    ...
# extract halflife and heating of nuclides after irradiation
def print_halflife_and_heat(nd, o):
    print(" ")
    print(" DOMINANT NUCLIDES AFTER IRRADIATION")
    print(" HALFLIFE(yrs) ", " HEAT(kW/kg)", " NUCLIDE")
    # get the nuclide list and mass for the first time step
    mass = o.getinventoryvalue(1, pf.INVENTORY_TOTAL_MASS())
    # sort the inventory by decay heat so the dominants can be found
    dom_zai, dom_heat = o.getsortedinventory(1, pf.INVENTORY_TOTAL_HEAT())
    # extract the 20 nuclides with the greatest total heating
    nrofnuclides = len(dom_zai)
    for i in range(nrofnuclides-21, nrofnuclides, 1):
        # call routine which extract nulide heat
        heating = dom_heat[i] / mass
        # call routine which extract nulide half life
        half = extract_halflife_from_nuc(nd, dom_zai[i])
        if half != -1:
            print( " {:.4E}      {:.4E}    {} ".format(
                        half/pf.util.YEAR_TO_SEC(), heating,
                        pf.util.nuclide_from_zai(m, dom_zai[i])) )
    ...
```

8.2.4 Gamma Spectra

The Output data will contain a binned gamma (photon) spectra for each time step. By default this is stored in the standard FISPACT-II 24-group structure but will be in user defined groups if they were set in the input. There are bespoke APIs for extracting the gamma spectrum and its bounds from the output data. The bounds have size 1 greater than the bin values.

8.2.4.1 Fortran

```
    ...
    ! print gamma spec
    write(6,*)""
    write(6,*)" GAMMA SPECTRUM "
    write(6,"(3(A13,x))") " LOWER En", " UPPER En ", " Spec "
    ! get the nr of spectrum boundries, same at all times
    nrofbins = FispactOutputDataGetInventoryGammaSpectrumNrOfBins(output_data, &
                            & monitor, 0)
    allocate( boundaries(0:nrofbins-1), spec(0:nrofbins-2))
    ! get the spectrum values at the required time steps
    call FispactOutputDataGetInventoryGammaSpectrum(output_data, monitor, &
                                & inv_index, nrofbins, boundaries, spec)

    do i = 0, size(boundaries)-1
        write(6,"(3(E12.4,2x))") boundaries(i), boundaries(i+1), spec(i)
```

```
    end do
    deallocate( boundaries, spec )
    ...
```

8.2.4.2 C

```
    ...
    // print gamma spec
    printf("\n\n");
    printf(" GAMMA SPECTRUM \n");
    printf( "  LOWER En    UPPER En    Spec \n");
    // get the nr of spectrum boundries, same at all times
    int nrofbins = FispactOutputDataGetInventoryGammaSpectrumNrOfBins(output_data,
                                                                      monitor, 0);
    double boundaries[nrofbins];
    double spec[nrofbins-1];
    // get the spectrum values at the required time steps
    FispactOutputDataGetInventoryGammaSpectrum(output_data, monitor,
                                inv_index, nrofbins, boundaries, spec);
    for (int i = 0; i < nrofbins; ++i) {
        printf("%12.4E %12.4E %12.4E\n", boundaries[i], boundaries[i+1],

    }
    ...
```

8.2.4.3 C++

```
    ...
    // print gamma spec
    std::cout <<  "  \n\n";
    std::cout <<  " GAMMA SPECTRUM\n";
    std::cout <<  std::setw(15) << "LOWER En" << std::setw(15) << "UPPER En" <<
                  std::setw(15) << "Spec" << "\n";
    // get the spectrum boundries, same at all times
    std::vector<double> bounds = output.getGammaSpectrumBoundaries(0);
    // get the spectrum values at the required time steps
    std::vector<double> spec = output.getGammaSpectrumBins(inv_index);

    for(int i=0; i< bounds.size()-1; ++i){
    std::cout <<  std::setw(15) << bounds[i] << std::setw(15) << bounds[i+1] <<
                  std::setw(15) << spec[i] << "\n";
    }
    ...
```

8.2.4.4 Python

```
...
# get the spectrum boundries, same at all times
bounds = o.getgammaspectrumboundaries(inv_index)
# get the spectrum values at the required time steps
spec = o.getgammaspectrumbin(inv_index)
for i in range(len(bounds)-1):
    print( " {:.4E}   {:.4E}   {:.4E} ".format(bounds[i],
                        bounds[i+1], spec[i]) )
...
```

8.2.5 Inventory Gamma Dose Rates

Much like the gamma spectrum, the gamma dose rates from the complete inventory have set of API's to access them at each time step. It should be noted by the user that dose rates are inherently related to geometry which FISPACT-II has no knowledge of. As such the dose rates calculated are characteristic approximation of the true dose rate. It should also be noted that FISPACT-II relies on the gamma attenuation data (the absorp file key) to calculate dose rates, if this data is not provided the dose rates are not calculated.

The dose rate object has entries for all of the relevant dose rate data: the type, the mass, the distance and the dose rate itself (in Sieverts/hour). Which of these are populated is dependant on the dose rate inputs set before the calculation: the distance and mass are only relevant to point dose rates and which dose type has been requested is stored in the dose type entry. The mass here is the mass of material used to define the point source dose rate in kg, the dose rate itself is for 1g of material at the specified distance. If a value in the dose rate object take a default value of 0 when not calculated or relevant.

The methods which populate a dose rate object take in pointers. This is to avoid excessive replication of inventory data and ease memory requirements. The example below extract the dose rate at each inventory time step.

8.2.5.1 Fortran

```
...
! dose rate type
type(dose_rate_data_t), pointer :: doseobj_ptr
type(c_ptr) :: c_dose_ptr
...
! print dose rate after irradiation
write(6,*) ""
write(6,*) " TIME(yrs) ", "  Dose(Sr/hr)"
time = 0.0_kr8
do i = 0, FispactOutputDataGetNrOfInventoryEntries(output_data, monitor)-1
    ! timesteps where flux amp = 0
    coolingtime = FispactOutputDataGetInventoryValueByKey(output_data, monitor, &
                            & i, FISPACT_OUTPUT_DATA_INVENTORY_COOL_TIME)
```

```
        ! timesteps where flux amp /= 0
        irradtime = FispactOutputDataGetInventoryValueByKey(output_data, monitor, &
                                    & i, FISPACT_OUTPUT_DATA_INVENTORY_IRRAD_TIME)
        ! do not print initial inventory dose rate
        if (coolingtime > 0.0_kr8 .or. irradtime > 0.0_kr8 ) then
            time = time + coolingtime
            allocate(doseobj_ptr, stat=status)
            c_dose_ptr = c_loc(doseobj_ptr)
            call FispactOutputDataGetInventoryDoseRate(output_data monitor, &
                            & i, c_dose_ptr)
            ! get dose value from object
            doserate = doseobj_ptr%dose
            write(6,"(2E12.4)") time/FISPACT_YEAR_TO_SEC, doserate
        end if
    end do
    ...
```

8.2.5.2 C

```
    ...
    // dose rate after irradiation
    printf("\n\n");
    printf( "  TIME(yrs)    Dose(Sr/hr) \n" );
    double time = 0.0;
    int nrofentries = FispactOutputDataGetNrOfInventoryEntries(output_data, monitor);
    for (int i = 0; i < nrofentries+1; ++i) {
        // timesteps where flux amp = 0
        double coolingtime = FispactOutputDataGetInventoryValueByKey(output_data,
                            monitor, i, FISPACT_OUTPUT_DATA_INVENTORY_COOL_TIME);
        // timesteps where flux amp /= 0
        double irradtime = FispactOutputDataGetInventoryValueByKey(output_data,
                           monitor, i, FISPACT_OUTPUT_DATA_INVENTORY_IRRAD_TIME);
        // do not print initial dose rate
        if ( (coolingtime > 0.0) || (irradtime > 0.0) ) {
            time += coolingtime/FISPACT_YEAR_TO_SEC;
            // get dose rate from inventory
            FISPACT_OUTPUT_DATA_INVENTORY_DOSE_RATE doseobj;
            FispactOutputDataGetInventoryDoseRate(output_data, monitor, i, &doseobj);
            double doserate = doseobj.dose;
            printf("%12.4E %12.4E \n", time, doserate);
        }
    }
    ...
```

8.2.5.3 C++

...

```
    // dose after irradiation
```

```
    double time = 0.0;
    std::cout <<  "  \n\n";
    std::cout <<  std::setw(15) << "TIME(yrs)" << std::setw(16) << "Dose(Sr/hr)\n";
    for(int i=0; i< output.getNrOfInventoryEntries(); ++i){
        // timesteps where flux amp = 0
        double coolingtime = output.getInventoryValue( i,
                                        FISPACT_OUTPUT_DATA_INVENTORY_COOL_TIME);
        // timesteps where flux amp /= 0
        double irradtime = output.getInventoryValue(i,
                                        FISPACT_OUTPUT_DATA_INVENTORY_IRRAD_TIME);
        // do not print initial inventory heating
        if ( (coolingtime > 0.0) || (irradtime > 0.0) ) {
            time += coolingtime/FISPACT_YEAR_TO_SEC;
            // get dose rate from inventory
            FISPACT_OUTPUT_DATA_INVENTORY_DOSE_RATE doseobj =
                            output.getInventoryDoseRate(i);
            double doserate = doseobj.dose;
            std::cout << std::setw(15) << time << std::setw(15)
                      << doserate << "\n";
        }
    }
    ...
```

8.2.5.4 Python

```
    ...
# extract dose rate from after irrad
def print_dose_after_irrad(o):
    time = 0
    print(" ")
    print(" TIME(yrs) ", "  Dose(Sr/hr)")
    for i in range(o.getnrofinventoryentries()):
        # get cooling time
        coolingtime = o.getinventoryvalue(i, pf.INVENTORY_COOL_TIME())
        irradtime = o.getinventoryvalue(i, pf.INVENTORY_IRRAD_TIME())
        if coolingtime > 0.0 or irradtime > 0.0:
            ## sum time
            time += coolingtime
            # get dose raet object at each time step
            doseobj = o.getinventorydoserate(i)
            # get dose value from object
            doserate = doseobj.dose
            print(" {:.4E}   {:.4E} ".format(time/pf.util.YEAR_TO_SEC(), doserate))
     ...
```

8.3 Writing the Output data and logs to a file

The complete set of output data can be written to a JSON output file with the API's provided. The JSON schema matches closely to the output data schema shown earlier.

8.3.0.1 Fortran

```
    ...
    ! tear down
    call FispactOutputDataWrite(output_data, monitor, &
                        & f_c_string(trim(runname)//".json"))
    ...
```

8.3.0.2 C

```
    ...
    // write output JSON
    char output_filename[128];
    strcpy(output_filename, runname);
    strcat(output_filename, ".json");
    FispactOutputDataWrite(output_data, monitor, output_filename);
    ...
```

8.3.0.3 C++

```
    ...
    // write output JSON
    fp::io::ToFile(output, monitor, runname + ".json");
    ...
```

8.3.0.4 Python

```
    ...
# write complete inventory to json output
pf.io.to_file(o, m, "{}.json".format(runname))
    ...
```

9 Clean up and Finalising API scripts

Once all of the desired actions have been included in an API script the objects and the global data should be deallocated. There are tear down and finalise methods to handle these procedures as part of the FISPACT-II API. The monitor object can also be written to a file, this will produce an output analogous to the standard FISPACT-II log file. This will allow the user to study any warnings that FISPACT-II as produced so that the results can be evaluated safely.

9.1 Fortran

```
...
    ! clean up
    call FispactInputDataDestroy(input_data, monitor)
    call FispactOutputDataDestroy(output_data, monitor)
    call FispactNuclearDataDestroy(nuclear_data, monitor)
    call FispactNuclearDataReaderDestroy(nuclear_data_reader, monitor)
    call FispactFinalise(monitor)

    ! write log
    call MonitorWriteToFile(monitor, f_c_string(trim(runname)//".log"))
    call MonitorDestroy(monitor)
...
```

9.2 C

```
...
    // clean up
    FispactInputDataDestroy(input_data, monitor);
    FispactOutputDataDestroy(output_data, monitor);
    FispactNuclearDataDestroy(nuclear_data, monitor);
    FispactNuclearDataReaderDestroy(nd_reader, monitor);

    char log_filename[128];
    strcpy(log_filename, runname);
    strcat(log_filename, ".log");
    MonitorWriteToFile(monitor, log_filename);

    FispactFinalise(monitor);
    MonitorDestroy(monitor);
...
```

9.3 C++

```
...
    // clean up
    fp::GlobalFinalise(monitor);
    fp::io::ToFile(monitor, runname + ".log");
...
```

9.4 Python

```
...
# tear down methods
pf.finalise(m)
```

```
# write logs to file
pf.io.to_file(m, "{}.log".format(runname))
...
```

10 Tools and Utilities

10.1 Group Convert

Like FISPACT-II the API makes use of group wise nuclear cross section data. It is essential for the correct collapsing of cross section with flux spectra that those spectra are in the same group structure. FISPACT-II allows a user to convert a flux spectrum from one group structure to another and the API also contains this functionality. It should be noted however that the group convert algorithms can not add information. If you convert from a sparse group structure to a fine structure detail can be lost. It is always safer to convert from a fine group structure to a less coarse structure. FISPACT-II group convert can be performed by a equal-lethargy per bin approximation or a equal-energy approximation (the functionality of the CNVTYPE keyword). It should be noted that the group convert methods expect the spectra and boundaries in order of ascending energy.

Unlike the group convert tool present in the command line version of FISPACT-II the API group convert is generic, it can convert from any input structure to any output structure. Allowing the user to fully utilise the functionally. The examples below show how a 709 group flux can be converted, by lethargy, to a 1102 group flux for use with the TENDL2019 cross section library, these examples make use of the included group structure bounds, see table 7.

10.1.1 Fortran

```
    ...
real(kr8) :: outflux(0:1101)
real(kr8) :: flux(0:708)
    ...
! group convert: input fluxes and group structure must be in order of ascending
! energy. The out flux in also in order of ascending energy
call FispactGroupConvertByLethargy(monitor, 709_ki4, FISPACT_GROUP_709(709:0:-1), &
      & flux(708:0:-1), 1102_ki4, FISPACT_GROUP_1102(1102:0:-1), outflux(0:1101))
    ...
```

10.1.2 C

```
    ...
// reverse the in bounds to get ascending order
double inbounds[710];
memcpy(inbounds, FISPACT_GROUP_709, (in_group_len+1)*sizeof(double));
```

```
    reverse_range(inbounds, 0, in_group_len);
    // reverse flux to ascending order
    reverse_range(flux, 0, (in_group_len-1) );

    // reverse the out bounds to get ascending order
    double outflux[1102];
    double outbounds[1103];
    memcpy(outbounds, FISPACT_GROUP_1102, (out_group_len+1)*sizeof(double));
    reverse_range(outbounds, 0, out_group_len);

    // perfom group convert
    FispactGroupConvertByLethargy(monitor, in_group_len, inbounds,
                                  flux, out_group_len, outbounds,
                                  outflux);
        ...
```

10.1.3 C++

```
        ...
    // perfom group convert
    reverse(flux.begin(), flux.end());
    std::vector<double> outflux = fp::groupconvert::GroupConvertByLethargy(monitor,
                                        fp::groups::G709(), flux, fp::groups::G1102() );
        ...
```

10.1.4 Python

```
        ...
    # reverse the order of the flux to assending in energy
    flux.reverse()
    #  convert 709 group spectrum to 1102 groups to match TENDL2019 group structure
    outflux = pf.groupconvert.bylethargy(m, pf.groups.G709(), flux, pf.groups.G1102())
        ...
```

10.2 Elemental Data

In the command liner version of FISPACT-II the elemental data (isotopic abundances, displacement energies etc) are hard coded, the values themselves taken from NIST. These values are also used by the FISPACT-II API and are loaded as part of the initialise methods. The API provides methods to alter and/or interrogate this global data if a user desires. This is can be done at any point after the initialise call, but will only affect the calculations if called before material definition are set in the input data object. With the elemental data getters and setters a user can interrogate/alter an elements:

- Isotopic Abundances
- Relative Atomic Mass
- Standard Density
- Atomic Displacement Energy

It is also possible to reset the elemental data to its default values at any time. In the following examples the natural abundance of Lithium is altered to be 60% ^{6}Li. If this is done before setting a material by element (SetMass methods) the resultant materials will use the altered abundance when determining which nuclides will be initially present. The getter methods can be used to probe the FISPACT-II default values, these are detailed in the code reference sheets.

10.2.1 Fortran

```
real(kr8) :: Li_abundance(0:1)
integer(ki4) :: Li_zais(0:1)
integer(ki4) :: Li_z
    ...
! set elemental abundace for Li
Li_zais = [ FispactGetZai(monitor, f_c_string("Li6")),&
          & FispactGetZai(monitor, f_c_string("Li7"))]
Li_abundance = [ 0.6, 0.4 ]
Li_z = FispactGetAtomicNumberFromElementName(monitor, f_c_string("Li"))
call FispactElementalDataSetNaturalIsotopeAbundances( monitor, Li_z,&
                                    & size(Li_zais), Li_zais, Li_abundance )
    ...
```

10.2.2 C

```
    ...
// set Li abundances
int Li_zais[2] = { FispactGetZai(monitor, "Li6"),
                   FispactGetZai(monitor, "Li7")};
double Li_abundance[2] = {0.6, 0.4};
int Li_z = FispactGetAtomicNumberFromElementName(monitor, "Li");
FispactElementalDataSetNaturalIsotopeAbundances( monitor, Li_z, 2, Li_zais,
                                                                    Li_abund
    ...
```

10.2.3 C++

```
    ...
// set Li abundances
int Li_z = fp::util::GetAtomicNumberFromElementName(monitor, "Li");
std::vector<int> Li_zai = { fp::util::GetZai(monitor, "Li6"),
                                                fp::util::GetZai(monitor, "Li7") };
std::vector<double> Li_abund = { 0.6, 0.4 };
fp::elementaldata::setNaturalIsotopeAbundances(monitor, Li_z , Li_zai, Li_abund );
    ...
```

10.2.4 Python

```
    ...
# set Li abundance
Li_z = pf.util.z_from_element(m, "Li")
Li_zai = [ pf.util.zai_from_name(m, "Li6"), pf.util.zai_from_name(m, "Li7") ]
Li_abund = [ 0.6, 0.4 ]
pf.elementaldata.setnaturalisotopeabundances(m, Li_z , Li_zai, Li_abund )
    ...
```

10.3 Nuclide Utilities

Included with the FISPACT-II API are a number of utility methods for determining nuclide names, atomic numbers, elements and ZAIs. Also included are method to retrieve the type of spectrum as a string from the spectrum type key. These methods will likely be made use of often as the API data structures are based on a nuclides integer ZAI's for identification rather than there names. These methods have been implemented throughout the examples used in this document, here the quantity input and output are given as the precise syntax is shown elsewhere. The code reference sheets cover these methods in complete detail. Table 9 details which quantities can be converted to other quantities.

Table 9: Nuclear Data projectile keys

In Quantity	Type	Out Quantity	Type
Nuclide name	string	ZAI	int
Element name	string	Atomic number	int
ZAI	int	Nuclide name	string
ZAI	int	Element name	string
Atomic number	int	Element name	string

This page has been left intentionally blank.

References

This page has been left intentionally blank.

APPENDICES

A Fortran Derived Types

The Fortran FISPACT-II API (which underpins all of the other language APIs) makes use a of a set of derived types, primarily as data structures. This section details these type for user convenience. These are also sated in the code reference sheets.

A.1 Nuclide Output Data type

```
!> Simple struct for inventory nuclide output data
type, bind(C), public :: fispact_inventory_nuclide_data_t
    !< The element name
    character(kind=c_char) :: element(NUCLIDE_ELEMENT_LENGTH+1)
    !< The element state
    character(kind=c_char) :: state(NUCLIDE_STATE_LENGTH+1)
    !< The atomic number
    integer(ki4)           :: isotope                          = 0_ki4
    !< The ZAI
    integer(ki4)           :: zai                              = 0_ki4
    !< The half life (s)
    real(kr8)              :: half_life                        = 0.0_kr8
    !< The number of atoms
    real(kr8)              :: atoms                            = 0.0_kr8
    !< The grams (g)
    real(kr8)              :: grams                            = 0.0_kr8
    !< The activity (Bq)
    real(kr8)              :: activity                         = 0.0_kr8
    !< The alpha activity (Bq)
    real(kr8)              :: alpha_activity                   = 0.0_kr8
    !< The beta activity (Bq)
    real(kr8)              :: beta_activity                    = 0.0_kr8
    !< The gamma activity (Bq)
    real(kr8)              :: gamma_activity                   = 0.0_kr8
    !< The total heat (kW)
    real(kr8)              :: heat                             = 0.0_kr8
    !< The alpha heat (kW)
    real(kr8)              :: alpha_heat                       = 0.0_kr8
    !< The beta heat (kW)
    real(kr8)              :: beta_heat                        = 0.0_kr8
    !< The gamma heat (kW)
    real(kr8)              :: gamma_heat                       = 0.0_kr8
     !< The dose rate (Sv/hr)
    real(kr8)              :: dose                             = 0.0_kr8
    !< The ingestion (Sv)
    real(kr8)              :: ingestion                        = 0.0_kr8
    !< The inhalation (Sv)
    real(kr8)              :: inhalation                       = 0.0_kr8
end type fispact_inventory_nuclide_data_t
```

A.2 Nuclide Output Dose type

```
!> Simple struct for inventory dose rate output data
type, bind(C), public :: fispact_inventory_dose_rate_data_t
        !< The dose rate type
    character(kind=c_char) :: type(OUTPUT_DATA_SHORT_NAME_LENGTH+1)
    !< The dose rate distance (m)
    real(kr8)              :: distance   = 0.0_kr8
     !< The dose rate mass (kg)
    real(kr8)              :: mass       = 0.0_kr8
    !< The dose rate dose (Sieverts/hour)
    real(kr8)              :: dose       = 0.0_kr8
end type fispact_inventory_dose_rate_data_t
```

A.3 Decay Data Spectral type

```
!> Simple struct for spectral line data
type, bind(C), public :: fispact_spectral_line_data_t
    real(kr8) :: energy           = 0.0_kr8 ! line energy (eV)
    real(kr8) :: energy_unc       = 0.0_kr8 ! line energy uncertainty (eV)
    real(kr8) :: intensity        = 0.0_kr8 ! line intensity (-)
    real(kr8) :: intensity_unc    = 0.0_kr8 ! line intensity uncertainty (-)
end type fispact_spectral_line_data_t
```

A.4 Callback types

```
!> The load callback state type when reading the nuclear data
type, bind(C), public :: fispact_nuclear_data_reader_callback_t
    character(kind=c_char)    :: key(MAX_KEY_LENGTH)
    character(kind=c_char)    :: path(MAX_PATH_LENGTH)
    integer(ki4)              :: index = 0_ki4
    integer(ki4)              :: total = 0_ki4
end type fispact_nuclear_data_reader_callback_t

!> Simple struct for callback data for fispact compute
type, bind(C), public :: fispact_process_callback_t
    character(kind=c_char)    :: process(MAX_PATH_LENGTH)
    integer(ki4)              :: index = 0_ki4
    integer(ki4)              :: total = 0_ki4
end type fispact_process_callback_t

!> Simple struct for callback data for index data
type, bind(C), public :: fispact_nuclear_data_index_callback_t
    integer(ki4)              :: zai   = 0_ki4
    integer(ki4)              :: index = 0_ki4
    integer(ki4)              :: total = 0_ki4
end type fispact_nuclear_data_index_callback_t
```

B Keyword Equivalents

The FISPACT-II API maintains much of the functionality of the standard command line driven version of FISPACT-II has via its keywords. In some cases this functionality occurs in a different form and this manual has attempted to cover these cases.

Table 10: The list of keywords present in FISPACT-II version 5.0 and which of the APIs support the functionality. If a keyword is not supported via a setter then N (No) is used, and D (deprecated) indicates that using a API approach, without any file writing, renders a lot of existing keywords unnecessary.

keyword	supported (Y=yes, N=no, D=deprecated)	
	API	**notes**
ALLDISPEN	Y	Fully Supported via FispactElementalData -SetAtomicDisplacementEnergy
ATDISPEN	Y	Fully Supported via FispactElementalData -SetAtomicDisplacementEnergy
ATOMS	Y	Fully supported via FispactInputDataSetSchedule
ATWO	Y	Fully supported via FispactInputDataSetIncludeLegalLimits
BREMSSTRAHLUNG	N	To be supported in pure API in future versions
CLEAR	Y	Fully supported via FispactInputDataSetEnableClearance
CLOBBER	D	No longer necessary, since pure API does not write files
CNVTYPE	Y	Fully supported see GRPCONVERT
COVARIANCE	Y	Fully supported via FispactNuclearDataReaderSetRead -Covariances
CULTAB	D	No longer necessary, since API does not write files

Table 10 continued from previous page

keyword	supported (Y=yes, N=no, D=deprecated) API	notes
CUMFYLD	Y	Fully supported via FispactNuclearDataReaderSetFission-YieldCumulativeOption
DENSITY	Y	Fully supported via FispactInputDataSetDensity
DEPLETION	N	To be supported in API in future versions
DOSE	Y	Fully supported via FispactInputDataSetDoseType, FispactInputDataSetDoseDistance
END	D	No longer necessary, since API does not read files
ENDPULSE	D	No longer necessary, since pulses can be performed using FispactInputDataSetSchedule
ERROR	D	No longer necessary, since sensitivity analysis can be performed using the API and nuclear data can be changed via setters
FISCHOOSE	N	Pathways analysis is excluded from API
FISPACT	D	No longer necessary, since no separation is made between the control, initial and inventory phases
FISYIELD	Y	Fully supported via FispactInputDataSetFissionYieldOption, FispactInputDataSetFissionYieldList
FLUX	Y	Fully supported via FispactInputDataSetSchedule
FUEL	Y	Fully supported via FispactInputDataSetFuel, FispactInputDataAppendFuel FispactInputDataResetFuel

Table 10 continued from previous page

keyword	supported (Y=yes, N=no, D=deprecated)	
	API	**notes**
FULLXS	D	No longer necessary, full cross section data is read in-memory
GENERIC	N	Pathways analysis is excluded from API
GETDECAY	D	No longer necessary, API uses in-memory decay data
GETXS	D	No longer necessary, API uses in-memory cross section data. Collapse is performed with every calculation, since API allows for changing incident particle spectra
GRAPH	D	No longer necessary, since API does not write files
GROUP	Y	Fully supported via FispactInputDataSetGammaEnergyBounds
GRPCONVERT	Y	Fully supported via FispactGroupConvertByEnergy and FispactGroupConvertByLethargy
HALF	Y	Fully supported, half lives are always output
HAZARDS	Y	Fully supported, hazard data is always included
INDEXPATH	N	Pathways analysis is excluded from API
JSON	Y	Whilst API does not write files by default, output data can be serialized to JSON via FispactOutputDataWrite
LIBVERSION	D	No longer necessary, set the nuclear data object with whichever data format the user requires
LOGLEVEL	D	No longer necessary, since the monitor API can control verbosity

Table 10 continued from previous page

keyword	supported (Y=yes, N=no, D=deprecated)	
	API	**notes**
LOOKAHEAD	N	Pathways analysis is excluded from API
MASS	Y	Fully supported via FispactInputDataSetMass, FispactInputDataSetMassTotal
MCSAMPLE	N	Sensitivity analysis is excluded from API
MCSEED	N	Sensitivity analysis is excluded from API
MIND	Y	Fully supported via FispactInputDataSetAtomsThreshold
MONITOR	D	No longer necessary, since the FispactProcess can take a callback for users to define custom monitors
NOCOMP	D	No longer necessary, since elemental data can be post processed from output object
NOERROR	Y	Fully supported via FispactNuclearDataReaderSetReadUncertainties
NOFISS	D	No longer necessary, since nuclear data is in-memory
NOSORT	D	No longer necessary, since callers can sort nuclides by post processing output
NOSTABLE	Y	Fully supported via FispactInputDataSetNoStable
NOT1	D	No longer necessary, since API does not write files
NOT2	D	No longer necessary, since API does not write files
NOT3	D	No longer necessary, since API does not write files

Table 10 continued from previous page

keyword	supported (Y=yes, N=no, D=deprecated)	
	API	**notes**
NOT4	D	No longer necessary, since API does not write files
NOXRAY	Y	Fully supported via FispactInputExcludeXrays
NUCGRAPH	D	No longer necessary, since API does not write files
OVER	N	Overwriting energy dependent cross section data is possible with nuclear data object, overwriting collapsed values is to be included in API in future versions
PARTITION	N	To be supported in API in future versions
PATH	N	Pathways analysis is excluded from API
PATHRESET	N	Pathways analysis is excluded from API
POWER	N	To be supported in API in future versions
PRINTLIB	D	No longer necessary, since API does not write files, utility tools to provide same functionality
PROBTABLE	N	To be supported in API in future versions
PROJECTILE	Y	Fully supported via FispactInputDataSetProjectile
PULSE	D	No longer necessary, since pulses can be performed using FispactInputDataSetSchedule
READGG	Y	Fully supported via FispactInputDataSetGammaEnergyBounds
READSF	Y	Fully supported, via nuclear data reader

Table 10 continued from previous page

keyword	supported (Y=yes, N=no, D=deprecated) API	notes
RESULT	N	Pathways analysis is excluded from API
ROUTES	N	Pathways analysis is excluded from API
SAVELINES	Y	Fully supported via FispactInputDataSetEnableSaveLines
SENSITIVITY	N	Sensitivity analysis is excluded from API
SORTDOMINANT	N	Pathways analysis is excluded from API
SPECTRUM	D	No longer necessary, since no output file is written with API, does the same as ATOMS
SPEK	Y	Fully supported, via FispactInputDataSetEnableApproxGammaSpec
SPLIT	D	No longer necessary, since no output file is written with API
SSFCHOOSE	N	To be supported in API in future versions
SSFDILUTION	N	To be supported in API in future versions
SSFFUEL	N	To be supported in API in future versions
SSFGEOMETRY	N	To be supported in API in future versions
SSFMASS	N	To be supported in API in future versions
STEP	D	No longer necessary, since no output file is written with API, does the same as ATOMS

Table 10 continued from previous page

keyword	supported (Y=yes, N=no, D=deprecated)	
	API	**notes**
TAB1	D	No longer necessary, since API does not write files
TAB2	D	No longer necessary, since API does not write files
TAB3	D	No longer necessary, since API does not write files
TAB4	D	No longer necessary, since API does not write files
TIME	Y	Fully supported via FispactInputDataSetSchedule
TOLERANCE	Y	Fully supported via FispactInputDataSetSolverTolerance
UNCERTAINTY	N	To be supported in API in future versions
UNCTYPE	N	To be supported in API in future versions
USEFISSION	Y	Fully supported via FispactInputDataSetEnableUseFission
USESPALLATION	N	To be supported in API in future versions
USEXSEXTRA	Y	Fully supported via nuclear data reader using FISPACT_ND_XS_EXTRA_KEY key
WALL	N	To be supported in API in future versions
XSTHRESHOLD	Y	Fully supported, via FispactInputDataSetXSThreshold
ZERO	N	Pathways analysis is excluded from API

C Fortran API Examples

C.1 FNS Inconel

Below is the complete FNS Inconel irradiation used as an example throughout this work using the Fortran FISPACT-II API.

```
! Fispact API FNS Inconel TENDL2019, decay 2020
program API_example
    ! import fispact api library
    use fispactapi
    implicit none
    ! define types and variables
    ! required for complete API functionality
    character(*), parameter :: runname = "FNSInconel_FortAPI_tendl19"

    character(len=FISPACT_MAX_PATH_LENGTH) :: nd_base_path
    type(c_ptr) :: monitor
    type(c_ptr) :: input_data
    type(c_ptr) :: output_data
    type(c_ptr) :: nuclear_data
    type(c_ptr) :: nuclear_data_reader
    ! callback pointers
    type(c_ptr) :: c_state_ptr
    type(c_ptr) :: c_nuclide_ptr
    type(fispact_nuclear_data_reader_callback_t), pointer :: reader_state_native_ptr
    type(fispact_process_callback_t), pointer :: process_state_native_ptr
    character(:), allocatable :: lastkey
    integer(ki4) :: status
    ! extra for this code
    integer(ki4) :: i,nrofnuclides
    real(kr8) :: coolingtime, irradtime
    real(kr8) :: time, mass, halflife
    real(kr8) :: heating, total_heat
    real(kr8) :: percent(0:3)
    integer(ki4) :: atomic_numbers(0:3)
    real(kr8) :: cooltimes(0:20)
    real(kr8) :: outflux(0:1101)
    real(kr8) :: flux(0:708)
    character(FISPACT_NUCLIDE_NAME_LENGTH) :: nuclidename
    real(kr8), allocatable :: dom_heat(:)
    integer(ki4), allocatable :: dom_zai(:)

    ! Zero arrays
    flux = 0.0_kr8
    outflux = 0.0_kr8
    atomic_numbers = 0_ki4
    percent = 0.0_kr8
    cooltimes = 0.0_kr8

    ! get path/to/nuclear data from command line
    call get_command_argument(1_ki4, nd_base_path)
    if (len_trim(nd_base_path) == 0) then
```

```
        write(*, *) "Argument required must be path to nuclear data library base"
        stop 1
    end if

    ! initialse objects and global data
    monitor = MonitorCreate()
    input_data = FispactInputDataCreate(monitor)
    output_data = FispactOutputDataCreate(monitor)
    nuclear_data = FispactNuclearDataCreate(monitor)
    nuclear_data_reader = FispactNuclearDataReaderCreate(monitor)
    call MonitorSetVerbosityLevel(monitor, MONITOR_SEVERITY_TRACE)

    ! initialise static and global data
    call FispactInitialise(monitor)

    ! load Nuclear Data
    ! set projectile
    call  FispactNuclearDataSetProjectile(nuclear_data, monitor, &
                                          & FISPACT_PROJECTILE_TYPE_NEUTRON)
    ! loading TENDL2019 and Decay2020 alongside hazards and radiological data
    ! from Decay2012 files
    ! set paths to the nuclear data, same paths and keys as those found in a FILES file
    ! TENDL2019 load, using the decay2020 index file
    call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
        & FISPACT_ND_IND_NUC_KEY, &
        & f_c_string(trim(nd_base_path)//"/decay2020/tendl19_decay2020_index"))
    call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
        & FISPACT_ND_XS_ENDF_KEY, &
        & f_c_string(trim(nd_base_path)//"/TENDL2019data/gendf-1102"))
    call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
        & FISPACT_ND_PROB_TAB_KEY, &
        & f_c_string(trim(nd_base_path)//"/TENDL2019data/tp-1102-294"))
    ! fission yeild data
    call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
        & FISPACT_ND_FY_ENDF_KEY, &
        & f_c_string(trim(nd_base_path)//"/GEFY61data/gefy61_nfy"))
    call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
        & FISPACT_ND_SF_ENDF_KEY, &
        & f_c_string(trim(nd_base_path)//"/GEFY61data/gefy61_nfy"))
    ! Deacy2020 decay data
    call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
        & FISPACT_ND_DK_ENDF_KEY, &
        & f_c_string(trim(nd_base_path)//"/decay2020/decay_2020"))
    ! attenuation data required for dose rate calculation
    call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
        & FISPACT_ND_ABSORP_KEY, &
        & f_c_string(trim(nd_base_path)//"/decay/abs_2012"))
    ! Hazards and radiological data from decay2012 files
    call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
        & FISPACT_ND_HAZARDS_KEY, &
        & f_c_string(trim(nd_base_path)//"/decay/hazards_2012"))
    call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
        & FISPACT_ND_CLEAR_KEY, &
        & f_c_string(trim(nd_base_path)//"/decay/clear_2012"))
    call FispactNuclearDataReaderSetPath(nuclear_data_reader, monitor, &
        & FISPACT_ND_A2DATA_KEY, &
```

```
        & f_c_string(trim(nd_base_path)//"/decay/a2_2012"))

    ! allocate our callback for reader
    allocate(reader_state_native_ptr, stat=status)
    c_state_ptr = c_loc(reader_state_native_ptr)

    ! load the nuclear data and cleanup reader, we no longer need it
    lastkey = ""
    call FispactNuclearDataReaderLoad(nuclear_data_reader, nuclear_data, &
                                      & monitor, c_state_ptr, c_callback_load)
    deallocate(reader_state_native_ptr)

    ! set inputs
    call FispactInputDataSetName(input_data, monitor, f_c_string(runname))
    !
    ! set flux details
    !
    flux(0:708) = [ &
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 2.9806E+08, 7.2391E+08, 7.3338E+08, 7.4309E+08,&
        &2.1367E+09, 3.1298E+09, 1.2641E+09, 6.3931E+08, 6.0755E+07, 3.0446E+07,&
        &2.1633E+07, 1.8685E+07, 1.5579E+07, 1.5817E+07, 1.5557E+07, 1.3784E+07,&
        &1.2367E+07, 1.1450E+07, 1.0447E+07, 1.0622E+07, 1.0742E+07, 1.0310E+07,&
        &1.0493E+07, 1.0576E+07, 1.0464E+07, 1.0659E+07, 1.0251E+07, 9.9709E+06,&
        &1.0168E+07, 2.2430E+07, 2.0564E+07, 1.9520E+07, 1.8852E+07, 1.8621E+07,&
        &1.8803E+07, 1.9212E+07, 1.9312E+07, 1.8698E+07, 1.6903E+07, 1.6879E+07,&
        &1.7133E+07, 1.6529E+07, 1.6401E+07, 1.6155E+07, 1.6995E+07, 1.7381E+07,&
        &1.6948E+07, 1.6652E+07, 1.6526E+07, 1.6376E+07, 1.6283E+07, 1.6050E+07,&
        &1.5988E+07, 1.5512E+07, 1.5916E+07, 1.5694E+07, 1.5805E+07, 1.5977E+07,&
        &1.5868E+07, 1.5489E+07, 1.5761E+07, 1.5463E+07, 1.5340E+07, 1.5227E+07,&
        &1.5058E+07, 1.5031E+07, 1.4791E+07, 1.4435E+07, 1.4673E+07, 1.4417E+07,&
        &1.4095E+07, 1.4044E+07, 1.3948E+07, 1.3512E+07, 1.3097E+07, 1.2946E+07,&
        &1.2726E+07, 1.2614E+07, 1.2580E+07, 1.1950E+07, 1.1605E+07, 1.1272E+07,&
        &1.0946E+07, 1.0604E+07, 1.0435E+07, 1.0073E+07, 9.7617E+06, 9.5178E+06,&
        &9.0618E+06, 8.7300E+06, 8.4719E+06, 7.9208E+06, 7.8539E+06, 7.5366E+06,&
        &7.2098E+06, 7.1581E+06, 6.6208E+06, 6.3730E+06, 6.2884E+06, 6.0896E+06,&
        &5.8604E+06, 5.7249E+06, 5.6011E+06, 4.9963E+06, 4.9964E+06, 4.7358E+06,&
        &4.5749E+06, 4.4232E+06, 4.3949E+06, 4.0158E+06, 3.6967E+06, 3.5937E+06,&
        &3.4541E+06, 3.3024E+06, 3.3809E+06, 3.3065E+06, 2.9941E+06, 2.7719E+06,&
        &2.3252E+06, 2.4091E+06, 2.3406E+06, 2.5279E+06, 2.5740E+06, 2.4514E+06,&
        &2.1743E+06, 1.9598E+06, 1.9485E+06, 1.8537E+06, 1.8537E+06, 1.6554E+06,&
        &1.5054E+06, 1.5054E+06, 1.6360E+06, 1.6701E+06, 1.6230E+06, 1.6230E+06,&
        &1.4395E+06, 1.3368E+06, 1.2342E+06, 1.2342E+06, 1.2342E+06, 1.1798E+06,&
        &1.1473E+06, 1.0320E+06, 1.0317E+06, 1.0217E+06, 9.9629E+05, 9.9628E+05,&
```

```
&8.8628E+05,  8.0295E+05,  8.0295E+05,  8.0295E+05,  9.7740E+05,  1.0185E+06,&
&9.4898E+05,  9.4898E+05,  8.8288E+05,  8.1851E+05,  7.9764E+05,  7.8468E+05,&
&7.8355E+05,  7.3070E+05,  6.9329E+05,  6.9319E+05,  6.8316E+05,  6.5778E+05,&
&6.5778E+05,  6.5778E+05,  6.5778E+05,  6.5778E+05,  5.7473E+05,  5.6058E+05,&
&5.6057E+05,  5.6058E+05,  5.6057E+05,  5.5658E+05,  5.5123E+05,  5.4074E+05,&
&5.1003E+05,  5.1002E+05,  5.1002E+05,  4.7404E+05,  4.7393E+05,  4.7393E+05,&
&4.7393E+05,  4.7393E+05,  4.3415E+05,  4.0394E+05,  4.0393E+05,  4.0393E+05,&
&4.0394E+05,  4.0163E+05,  3.8742E+05,  3.8742E+05,  3.8742E+05,  3.8742E+05,&
&3.8742E+05,  3.8988E+05,  3.9088E+05,  3.9088E+05,  3.8714E+05,  3.8264E+05,&
&3.6002E+05,  3.0248E+05,  3.0621E+05,  3.3614E+05,  3.3614E+05,  3.1955E+05,&
&3.3173E+05,  3.2173E+05,  3.1802E+05,  3.0931E+05,  3.0342E+05,  2.9074E+05,&
&2.7357E+05,  2.7357E+05,  2.7357E+05,  2.7357E+05,  2.7357E+05,  2.3473E+05,&
&2.3458E+05,  2.3458E+05,  2.3458E+05,  2.3459E+05,  2.2167E+05,  2.1183E+05,&
&2.1184E+05,  2.1183E+05,  2.1184E+05,  2.0913E+05,  1.9234E+05,  1.9233E+05,&
&1.9234E+05,  1.9234E+05,  1.9233E+05,  1.7840E+05,  1.7271E+05,  1.7271E+05,&
&1.7271E+05,  1.7271E+05,  1.6986E+05,  1.6259E+05,  1.6259E+05,  1.6259E+05,&
&1.6259E+05,  1.6259E+05,  1.6342E+05,  1.6356E+05,  1.6356E+05,  1.6356E+05,&
&1.6356E+05,  1.5175E+05,  1.3570E+05,  1.3570E+05,  1.3570E+05,  1.3570E+05,&
&1.3570E+05,  1.3554E+05,  1.3554E+05,  1.3554E+05,  1.3554E+05,  1.3554E+05,&
&1.3184E+05,  1.2902E+05,  1.2902E+05,  1.2902E+05,  1.2902E+05,  1.2699E+05,&
&1.1435E+05,  1.1435E+05,  1.1435E+05,  1.1435E+05,  1.1435E+05,  1.0523E+05,&
&1.0150E+05,  1.0149E+05,  1.0150E+05,  1.0149E+05,  9.5928E+04,  8.1656E+04,&
&8.1657E+04,  8.1656E+04,  8.1656E+04,  8.1656E+04,  9.2951E+04,  9.4913E+04,&
&9.4913E+04,  9.4913E+04,  9.4913E+04,  8.6281E+04,  7.4521E+04,  7.4521E+04,&
&7.4521E+04,  7.4521E+04,  7.4521E+04,  6.1552E+04,  6.1482E+04,  6.1482E+04,&
&6.1482E+04,  6.1482E+04,  5.6493E+04,  5.2666E+04,  5.2666E+04,  5.2666E+04,&
&5.2666E+04,  5.4200E+04,  6.3842E+04,  6.3842E+04,  6.3842E+04,  6.3841E+04,&
&6.3841E+04,  5.7042E+04,  5.4245E+04,  5.4245E+04,  5.4245E+04,  5.4245E+04,&
&5.3570E+04,  5.1834E+04,  5.1834E+04,  5.1834E+04,  5.1834E+04,  5.1834E+04,&
&4.8436E+04,  4.7843E+04,  4.7842E+04,  4.7842E+04,  4.7843E+04,  4.5208E+04,&
&4.1608E+04,  4.1608E+04,  4.1608E+04,  4.1608E+04,  4.1608E+04,  3.7355E+04,&
&3.7329E+04,  3.7329E+04,  3.7329E+04,  3.7329E+04,  3.8387E+04,  3.9202E+04,&
&3.9202E+04,  3.9201E+04,  3.9202E+04,  3.9728E+04,  4.3059E+04,  4.3059E+04,&
&4.3059E+04,  4.3059E+04,  4.3059E+04,  2.9553E+04,  2.3978E+04,  2.3978E+04,&
&2.3978E+04,  2.3978E+04,  2.5107E+04,  2.8026E+04,  2.8026E+04,  2.8026E+04,&
&2.8026E+04,  2.8025E+04,  2.2712E+04,  2.1778E+04,  2.1778E+04,  2.1778E+04,&
&2.1778E+04,  2.2753E+04,  2.4090E+04,  2.4090E+04,  2.4090E+04,  2.4090E+04,&
&2.4090E+04,  1.8954E+04,  1.8918E+04,  1.8918E+04,  1.8918E+04,  1.8918E+04,&
&1.7435E+04,  1.6289E+04,  1.6289E+04,  1.6289E+04,  1.6289E+04,  1.6196E+04,&
&1.5601E+04,  1.5601E+04,  1.5601E+04,  1.5601E+04,  1.5601E+04,  1.4146E+04,&
&1.3543E+04,  1.3543E+04,  1.3543E+04,  1.3543E+04,  1.3819E+04,  1.4535E+04,&
&1.4535E+04,  1.4535E+04,  1.4535E+04,  1.4535E+04,  1.4109E+04,  1.4033E+04,&
&1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,&
&1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,&
&1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,&
&1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4034E+04,&
&1.4033E+04,  1.4033E+04,  1.4034E+04,  1.4033E+04,  1.4033E+04,  1.1922E+04,&
&1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,&
&1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,&
&1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,&
&1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,&
&1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,&
&1.1922E+04,  1.1923E+04,  1.1922E+04,  1.1922E+04,  1.1923E+04,  1.1922E+04,&
&1.1922E+04,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,&
```

```
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,&
        &0.0000E+00 ]
    ! group convert: input fluxes and group structure must be in order of ascending
    ! energy. The out flux in also in order of ascending energy
    call FispactGroupConvertByLethargy(monitor, 709_ki4, FISPACT_GROUP_709(709:0:-1), &
           & flux(708:0:-1), 1102_ki4, FISPACT_GROUP_1102(1102:0:-1), outflux(0:1101))
    call FispactInputDataSetFlux(input_data, monitor, 1102_ki4, &
           & FISPACT_GROUP_1102(1102:0:-1), outflux(0:1101))
    call FispactInputDataSetFluxWallLoading(input_data, monitor, 1.0_kr8)
    call FispactInputDataSetFluxName(input_data, monitor, f_c_string("FNS Inconel Flux"))
    !
    ! material details
    !
    ! set density (g/cm3)
    call FispactInputDataSetDensity(input_data, monitor,  8.42_kr8)
    ! set atoms threshold, MIND keyword
    call FispactInputDataSetAtomsThreshold(input_data, monitor, 1.0e3_kr8)
    ! set total mass (kg)
    call FispactInputDataSetMassTotal(input_data, monitor, 1.0E-3_kr8)
    ! set material my weight percentage, MASS keyword
    ! atomic numbers for elements to be included, using utilities to convert
    ! from element names
    atomic_numbers = [ FispactGetAtomicNumberFromElementName(monitor, 'Ni'),&
                       &FispactGetAtomicNumberFromElementName(monitor, 'Mn'),&
                       &FispactGetAtomicNumberFromElementName(monitor, 'Fe'),&
                       FispactGetAtomicNumberFromElementName(monitor, 'Cr') ]
    ! percentge of each element
    percent = [ 75.82_kr8, 0.39_kr8, 7.82_kr8, 15.97_kr8 ]
    ! input materials
    call FispactInputDataSetMass(input_data, monitor, size(percent), &
                                  & atomic_numbers, percent)
    !
    ! set irradiation schedule
```

```
!
! 5 min irradiation
call FispactInputDataSetSchedule(input_data, monitor, 1_ki4, &
                                 & [5.0_kr8*FISPACT_MIN_TO_SEC], [1.116E+10_kr8])
! cooling steps
cooltimes = [36.0_kr8, 15.0_kr8, 16.0_kr8, 15.0_kr8, 15.0_kr8, 26.0_kr8, &
            & 33.0_kr8, 36.0_kr8, 53.0_kr8, 66.0_kr8, 66.0_kr8, 97.0_kr8, &
            & 127.0_kr8, 126.0_kr8, 187.0_kr8, 246.0_kr8, 244.0_kr8, &
            & 246.0_kr8, 428.0_kr8, 606.0_kr8, 607.0_kr8 ]
do i=0,size(cooltimes)-1
    call FispactInputDataAppendSchedule(input_data, monitor, cooltimes(i), 0.0_kr8)
end do

! run fispact
allocate(process_state_native_ptr, stat=status)
c_state_ptr = c_loc(process_state_native_ptr)
call FispactProcess(input_data, nuclear_data, output_data, monitor, c_state_ptr, c_callback_pr
deallocate(process_state_native_ptr)

! extract desired outputs

! print heating after irradiation
write(6,*) ""
write(6,*) " TIME(yrs) ", "  HEAT(kW/kg)"
time = 0.0_kr8
do i = 0, FispactOutputDataGetNrOfInventoryEntries(output_data, monitor)-1
    ! timesteps where flux amp = 0
    coolingtime = FispactOutputDataGetInventoryValueByKey(output_data, monitor, &
                            & i, FISPACT_OUTPUT_DATA_INVENTORY_COOL_TIME)
    ! timesteps where flux amp /= 0
    irradtime = FispactOutputDataGetInventoryValueByKey(output_data, monitor, &
                            & i, FISPACT_OUTPUT_DATA_INVENTORY_IRRAD_TIME)
    ! do not print initial inventory heating
    if (coolingtime > 0.0_kr8 .or. irradtime > 0.0_kr8 ) then
        time = time + coolingtime
        ! get total heating from inventory
        total_heat = FispactOutputDataGetInventoryValueByKey(output_data, monitor, &
                            & i, FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_HEAT)
        ! get total mass from inventory
        mass = FispactOutputDataGetInventoryValueByKey(output_data, monitor, &
                            & i, FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_MASS)
        write(6,"(2E12.4)") time/FISPACT_YEAR_TO_SEC, total_heat/mass
    end if
end do

! print dominatants heats and half-life after irradiation
write(6,*) ""
write(6,*) " DOMINANT NUCLIDES AFTER IRRADIATION"
write(6,*) " HALFLIFE(yrs) ", "HEAT(kW/kg)", " NUCLIDE"
! mass at this time step
mass = FispactOutputDataGetInventoryValueByKey(output_data, monitor, &
                        & 1_ki4, FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_MASS)
! number of nuclides in inventory at time step
nrofnuclides = FispactOutputDataGetInventoryNrOfNuclides(output_data, &
                                                & monitor, 1_ki4)
! allocate arrays for dominant zais  and avalues
```

```
    allocate(  dom_heat(0:nrofnuclides-1), dom_zai(0:nrofnuclides-1)   )
    ! extract dominants
    call FispactOutputDataGetInventorySortedByKey(output_data, monitor, 1_ki4, &
                                    & FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_HEAT, &
                                    &nrofnuclides, dom_zai, dom_heat )
    do i = nrofnuclides-21, nrofnuclides-1, 1
        heating = dom_heat(i) / mass
        halflife = extract_halflife_from_nuc(monitor, nuclear_data, dom_zai(i))
        if (halflife /= -1.0_kr8) then
            call FispactGetNuclideName(monitor, dom_zai(i), nuclidename)
            write(6,"(E12.4, 2x,E12.4, 4x, 6A)") halflife/FISPACT_YEAR_TO_SEC, &
                        & heating, nuclidename
        end if
    end do

    deallocate( dom_heat, dom_zai )
    ! tear down
    call FispactOutputDataWrite(output_data, monitor, &
                        & f_c_string(trim(runname)//".json"))

    ! clean up
    call FispactInputDataDestroy(input_data, monitor)
    call FispactOutputDataDestroy(output_data, monitor)
    call FispactNuclearDataDestroy(nuclear_data, monitor)
    call FispactNuclearDataReaderDestroy(nuclear_data_reader, monitor)
    call FispactFinalise(monitor)

    ! write log
    call MonitorWriteToFile(monitor, f_c_string(trim(runname)//".log"))
    call MonitorDestroy(monitor)

    contains

    ! callback for nuclear data reader
    subroutine c_callback_load(c_state) bind(C)
        use, intrinsic :: iso_fortran_env, only : stdout=>output_unit
        type(c_ptr), intent(in), value :: c_state

        type(fispact_nuclear_data_reader_callback_t), pointer :: ptr
        character(:), allocatable :: fkey

        call c_f_pointer(c_state, ptr)

        fkey = trim(c_f_string(ptr%key, FISPACT_MAX_KEY_LENGTH-1))
        if(fkey /= trim(lastkey))then
            write(stdout, '(A)') achar(13)//"  reading "//fkey &
            & //" data from: "// &
            & trim(c_f_string(ptr%path, FISPACT_MAX_PATH_LENGTH-1))//achar(13)
            flush(stdout)
        end if

        ! simple progress bar - could use a better library for this
        write(stdout, '(A3, I4.1, A1, I4.1, A2)', advance='NO') &
        & achar(13)//" [", ptr%index, "/", ptr%total, "]"//achar(13)
        flush(stdout)
        lastkey = fkey
```

```
    end subroutine c_callback_load

    ! callback for fispact compute
    subroutine c_callback_process(c_state) bind(C)
        type(c_ptr), intent(in), value :: c_state

        type(fispact_process_callback_t), pointer :: ptr

        call c_f_pointer(c_state, ptr)
        write(*, '(A2, I2.1, A1, I2.1, A1, A, A, A2)') &
                & " [", ptr%index, "/", ptr%total, "]", &
                & " processing ", trim(c_f_string(ptr%process, FISPACT_MAX_PATH_LENGTH))

    end subroutine c_callback_process

    ! function for extracting half life from nuclear data
    function extract_halflife_from_nuc(monitor, nuclear_data, zai) result(halflife)
        type(c_ptr), intent(in) :: monitor
        type(c_ptr), intent(in) :: nuclear_data
        integer(ki4), intent(in) :: zai

        integer(ki4) :: nrofzais
        integer(ki4) :: i
        real(kr8) :: halflife
        integer(ki4), allocatable :: decay_zais(:)

        ! set default half life value
        halflife = -1.0_kr8
        ! number of nucludes with decay data
        nrofzais = FispactNuclearDataGetDecayDataSize(nuclear_data, monitor)
        ! alloacte arrays
        allocate(decay_zais(0:nrofzais-1))
        ! fill array with zai's from decay data
        call FispactNuclearDataGetDecayZais(nuclear_data, monitor, &
                                    & nrofzais, decay_zais(0:nrofzais-1))
        ! loop over decay zais to find request zai
        do i = 0, nrofzais-1
            if (decay_zais(i) == zai) then
                ! get halflife for found zai
                halflife = FispactNuclearDataGetDecayHalfLife(nuclear_data, monitor, i)
                exit
            end if
        end do
        ! deallocate arrays
        deallocate(decay_zais)
    end function extract_halflife_from_nuc

end program API_example
```

C.1.1 Terminal Output

```
TIME(yrs)   HEAT(kW/kg)
0.0000E+00  0.4324E-06
0.1141E-05  0.3781E-06
0.2757E-05  0.3581E-06
0.4880E-05  0.3382E-06
0.7478E-05  0.3209E-06
0.1055E-04  0.3046E-06
0.1445E-04  0.2789E-06
0.1939E-04  0.2500E-06
0.2548E-04  0.2226E-06
0.3324E-04  0.1885E-06
0.4310E-04  0.1543E-06
0.5504E-04  0.1273E-06
0.7006E-04  0.9702E-07
0.8911E-04  0.6931E-07
0.1121E-03  0.5101E-07
0.1411E-03  0.3386E-07
0.1779E-03  0.2172E-07
0.2224E-03  0.1612E-07
0.2746E-03  0.1296E-07
0.3405E-03  0.1037E-07
0.4255E-03  0.8838E-08
0.5298E-03  0.8041E-08

DOMINANT NUCLIDES AFTER IRRADIATION
HALFLIFE(yrs) HEAT(kW/kg) NUCLIDE
0.1940E+00    0.4584E-10    Co58
0.8683E-06    0.5046E-10    Co63
0.8556E-07    0.1444E-09    Mn58m
0.2066E-05    0.1951E-09    Mn58
0.1038E-02    0.2070E-09    Co58m
0.1882E-03    0.3081E-09    Co61
0.1103E-04    0.5156E-09    Ti51
0.7966E-04    0.5830E-09    Cr49
0.1618E-04    0.6470E-09    Fe53
0.1137E-04    0.7515E-09    Fe61
0.4096E-02    0.1056E-08    Ni57
0.6731E-05    0.1487E-08    Cr55
0.2706E-05    0.1836E-08    Mn57
0.9506E-08    0.2989E-08    Co64
0.1990E-04    0.4456E-08    Co60m
0.2645E-04    0.7237E-08    Co62m
0.2942E-03    0.7290E-08    Mn56
0.1578E-05    0.9169E-08    V54
0.3080E-05    0.2718E-07    V53
0.2928E-05    0.3281E-07    Co62
0.7120E-05    0.3334E-06    V52
```

D C API Examples

D.1 FNS Inconel

Below is the complete FNS Inconel irradiation used as an example throughout this work using the Python FISPACT-II API.

```
// Fispact API FNS Inconel TENDL2019, decay 2020
#include <string.h>
#include <stdio.h>

#include "fispactapi.h"

// callback for nuclear data load
void load_callback(FISPACT_NUCLEAR_DATA_LOAD_CALLBACK_DATA* data){
    printf("\33[2K\r%s: %s [%i/%i]", data->key, data->path, data->index, data->total );
    fflush(stdout);
}

// callback fispact compute process
void process_callback(FISPACT_COMPUTE_CALLBACK_DATA* data){
    printf("\33[2K\r%s: [%i/%i]", data->process, data->index, data->total );
    fflush(stdout);
}

// function to reverse the order of flux
void reverse_range(double* buffer, int left, int right)
{
    while (left < right)
    {
        double temp = buffer[left];
        buffer[left++] = buffer[right];
        buffer[right--] = temp;
    }
}

// function for extracting half life from nuclear data
double extract_halflife_from_nuc(MONITOR_PTR_VALUE monitor,
                                 FISPACT_ND_PTR_VALUE nuclear_data, int zai)
{
    // number of nuclides with decay data
    int nrofzais = FispactNuclearDataGetDecayDataSize(nuclear_data, monitor);
    // define arrays
    int decay_zais[nrofzais];
    // get decay sizes from decay data
    FispactNuclearDataGetDecayZais(nuclear_data, monitor, nrofzais, decay_zais);
    for (int i = 0; i < nrofzais+1; ++i) {
        if (decay_zais[i] == zai) {
            double halflife = FispactNuclearDataGetDecayHalfLife(nuclear_data,
                                                                 monitor, i);
            return halflife;
        }
    }
```

```
    return -1.0;
}


// main program
int main(int argc, const char * argv[]){

    const char runname[] = "FNSInconel_CAPI_tendl19";
    char nuclear_data_path[FISPACT_MAX_PATH_LENGTH];

    // read nuclear data path from command line
    if (argc == 2) {
        strcpy(nuclear_data_path, argv[1]);
    }
    else{
        printf("\nExpect <program> /path/to/nuclear_data. Exiting...\n");
        return 1;
    }

    MONITOR_PTR_VALUE monitor;
    FISPACT_INPUT_PTR_VALUE input_data;
    FISPACT_OUTPUT_PTR_VALUE output_data;
    FISPACT_ND_PTR_VALUE nuclear_data;
    FISPACT_ND_READER_PTR_VALUE nd_reader;

    // initialise required objects
    monitor = MonitorCreate();
    FispactInitialise(monitor);
    input_data = FispactInputDataCreate(monitor);
    output_data = FispactOutputDataCreate(monitor);
    nuclear_data = FispactNuclearDataCreate(monitor);
    nd_reader = FispactNuclearDataReaderCreate(monitor);

    // set the minimum verbosity default to lowest level - trace
    MonitorSetVerbosityLevel(monitor, MONITOR_SEVERITY_TRACE);

    // loading TENDL2019 and Decay2020 alongside hazards and radiological
    // data from Decay2012 files
    // set projectile
    FispactNuclearDataSetProjectile(nuclear_data, monitor,
                                    FISPACT_PROJECTILE_TYPE_NEUTRON)

    // set paths to the nuclear data, same patsh and keys as those
    // found in a FILES file
    char path[FISPACT_MAX_PATH_LENGTH];
    strcpy(path, nuclear_data_path);
    // TENDL2019 load, using the decay2020 index file
    strcat(path, "/decay2020/tendl19_decay2020_index");
    FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_IND_NUC_KEY, path);
    strcpy(path, nuclear_data_path);
    strcat(path, "/TENDL2019data/gendf-1102");
    FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_XS_ENDF_KEY, path);
    strcpy(path, nuclear_data_path);
    strcat(path, "/TENDL2019data/tp-1102-194");
    FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_PROB_TAB_KEY, path);
    // fission yeild data
```

```
strcpy(path, nuclear_data_path);
strcat(path, "/GEFY61data/gefy61_nfy");
FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_FY_ENDF_KEY, path);
strcpy(path, nuclear_data_path);
strcat(path, "/GEFY61data/gefy61_sfy");
FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_SF_ENDF_KEY, path);
// decay2020 decay data
strcpy(path, nuclear_data_path);
strcat(path, "/decay2020/decay_2020");
FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_DK_ENDF_KEY, path);
// attenuation data required for dose rate calculation
strcpy(path, nuclear_data_path);
strcat(path, "/decay/abs_2012");
FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_ABSORP_KEY, path);
//  Hazards and radiological data from decay2012 files
strcpy(path, nuclear_data_path);
strcat(path, "/decay/hazards_2012");
FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_HAZARDS_KEY, path);
strcpy(path, nuclear_data_path);
strcat(path, "/decay/clear_2012");
FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_CLEAR_KEY, path);
strcpy(path, nuclear_data_path);
strcat(path, "/decay/a2_2012");
FispactNuclearDataReaderSetPath(nd_reader, monitor, FISPACT_ND_A2DATA_KEY, path);
// load nuclear data
FISPACT_NUCLEAR_DATA_LOAD_CALLBACK_DATA load_data;
FispactNuclearDataReaderLoad(nd_reader, nuclear_data, monitor,
                                        &load_data, load_callback);

//
// set input data
//
// set name
FispactInputDataSetName(input_data, monitor, runname);
//
// set flux
//
const int in_group_len = 709;
const int out_group_len = 1102;
double flux[709] = {
        0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
        0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
        0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
        0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
        0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
        0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
        0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
        0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
        0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
        0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
        0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
        0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
        0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
        0.0000E+00, 0.0000E+00, 2.9806E+08, 7.2391E+08, 7.3338E+08, 7.4309E+08,
        2.1367E+09, 3.1298E+09, 1.2641E+09, 6.3931E+08, 6.0755E+07, 3.0446E+07,
        2.1633E+07, 1.8685E+07, 1.5579E+07, 1.5817E+07, 1.5557E+07, 1.3784E+07,
```

```
1.2367E+07,  1.1450E+07,  1.0447E+07,  1.0622E+07,  1.0742E+07,  1.0310E+07,
1.0493E+07,  1.0576E+07,  1.0464E+07,  1.0659E+07,  1.0251E+07,  9.9709E+06,
1.0168E+07,  2.2430E+07,  2.0564E+07,  1.9520E+07,  1.8852E+07,  1.8621E+07,
1.8803E+07,  1.9212E+07,  1.9312E+07,  1.8698E+07,  1.6903E+07,  1.6879E+07,
1.7133E+07,  1.6529E+07,  1.6401E+07,  1.6155E+07,  1.6995E+07,  1.7381E+07,
1.6948E+07,  1.6652E+07,  1.6526E+07,  1.6376E+07,  1.6283E+07,  1.6050E+07,
1.5988E+07,  1.5512E+07,  1.5916E+07,  1.5694E+07,  1.5805E+07,  1.5977E+07,
1.5868E+07,  1.5489E+07,  1.5761E+07,  1.5463E+07,  1.5340E+07,  1.5227E+07,
1.5058E+07,  1.5031E+07,  1.4791E+07,  1.4435E+07,  1.4673E+07,  1.4417E+07,
1.4095E+07,  1.4044E+07,  1.3948E+07,  1.3512E+07,  1.3097E+07,  1.2946E+07,
1.2726E+07,  1.2614E+07,  1.2580E+07,  1.1950E+07,  1.1605E+07,  1.1272E+07,
1.0946E+07,  1.0604E+07,  1.0435E+07,  1.0073E+07,  9.7617E+06,  9.5178E+06,
9.0618E+06,  8.7300E+06,  8.4719E+06,  7.9208E+06,  7.8539E+06,  7.5366E+06,
7.2098E+06,  7.1581E+06,  6.6208E+06,  6.3730E+06,  6.2884E+06,  6.0896E+06,
5.8604E+06,  5.7249E+06,  5.6011E+06,  4.9963E+06,  4.9964E+06,  4.7358E+06,
4.5749E+06,  4.4232E+06,  4.3949E+06,  4.0158E+06,  3.6967E+06,  3.5937E+06,
3.4541E+06,  3.3024E+06,  3.3809E+06,  3.3065E+06,  2.9941E+06,  2.7719E+06,
2.3252E+06,  2.4091E+06,  2.3406E+06,  2.5279E+06,  2.5740E+06,  2.4514E+06,
2.1743E+06,  1.9598E+06,  1.9485E+06,  1.8537E+06,  1.8537E+06,  1.6554E+06,
1.5054E+06,  1.5054E+06,  1.6360E+06,  1.6701E+06,  1.6230E+06,  1.6230E+06,
1.4395E+06,  1.3368E+06,  1.2342E+06,  1.2342E+06,  1.2342E+06,  1.1798E+06,
1.1473E+06,  1.0320E+06,  1.0317E+06,  1.0217E+06,  9.9629E+05,  9.9628E+05,
8.8628E+05,  8.0295E+05,  8.0295E+05,  8.0295E+05,  9.7740E+05,  1.0185E+06,
9.4898E+05,  9.4898E+05,  8.8288E+05,  8.1851E+05,  7.9764E+05,  7.8468E+05,
7.8355E+05,  7.3070E+05,  6.9329E+05,  6.9319E+05,  6.8316E+05,  6.5778E+05,
6.5778E+05,  6.5778E+05,  6.5778E+05,  6.5778E+05,  5.7473E+05,  5.6058E+05,
5.6057E+05,  5.6058E+05,  5.6057E+05,  5.5658E+05,  5.5123E+05,  5.4074E+05,
5.1003E+05,  5.1002E+05,  5.1002E+05,  4.7404E+05,  4.7393E+05,  4.7393E+05,
4.7393E+05,  4.7393E+05,  4.3415E+05,  4.0394E+05,  4.0393E+05,  4.0393E+05,
4.0394E+05,  4.0163E+05,  3.8742E+05,  3.8742E+05,  3.8742E+05,  3.8742E+05,
3.8742E+05,  3.8988E+05,  3.9088E+05,  3.9088E+05,  3.8714E+05,  3.8264E+05,
3.6002E+05,  3.0248E+05,  3.0621E+05,  3.3614E+05,  3.3614E+05,  3.1955E+05,
3.3173E+05,  3.2173E+05,  3.1802E+05,  3.0931E+05,  3.0342E+05,  2.9074E+05,
2.7357E+05,  2.7357E+05,  2.7357E+05,  2.7357E+05,  2.7357E+05,  2.3473E+05,
2.3458E+05,  2.3458E+05,  2.3458E+05,  2.3459E+05,  2.2167E+05,  2.1183E+05,
2.1184E+05,  2.1183E+05,  2.1184E+05,  2.0913E+05,  1.9234E+05,  1.9233E+05,
1.9234E+05,  1.9234E+05,  1.9233E+05,  1.7840E+05,  1.7271E+05,  1.7271E+05,
1.7271E+05,  1.7271E+05,  1.6986E+05,  1.6259E+05,  1.6259E+05,  1.6259E+05,
1.6259E+05,  1.6259E+05,  1.6342E+05,  1.6356E+05,  1.6356E+05,  1.6356E+05,
1.6356E+05,  1.5175E+05,  1.3570E+05,  1.3570E+05,  1.3570E+05,  1.3570E+05,
1.3570E+05,  1.3554E+05,  1.3554E+05,  1.3554E+05,  1.3554E+05,  1.3554E+05,
1.3184E+05,  1.2902E+05,  1.2902E+05,  1.2902E+05,  1.2902E+05,  1.2699E+05,
1.1435E+05,  1.1435E+05,  1.1435E+05,  1.1435E+05,  1.1435E+05,  1.0523E+05,
1.0150E+05,  1.0149E+05,  1.0150E+05,  1.0149E+05,  9.5928E+04,  8.1656E+04,
8.1657E+04,  8.1656E+04,  8.1656E+04,  8.1656E+04,  9.2951E+04,  9.4913E+04,
9.4913E+04,  9.4913E+04,  9.4913E+04,  8.6281E+04,  7.4521E+04,  7.4521E+04,
7.4521E+04,  7.4521E+04,  7.4521E+04,  6.1552E+04,  6.1482E+04,  6.1482E+04,
6.1482E+04,  6.1482E+04,  5.6493E+04,  5.2666E+04,  5.2666E+04,  5.2666E+04,
5.2666E+04,  5.4200E+04,  6.3842E+04,  6.3842E+04,  6.3842E+04,  6.3841E+04,
6.3841E+04,  5.7042E+04,  5.4245E+04,  5.4245E+04,  5.4245E+04,  5.4245E+04,
5.3570E+04,  5.1834E+04,  5.1834E+04,  5.1834E+04,  5.1834E+04,  5.1834E+04,
4.8436E+04,  4.7843E+04,  4.7842E+04,  4.7842E+04,  4.7843E+04,  4.5208E+04,
4.1608E+04,  4.1608E+04,  4.1608E+04,  4.1608E+04,  4.1608E+04,  3.7355E+04,
3.7329E+04,  3.7329E+04,  3.7329E+04,  3.7329E+04,  3.8387E+04,  3.9202E+04,
3.9202E+04,  3.9201E+04,  3.9202E+04,  3.9728E+04,  4.3059E+04,  4.3059E+04,
4.3059E+04,  4.3059E+04,  4.3059E+04,  2.9553E+04,  2.3978E+04,  2.3978E+04,
```

```
2.3978E+04, 2.3978E+04, 2.5107E+04, 2.8026E+04, 2.8026E+04, 2.8026E+04,
2.8026E+04, 2.8025E+04, 2.2712E+04, 2.1778E+04, 2.1778E+04, 2.1778E+04,
2.1778E+04, 2.2753E+04, 2.4090E+04, 2.4090E+04, 2.4090E+04, 2.4090E+04,
2.4090E+04, 1.8954E+04, 1.8918E+04, 1.8918E+04, 1.8918E+04, 1.8918E+04,
1.7435E+04, 1.6289E+04, 1.6289E+04, 1.6289E+04, 1.6289E+04, 1.6196E+04,
1.5601E+04, 1.5601E+04, 1.5601E+04, 1.5601E+04, 1.5601E+04, 1.4146E+04,
1.3543E+04, 1.3543E+04, 1.3543E+04, 1.3543E+04, 1.3819E+04, 1.4535E+04,
1.4535E+04, 1.4535E+04, 1.4535E+04, 1.4535E+04, 1.4109E+04, 1.4033E+04,
1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04,
1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04,
1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04,
1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4034E+04,
1.4033E+04, 1.4033E+04, 1.4034E+04, 1.4033E+04, 1.4033E+04, 1.1922E+04,
1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04,
1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04,
1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04,
1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04,
1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04,
1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04,
1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04,
1.1922E+04, 1.1923E+04, 1.1922E+04, 1.1922E+04, 1.1923E+04, 1.1922E+04,
1.1922E+04, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00 };

// reverse the in bounds to get ascending order
double inbounds[710];
memcpy(inbounds, FISPACT_GROUP_709, (in_group_len+1)*sizeof(double));
reverse_range(inbounds, 0, in_group_len);
// reverse flux to ascending order
reverse_range(flux, 0, (in_group_len-1) );

// reverse the out bounds to get ascending order
```

```
double outflux[1102];
double outbounds[1103];
memcpy(outbounds, FISPACT_GROUP_1102, (out_group_len+1)*sizeof(double));
reverse_range(outbounds, 0, out_group_len);

// perfom group convert and set flux
FispactGroupConvertByLethargy(monitor, in_group_len,  inbounds,
                              flux, out_group_len, outbounds,
                              outflux);
FispactInputDataSetFlux(input_data, monitor, out_group_len, outbounds, outflux);
FispactInputDataSetFluxWallLoading(input_data, monitor, 1.0);
FispactInputDataSetFluxName(input_data, monitor, "709 dummy flux");
//
// set material details
//
// set density (g/cm3)
FispactInputDataSetDensity(input_data, monitor,  8.42);
// set atoms threshold, MIND keyword
FispactInputDataSetAtomsThreshold(input_data, monitor, 1.0e3);
// set total mass (kg)
FispactInputDataSetMassTotal(input_data, monitor, 1.0E-3);
// set material my weight percentage, MASS keyword
// atomic numbers for elements to be included, using utilities
// to convert from element names
int atomic_numbers[4] = { FispactGetAtomicNumberFromElementName(monitor, "Ni"),
                    FispactGetAtomicNumberFromElementName(monitor, "Mn"),
                    FispactGetAtomicNumberFromElementName(monitor, "Fe"),
                    FispactGetAtomicNumberFromElementName(monitor, "Cr") };
// percentge of each element
double percent[4] = { 75.82, 0.39, 7.82, 15.97 };
// input materials
FispactInputDataSetMass(input_data, monitor, 4, atomic_numbers, percent);
//
// set irradiation schedule
//
// 5 min irradiation
double irradiationtime[1] = {5.0*FISPACT_MIN_TO_SEC};
double fluxamp[1] = {1.116E+10};
FispactInputDataSetSchedule(input_data, monitor, 1, irradiationtime, fluxamp);
// cooling steps
double cooltimes[21] = {36.0, 15.0, 16.0, 15.0, 15.0, 26.0, 33.0, 36.0,
            53.0, 66.0, 66.0, 97.0, 127.0, 126.0, 187.0, 246.0,
            244.0, 246.0, 428.0, 606.0, 607.0 };

for (int i = 0; i < 20; ++i) {
     FispactInputDataAppendSchedule(input_data, monitor, cooltimes[i], 0.0);
}

// run fispact
FISPACT_COMPUTE_CALLBACK_DATA compute_data;
FispactProcess(input_data, nuclear_data, output_data,
                    monitor, &compute_data, process_callback);

//
// extract desired outputs
//
```

```
// heating after irradiation
printf("\n\n");
printf( "  TIME(yrs)    HEAT(kW/kg) \n" );
double time = 0.0;
int nrofentries = FispactOutputDataGetNrOfInventoryEntries(output_data, monitor);
for (int i = 0; i < nrofentries+1; ++i) {
    // timesteps where flux amp = 0
    double coolingtime = FispactOutputDataGetInventoryValueByKey(output_data,
                    monitor, i, FISPACT_OUTPUT_DATA_INVENTORY_COOL_TIME);
    // timesteps where flux amp /= 0
    double irradtime = FispactOutputDataGetInventoryValueByKey(output_data,
                   monitor, i, FISPACT_OUTPUT_DATA_INVENTORY_IRRAD_TIME);
    // do not print initial inventory heating
    if ( (coolingtime > 0.0) || (irradtime > 0.0) ) {
        time += coolingtime/FISPACT_YEAR_TO_SEC;
        // get total heta from inventory
        double total_heat = FispactOutputDataGetInventoryValueByKey(output_data,
                monitor, i, FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_HEAT);
        // get total mass from inventory
        double mass = FispactOutputDataGetInventoryValueByKey(output_data, monitor,
                i, FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_MASS);
        double heatpermass = total_heat/mass;
        printf("%12.4E %12.4E \n", time, heatpermass);
    }
}

// print dominants heats and halflife after irradiation
printf("\n\n");
printf("  DOMINANT NUCLIDES AFTER IRRADIATION\n" );
printf( "  TIME(yrs)     HEAT(kW/kg) NUCLIDE\n" );
//  mass at this timestep
double mass = FispactOutputDataGetInventoryValueByKey(output_data, monitor,
                   1, FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_MASS);
// number of nuclides at his timestep
int nrofnuclides = FispactOutputDataGetInventoryNrOfNuclides(output_data,
                                                              monitor, 1);
// define arrays
double dom_heat[nrofnuclides];
int dom_zai[nrofnuclides];
// extract dominants
FispactOutputDataGetInventorySortedByKey(output_data, monitor, 1,
                                FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_HEAT,
                                nrofnuclides, dom_zai, dom_heat );

for (int i = (nrofnuclides-21); i < (nrofnuclides); ++i) {
    double heating = dom_heat[i] / mass;
    double halflife = extract_halflife_from_nuc(monitor, nuclear_data, dom_zai[i]);
    if (halflife != -1.0) {
        char nuclidename[FISPACT_NUCLIDE_NAME_LENGTH];
        FispactGetNuclideName(monitor, dom_zai[i], nuclidename);
        printf( "%12.4E %12.4E %6s \n", halflife/FISPACT_YEAR_TO_SEC,
                heating, nuclidename);
    }
}

// write output JSON
```

```
    char output_filename[128];
    strcpy(output_filename, runname);
    strcat(output_filename, ".json");
    FispactOutputDataWrite(output_data, monitor, output_filename);

    // clean up
    FispactInputDataDestroy(input_data, monitor);
    FispactOutputDataDestroy(output_data, monitor);
    FispactNuclearDataDestroy(nuclear_data, monitor);
    FispactNuclearDataReaderDestroy(nd_reader, monitor);

    char log_filename[128];
    strcpy(log_filename, runname);
    strcat(log_filename, ".log");
    MonitorWriteToFile(monitor, log_filename);

    FispactFinalise(monitor);
    MonitorDestroy(monitor);

    return 0;
}
```

D.1.1 Terminal Output

```
  TIME(yrs)   HEAT(kW/kg)
  0.0000E+00   4.3240E-07
  1.1408E-06   3.7809E-07
  2.7569E-06   3.5810E-07
  4.8800E-06   3.3822E-07
  7.4784E-06   3.2086E-07
  1.0552E-05   3.0463E-07
  1.4450E-05   2.7889E-07
  1.9393E-05   2.5003E-07
  2.5477E-05   2.2259E-07
  3.3241E-05   1.8847E-07
  4.3096E-05   1.5428E-07
  5.5042E-05   1.2732E-07
  7.0062E-05   9.7018E-08
  8.9107E-05   6.9310E-08
  1.1214E-04   5.1012E-08
  1.4111E-04   3.3858E-08
  1.7787E-04   2.1722E-08
  2.2236E-04   1.6125E-08
  2.7464E-04   1.2959E-08
  3.4049E-04   1.0370E-08
  4.2554E-04   8.8376E-09


  DOMINANT NUCLIDES AFTER IRRADIATION
  TIME(yrs)    HEAT(kW/kg) NUCLIDE
  1.9398E-01   4.5842E-11 Co58
  8.6825E-07   5.0456E-11 Co63
```

```
  8.5558E-08   1.4440E-10 Mn58m
  2.0661E-06   1.9508E-10 Mn58
  1.0381E-03   2.0703E-10 Co58m
  1.8823E-04   3.0807E-10 Co61
  1.1027E-05   5.1556E-10 Ti51
  7.9664E-05   5.8303E-10 Cr49
  1.6180E-05   6.4704E-10 Fe53
  1.1370E-05   7.5152E-10 Fe61
  4.0958E-03   1.0561E-09 Ni57
  6.7305E-06   1.4868E-09 Cr55
  2.7062E-06   1.8364E-09 Mn57
  9.5064E-09   2.9888E-09 Co64
  1.9901E-05   4.4556E-09 Co60m
  2.6447E-05   7.2372E-09 Co62m
  2.9418E-04   7.2896E-09 Mn56
  1.5781E-06   9.1686E-09 V54
  3.0801E-06   2.7177E-08 V53
  2.9280E-06   3.2810E-08 Co62
  7.1203E-06   3.3343E-07 V52
```

E C++ API Examples

E.1 FNS Inconel

Below is the complete FNS Inconel irradiation used as an example throughout this work using the Python FISPACT-II API.

```
// Fispact API FNS Inconel TENDL2019, decay 2020
#include <iostream>
#include <iomanip>
#include <fstream>
#include <vector>
#include <string>
#include <chrono>
#include <thread>
#include <math.h>

#include "common.hpp"
#include "monitor.hpp"
#include "exceptions.hpp"

#include "fispactnucleardata.hpp"
#include "fispactinputdata.hpp"
#include "fispactoutputdata.hpp"
#include "fispactcompute.hpp"
#include "fispactgroupstructures.hpp"
#include "fispactgroupconvert.hpp"
#include "fispactelementaldata.hpp"
#include "fispactutil.hpp"

namespace fp = fispact;
```

```
// nuclear data load callback
void load_callback(std::string key, std::string path, int i, int t){
    std::cout << "\33[2K\r" << key << ": " << path << " [" << i << "/" << t << "]" << std::flush;
}

// fispact compute process callback
void process_callback(std::string process_name, int i, int t){
    std::cout << "\33[2K\r [" << i << "/" << t << "] " << process_name << std::flush;
}

// function for extracting half life from nuclear data
double extract_halflife_from_nuc( const fp::NuclearData& nd, int zai){
    // number of nuclides with decay data
    int nrofzais = nd.getDecayDataSize();
    // get decay sizes from decay data
    std::vector<int> decay_zais = nd.getDecayZais();
    for (int i = 0; i < nrofzais+1; ++i) {
        if (decay_zais[i] == zai) {
            double halflife = nd.getDecayHalfLife(i);
            return halflife;
        }
    }
    return -1.0;
}


// main prpgram
int main(int argc, const char * argv[]){

    const std::string runname = "FNSInconel_C++API_tendl19";
    std::string nd_base_path = "";

    // read nuclear data path from command line
    if (argc == 2) {
        nd_base_path = argv[1];
    }
    else{
        std::cout << "Expect <program> /path/to/nuclear_data. Exiting..." << std::endl;
        return 1;
    }


    fp::FispactMonitor monitor(runname + ".log");
    fp::FispactMonitor::CMonitor& mpp = monitor.native(); // c++ native type

    // initialise required objects
    // fispact data types
    fp::NuclearData nd(monitor);
    fp::io::NuclearDataReader nd_reader(monitor);
    fp::InputData input(monitor);
    fp::OutputData output(monitor);

    // set the minimum verbosity default to lowest level - trace
    mpp.setVerbosityLevel(fp::severity::level::trace);
    // initialise gloabl data
```

```
fp::GlobalInitialise(monitor);

// loading TENDL2019 and Decay2020 alongside hazards and radiological
// data from Decay2012 files
// set projectile
nd.setProjectile(FISPACT_PROJECTILE_TYPE_NEUTRON);
// set paths to the nuclear data, same patsh and keys as those
// found in a FILES file
// TENDL2019 load, using the decay2020 index file
nd_reader.setPath(FISPACT_ND_IND_NUC_KEY, nd_base_path +
                                    "/decay2020/tendl19_decay2020_index");
nd_reader.setPath(FISPACT_ND_XS_ENDF_KEY, nd_base_path +
                                    "/TENDL2019data/gendf-1102");
nd_reader.setPath(FISPACT_ND_PROB_TAB_KEY, nd_base_path +
                                    "/TENDL2019data/tp-1102-194");
// fission yeild data
nd_reader.setPath(FISPACT_ND_FY_ENDF_KEY, nd_base_path + "/GEFY61data/gefy61_nfy");
nd_reader.setPath(FISPACT_ND_SF_ENDF_KEY, nd_base_path + "/GEFY61data/gefy61_sfy");
// decay2020 decay data
nd_reader.setPath(FISPACT_ND_DK_ENDF_KEY, nd_base_path + "/decay2020/decay_2020");
// attenuation data required for dose rate calculation
nd_reader.setPath(FISPACT_ND_ABSORP_KEY, nd_base_path + "/decay/abs_2012");
//  Hazards and radiological data from decay2012 files
nd_reader.setPath(FISPACT_ND_HAZARDS_KEY, nd_base_path + "/decay/hazards_2012");
nd_reader.setPath(FISPACT_ND_CLEAR_KEY, nd_base_path + "/decay/clear_2012");
nd_reader.setPath(FISPACT_ND_A2DATA_KEY, nd_base_path + "/decay/a2_2012");
// load nuclear data
nd_reader.load(nd, &load_callback);
//
// set input data
//
// set name
input.setName(runname);
//
// set flux
//
// flux in decending order
std::vector<double> flux = {
    0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
    0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
    0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
    0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
    0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
    0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
    0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
    0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
    0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
    0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
    0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
    0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
    0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
    0.0000E+00, 0.0000E+00, 2.9806E+08, 7.2391E+08, 7.3338E+08, 7.4309E+08,
    2.1367E+09, 3.1298E+09, 1.2641E+09, 6.3931E+08, 6.0755E+07, 3.0446E+07,
    2.1633E+07, 1.8685E+07, 1.5579E+07, 1.5817E+07, 1.5557E+07, 1.3784E+07,
    1.2367E+07, 1.1450E+07, 1.0447E+07, 1.0622E+07, 1.0742E+07, 1.0310E+07,
    1.0493E+07, 1.0576E+07, 1.0464E+07, 1.0659E+07, 1.0251E+07, 9.9709E+06,
```

```
1.0168E+07,  2.2430E+07,  2.0564E+07,  1.9520E+07,  1.8852E+07,  1.8621E+07,
1.8803E+07,  1.9212E+07,  1.9312E+07,  1.8698E+07,  1.6903E+07,  1.6879E+07,
1.7133E+07,  1.6529E+07,  1.6401E+07,  1.6155E+07,  1.6995E+07,  1.7381E+07,
1.6948E+07,  1.6652E+07,  1.6526E+07,  1.6376E+07,  1.6283E+07,  1.6050E+07,
1.5988E+07,  1.5512E+07,  1.5916E+07,  1.5694E+07,  1.5805E+07,  1.5977E+07,
1.5868E+07,  1.5489E+07,  1.5761E+07,  1.5463E+07,  1.5340E+07,  1.5227E+07,
1.5058E+07,  1.5031E+07,  1.4791E+07,  1.4435E+07,  1.4673E+07,  1.4417E+07,
1.4095E+07,  1.4044E+07,  1.3948E+07,  1.3512E+07,  1.3097E+07,  1.2946E+07,
1.2726E+07,  1.2614E+07,  1.2580E+07,  1.1950E+07,  1.1605E+07,  1.1272E+07,
1.0946E+07,  1.0604E+07,  1.0435E+07,  1.0073E+07,  9.7617E+06,  9.5178E+06,
9.0618E+06,  8.7300E+06,  8.4719E+06,  7.9208E+06,  7.8539E+06,  7.5366E+06,
7.2098E+06,  7.1581E+06,  6.6208E+06,  6.3730E+06,  6.2884E+06,  6.0896E+06,
5.8604E+06,  5.7249E+06,  5.6011E+06,  4.9963E+06,  4.9964E+06,  4.7358E+06,
4.5749E+06,  4.4232E+06,  4.3949E+06,  4.0158E+06,  3.6967E+06,  3.5937E+06,
3.4541E+06,  3.3024E+06,  3.3809E+06,  3.3065E+06,  2.9941E+06,  2.7719E+06,
2.3252E+06,  2.4091E+06,  2.3406E+06,  2.5279E+06,  2.5740E+06,  2.4514E+06,
2.1743E+06,  1.9598E+06,  1.9485E+06,  1.8537E+06,  1.8537E+06,  1.6554E+06,
1.5054E+06,  1.5054E+06,  1.6360E+06,  1.6701E+06,  1.6230E+06,  1.6230E+06,
1.4395E+06,  1.3368E+06,  1.2342E+06,  1.2342E+06,  1.2342E+06,  1.1798E+06,
1.1473E+06,  1.0320E+06,  1.0317E+06,  1.0217E+06,  9.9629E+05,  9.9628E+05,
8.8628E+05,  8.0295E+05,  8.0295E+05,  8.0295E+05,  9.7740E+05,  1.0185E+06,
9.4898E+05,  9.4898E+05,  8.8288E+05,  8.1851E+05,  7.9764E+05,  7.8468E+05,
7.8355E+05,  7.3070E+05,  6.9329E+05,  6.9319E+05,  6.8316E+05,  6.5778E+05,
6.5778E+05,  6.5778E+05,  6.5778E+05,  6.5778E+05,  5.7473E+05,  5.6058E+05,
5.6057E+05,  5.6058E+05,  5.6057E+05,  5.5658E+05,  5.5123E+05,  5.4074E+05,
5.1003E+05,  5.1002E+05,  5.1002E+05,  4.7404E+05,  4.7393E+05,  4.7393E+05,
4.7393E+05,  4.7393E+05,  4.3415E+05,  4.0394E+05,  4.0393E+05,  4.0393E+05,
4.0394E+05,  4.0163E+05,  3.8742E+05,  3.8742E+05,  3.8742E+05,  3.8742E+05,
3.8742E+05,  3.8988E+05,  3.9088E+05,  3.9088E+05,  3.8714E+05,  3.8264E+05,
3.6002E+05,  3.0248E+05,  3.0621E+05,  3.3614E+05,  3.3614E+05,  3.1955E+05,
3.3173E+05,  3.2173E+05,  3.1802E+05,  3.0931E+05,  3.0342E+05,  2.9074E+05,
2.7357E+05,  2.7357E+05,  2.7357E+05,  2.7357E+05,  2.7357E+05,  2.3473E+05,
2.3458E+05,  2.3458E+05,  2.3458E+05,  2.3459E+05,  2.2167E+05,  2.1183E+05,
2.1184E+05,  2.1183E+05,  2.1184E+05,  2.0913E+05,  1.9234E+05,  1.9233E+05,
1.9234E+05,  1.9234E+05,  1.9233E+05,  1.7840E+05,  1.7271E+05,  1.7271E+05,
1.7271E+05,  1.7271E+05,  1.6986E+05,  1.6259E+05,  1.6259E+05,  1.6259E+05,
1.6259E+05,  1.6259E+05,  1.6342E+05,  1.6356E+05,  1.6356E+05,  1.6356E+05,
1.6356E+05,  1.5175E+05,  1.3570E+05,  1.3570E+05,  1.3570E+05,  1.3570E+05,
1.3570E+05,  1.3554E+05,  1.3554E+05,  1.3554E+05,  1.3554E+05,  1.3554E+05,
1.3184E+05,  1.2902E+05,  1.2902E+05,  1.2902E+05,  1.2902E+05,  1.2699E+05,
1.1435E+05,  1.1435E+05,  1.1435E+05,  1.1435E+05,  1.1435E+05,  1.0523E+05,
1.0150E+05,  1.0149E+05,  1.0150E+05,  1.0149E+05,  9.5928E+04,  8.1656E+04,
8.1657E+04,  8.1656E+04,  8.1656E+04,  8.1656E+04,  9.2951E+04,  9.4913E+04,
9.4913E+04,  9.4913E+04,  9.4913E+04,  8.6281E+04,  7.4521E+04,  7.4521E+04,
7.4521E+04,  7.4521E+04,  7.4521E+04,  6.1552E+04,  6.1482E+04,  6.1482E+04,
6.1482E+04,  6.1482E+04,  5.6493E+04,  5.2666E+04,  5.2666E+04,  5.2666E+04,
5.2666E+04,  5.4200E+04,  6.3842E+04,  6.3842E+04,  6.3842E+04,  6.3841E+04,
6.3841E+04,  5.7042E+04,  5.4245E+04,  5.4245E+04,  5.4245E+04,  5.4245E+04,
5.3570E+04,  5.1834E+04,  5.1834E+04,  5.1834E+04,  5.1834E+04,  5.1834E+04,
4.8436E+04,  4.7843E+04,  4.7842E+04,  4.7842E+04,  4.7843E+04,  4.5208E+04,
4.1608E+04,  4.1608E+04,  4.1608E+04,  4.1608E+04,  4.1608E+04,  3.7355E+04,
3.7329E+04,  3.7329E+04,  3.7329E+04,  3.7329E+04,  3.8387E+04,  3.9202E+04,
3.9202E+04,  3.9201E+04,  3.9202E+04,  3.9728E+04,  4.3059E+04,  4.3059E+04,
4.3059E+04,  4.3059E+04,  4.3059E+04,  2.9553E+04,  2.3978E+04,  2.3978E+04,
2.3978E+04,  2.3978E+04,  2.5107E+04,  2.8026E+04,  2.8026E+04,  2.8026E+04,
2.8026E+04,  2.8025E+04,  2.2712E+04,  2.1778E+04,  2.1778E+04,  2.1778E+04,
```

```
    2.1778E+04,  2.2753E+04,  2.4090E+04,  2.4090E+04,  2.4090E+04,  2.4090E+04,
    2.4090E+04,  1.8954E+04,  1.8918E+04,  1.8918E+04,  1.8918E+04,  1.8918E+04,
    1.7435E+04,  1.6289E+04,  1.6289E+04,  1.6289E+04,  1.6289E+04,  1.6196E+04,
    1.5601E+04,  1.5601E+04,  1.5601E+04,  1.5601E+04,  1.5601E+04,  1.4146E+04,
    1.3543E+04,  1.3543E+04,  1.3543E+04,  1.3543E+04,  1.3819E+04,  1.4535E+04,
    1.4535E+04,  1.4535E+04,  1.4535E+04,  1.4535E+04,  1.4109E+04,  1.4033E+04,
    1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,
    1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,
    1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,
    1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4033E+04,  1.4034E+04,
    1.4033E+04,  1.4033E+04,  1.4034E+04,  1.4033E+04,  1.4033E+04,  1.1922E+04,
    1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,
    1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,
    1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,
    1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,
    1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,
    1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,
    1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,  1.1922E+04,
    1.1922E+04,  1.1923E+04,  1.1922E+04,  1.1922E+04,  1.1923E+04,  1.1922E+04,
    1.1922E+04,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
    0.0000E+00 };

// perfom group convert and set flux
reverse(flux.begin(), flux.end());
std::vector<double> outflux = fp::groupconvert::GroupConvertByLethargy(monitor,
                                fp::groups::G709(), flux, fp::groups::G1102() );
input.setFlux(fp::groups::G1102(), outflux);
input.setFluxWallLoading(1.0);
input.setFluxName("709 dummy flux");
//
// set material details
//
```

```
    // set density (g/cm3)
    input.setDensity(8.42);
    // set atoms threshold, MIND keyword
    input.setAtomsThreshold(1.0e3);
    // set total mass (kg)
    input.setMassTotal(1.0E-3);
    // set material my weight percentage, MASS keyword
    // atomic numbers for elements to be included, using utilities
    // to convert from element names
    std::vector<int> atomic_numbers = {
                        fp::util::GetAtomicNumberFromElementName(monitor, "Ni"),
                        fp::util::GetAtomicNumberFromElementName(monitor, "Mn"),
                        fp::util::GetAtomicNumberFromElementName(monitor, "Fe"),
                        fp::util::GetAtomicNumberFromElementName(monitor, "Cr") };
    // percentge of each element
    std::vector<double>  percent = { 75.82, 0.39, 7.82, 15.97 };
    // input materials
    input.setMass(atomic_numbers, percent);
    //
    // set irradiation schedule
    //
    // 5 min irradiation
    std::vector<double> irradiationtime = {5.0*FISPACT_MIN_TO_SEC};
    std::vector<double> fluxamp = {1.116E+10};
    input.setSchedule(irradiationtime, fluxamp);
    // cooling steps
    std::vector<double> cooltimes = {36.0, 15.0, 16.0, 15.0, 15.0, 26.0, 33.0, 36.0,
                53.0, 66.0, 66.0, 97.0, 127.0, 126.0, 187.0, 246.0,
                244.0, 246.0, 428.0, 606.0, 607.0 };

    for (int i = 0; i < 20; ++i) {
        input.appendSchedule(cooltimes[i], 0.0);
    }

    // run fispact
    fp::Process(input, nd, output, monitor, process_callback);

    //
    // extract desired outputs
    //
    // heating after irradiation
    double time = 0.0;
    std::cout <<  "  \n\n";
    std::cout <<  std::setw(15) << "TIME(yrs)" << std::setw(16) << "HEAT(kW/kg)\n";
    for(int i=0; i< output.getNrOfInventoryEntries(); ++i){
        // timesteps where flux amp = 0
        double coolingtime = output.getInventoryValue( i,
                                        FISPACT_OUTPUT_DATA_INVENTORY_COOL_TIME);
        // timesteps where flux amp /= 0
        double irradtime = output.getInventoryValue(i,
                                        FISPACT_OUTPUT_DATA_INVENTORY_IRRAD_TIME);
        // do not print initial inventory heating
        if ( (coolingtime > 0.0) || (irradtime > 0.0) ) {
            time += coolingtime/FISPACT_YEAR_TO_SEC;
            // get total heta from inventory
            double total_heat = output.getInventoryValue( i,
```

```
                                                FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_HEAT);
            // get total mass from inventory
            double mass = output.getInventoryValue(i,
                                                    FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_MASS);
            std::cout << std::setw(15) << time << std::setw(15)
                      << total_heat/mass << "\n";
        }
    }

    // print dominants heats and halflife after irradiation
    std::cout <<"\n\n";
    std::cout <<"  DOMINANT NUCLIDES AFTER IRRADIATION\n";
    std::cout << std::setw(15) << "TIME(yrs)" << std::setw(15) << "HEAT(kW/kg)"
              << std::setw(16) <<  "NUCLIDE\n";
    //  mass at this timestep
    double mass = output.getInventoryValue(1,
                                FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_MASS);
    // sort inventory by total heat
    std::pair<std::vector<int>, std::vector<double>>  dom_sort =
            output.getSortedInventory(1, FISPACT_OUTPUT_DATA_INVENTORY_TOTAL_HEAT);
    std::vector<int> dom_zai = std::get<0>(dom_sort);
    std::vector<double> dom_heat = std::get<1>(dom_sort);
    // number of nuclides at his timestep
    int nrofnuclides = dom_zai.size();
    for (int i = (nrofnuclides-21); i < nrofnuclides; ++i) {
        double heating = dom_heat[i] / mass;
        double halflife = extract_halflife_from_nuc(nd, dom_zai[i]);
        if (halflife != -1.0) {
            std::string nuclidename = fp::util::GetNuclideName(monitor, dom_zai[i] );
            std::cout << std::setw(15) << halflife/FISPACT_YEAR_TO_SEC
                      << std::setw(15) << heating
                      << std::setw(15) << nuclidename << "\n";
        }
    }

    // write output JSON
    fp::io::ToFile(output, monitor, runname + ".json");
    // clean up
    fp::GlobalFinalise(monitor);
    fp::io::ToFile(monitor, runname + ".log");

    return 0;
}
```

E.1.1 Terminal Output

```
      TIME(yrs)    HEAT(kW/kg)
              0    4.32396e-07
    1.14077e-06    3.78095e-07
    2.75686e-06    3.58104e-07
    4.87997e-06    3.38219e-07
    7.47839e-06    3.20865e-07
```

```
   1.05521e-05    3.04631e-07
   1.44498e-05    2.78894e-07
   1.93931e-05    2.50031e-07
   2.54772e-05    2.22594e-07
   3.32408e-05    1.88468e-07
   4.30958e-05    1.54284e-07
   5.50422e-05    1.27318e-07
   7.00624e-05    9.70175e-08
   8.91069e-05    6.93101e-08
   0.000112144    5.10117e-08
   0.000141107    3.38579e-08
   0.000177865    2.17225e-08
   0.000222355    1.61247e-08
   0.000274641    1.29591e-08
   0.000340489    1.03704e-08
   0.000425539    8.83759e-09


 DOMINANT NUCLIDES AFTER IRRADIATION
     TIME(yrs)     HEAT(kW/kg)         NUCLIDE
      0.193977    4.58415e-11            Co58
   8.68254e-07    5.04559e-11            Co63
   8.55578e-08    1.44403e-10          Mn58m
   2.06606e-06     1.9508e-10            Mn58
     0.0010381    2.07031e-10          Co58m
   0.000188227    3.08073e-10            Co61
   1.10275e-05    5.15555e-10            Ti51
   7.96639e-05    5.83034e-10            Cr49
   1.61799e-05    6.47039e-10            Fe53
   1.13697e-05    7.51516e-10            Fe61
    0.00409581    1.05612e-09            Ni57
   6.73055e-06    1.48678e-09            Cr55
   2.70616e-06    1.83645e-09            Mn57
   9.50643e-09    2.98881e-09            Co64
   1.99008e-05     4.4556e-09          Co60m
   2.64469e-05    7.23717e-09          Co62m
    0.00029418    7.28958e-09            Mn56
   1.57807e-06    9.16856e-09             V54
   3.08008e-06     2.7177e-08             V53
   2.92798e-06    3.28096e-08            Co62
   7.12031e-06    3.33425e-07             V52
```

F Python API Examples

F.1 FNS Inconel

Below is the complete FNS Inconel irradiation used as an example throughout this work using the Python FISPACT-II API.

```
# Fispact API FNS Inconel TENDL2019, decay 2020
import pyfispact as pf
```

```
import sys
# if pyfispact not on PYTHONPATH can use
# sys.path.append(path/to/pyfispact)
import os
import argparse

# define Monitor, error handling object
m = pf.Monitor()

# read command line args
def readargs():
    # read path/to/nuclear data from command line
    parser = argparse.ArgumentParser(description='Process file arguments.')
    parser.add_argument("-n", "--nucleardata", type=str, default="./", help='Point to base nuclear
    args = parser.parse_args()
    nd_base_path = args.nucleardata
    return nd_base_path

# load nuclear data
def load_nuclear_data(nd_base_path):
    # loading TENDL2019 and Decay2020 alongside hazards and radiological
    # data from Decay2012 files
    # initialise nuclear data object
    nd = pf.NuclearData(m)
    # initialise nuclear data reader object
    ndr = pf.io.NuclearDataReader(m)
    #  set projectile to neutron
    nd.setprojectile(pf.PROJECTILE_NEUTRON())
    # set paths to the nuclear data, same patsh and keys as those
    # found in a FILES file
    # TENDL2019 load, using the decay2020 index file
    ndr.setpath(pf.io.ND_IND_NUC_KEY(),   os.path.join(nd_base_path,
                'decay2020', 'tendl19_decay2020_index'))
    ndr.setpath(pf.io.ND_XS_ENDF_KEY(),   os.path.join(nd_base_path,
                'TENDL2019data', 'gendf-1102'))
    ndr.setpath(pf.io.ND_PROB_TAB_KEY(), os.path.join(nd_base_path,
                'TENDL2019data', 'tp-1102-194'))
    # fission yeild data
    ndr.setpath(pf.io.ND_FY_ENDF_KEY(),   os.path.join(nd_base_path,
                'GEFY61data', 'gefy61_nfy'))
    ndr.setpath(pf.io.ND_SF_ENDF_KEY(),   os.path.join(nd_base_path,
                'GEFY61data', 'gefy61_sfy'))
    # Deacy2020 decay data
    ndr.setpath(pf.io.ND_DK_ENDF_KEY(),   os.path.join(nd_base_path,
                'decay2020', 'decay_2020'))
    # Hazards and radiological data from decay2012 files
    ndr.setpath(pf.io.ND_HAZARDS_KEY(),   os.path.join(nd_base_path,
                'decay', 'hazards_2012'))
    ndr.setpath(pf.io.ND_CLEAR_KEY(),     os.path.join(nd_base_path,
                'decay', 'clear_2012'))
    ndr.setpath(pf.io.ND_A2DATA_KEY(),    os.path.join(nd_base_path,
                'decay', 'a2_2012'))
    # attenuation data required for dose rate calculation
    ndr.setpath(pf.io.ND_ABSORP_KEY(),    os.path.join(nd_base_path,
                'decay', 'abs_2012'))
    ndr.load(nd, op=loadfunc)
```

```
    return nd

# nuclear data call back
def loadfunc(k, p, index, total):
    print(" [{}/{}]  Reading {}: {}".format(index, total, k, p), end="\r")
    sys.stdout.write("\033[K")

# set input data
def set_input_data(runname):
    # initialise input data object
    ip = pf.InputData(m)
    # set run name
    ip.setname(runname)
    #
    # set flux details
    #
    # the list of flux values, here in decending energy, 709group structure
    flux = [  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  2.9806E+08,  7.2391E+08,  7.3338E+08,  7.4309E+08,
            2.1367E+09,  3.1298E+09,  1.2641E+09,  6.3931E+08,  6.0755E+07,  3.0446E+07,
            2.1633E+07,  1.8685E+07,  1.5579E+07,  1.5817E+07,  1.5557E+07,  1.3784E+07,
            1.2367E+07,  1.1450E+07,  1.0447E+07,  1.0622E+07,  1.0742E+07,  1.0310E+07,
            1.0493E+07,  1.0576E+07,  1.0464E+07,  1.0659E+07,  1.0251E+07,  9.9709E+06,
            1.0168E+07,  2.2430E+07,  2.0564E+07,  1.9520E+07,  1.8852E+07,  1.8621E+07,
            1.8803E+07,  1.9212E+07,  1.9312E+07,  1.8698E+07,  1.6903E+07,  1.6879E+07,
            1.7133E+07,  1.6529E+07,  1.6401E+07,  1.6155E+07,  1.6995E+07,  1.7381E+07,
            1.6948E+07,  1.6652E+07,  1.6526E+07,  1.6376E+07,  1.6283E+07,  1.6050E+07,
            1.5988E+07,  1.5512E+07,  1.5916E+07,  1.5694E+07,  1.5805E+07,  1.5977E+07,
            1.5868E+07,  1.5489E+07,  1.5761E+07,  1.5463E+07,  1.5340E+07,  1.5227E+07,
            1.5058E+07,  1.5031E+07,  1.4791E+07,  1.4435E+07,  1.4673E+07,  1.4417E+07,
            1.4095E+07,  1.4044E+07,  1.3948E+07,  1.3512E+07,  1.3097E+07,  1.2946E+07,
            1.2726E+07,  1.2614E+07,  1.2580E+07,  1.1950E+07,  1.1605E+07,  1.1272E+07,
            1.0946E+07,  1.0604E+07,  1.0435E+07,  1.0073E+07,  9.7617E+06,  9.5178E+06,
            9.0618E+06,  8.7300E+06,  8.4719E+06,  7.9208E+06,  7.8539E+06,  7.5366E+06,
            7.2098E+06,  7.1581E+06,  6.6208E+06,  6.3730E+06,  6.2884E+06,  6.0896E+06,
            5.8604E+06,  5.7249E+06,  5.6011E+06,  4.9963E+06,  4.9964E+06,  4.7358E+06,
            4.5749E+06,  4.4232E+06,  4.3949E+06,  4.0158E+06,  3.6967E+06,  3.5937E+06,
            3.4541E+06,  3.3024E+06,  3.3809E+06,  3.3065E+06,  2.9941E+06,  2.7719E+06,
            2.3252E+06,  2.4091E+06,  2.3406E+06,  2.5279E+06,  2.5740E+06,  2.4514E+06,
            2.1743E+06,  1.9598E+06,  1.9485E+06,  1.8537E+06,  1.8537E+06,  1.6554E+06,
            1.5054E+06,  1.5054E+06,  1.6360E+06,  1.6701E+06,  1.6230E+06,  1.6230E+06,
            1.4395E+06,  1.3368E+06,  1.2342E+06,  1.2342E+06,  1.2342E+06,  1.1798E+06,
            1.1473E+06,  1.0320E+06,  1.0317E+06,  1.0217E+06,  9.9629E+05,  9.9628E+05,
            8.8628E+05,  8.0295E+05,  8.0295E+05,  8.0295E+05,  9.7740E+05,  1.0185E+06,
```

```
9.4898E+05, 9.4898E+05, 8.8288E+05, 8.1851E+05, 7.9764E+05, 7.8468E+05,
7.8355E+05, 7.3070E+05, 6.9329E+05, 6.9319E+05, 6.8316E+05, 6.5778E+05,
6.5778E+05, 6.5778E+05, 6.5778E+05, 6.5778E+05, 5.7473E+05, 5.6058E+05,
5.6057E+05, 5.6058E+05, 5.6057E+05, 5.5658E+05, 5.5123E+05, 5.4074E+05,
5.1003E+05, 5.1002E+05, 5.1002E+05, 4.7404E+05, 4.7393E+05, 4.7393E+05,
4.7393E+05, 4.7393E+05, 4.3415E+05, 4.0394E+05, 4.0393E+05, 4.0393E+05,
4.0394E+05, 4.0163E+05, 3.8742E+05, 3.8742E+05, 3.8742E+05, 3.8742E+05,
3.8742E+05, 3.8988E+05, 3.9088E+05, 3.9088E+05, 3.8714E+05, 3.8264E+05,
3.6002E+05, 3.0248E+05, 3.0621E+05, 3.3614E+05, 3.3614E+05, 3.1955E+05,
3.3173E+05, 3.2173E+05, 3.1802E+05, 3.0931E+05, 3.0342E+05, 2.9074E+05,
2.7357E+05, 2.7357E+05, 2.7357E+05, 2.7357E+05, 2.7357E+05, 2.3473E+05,
2.3458E+05, 2.3458E+05, 2.3458E+05, 2.3459E+05, 2.2167E+05, 2.1183E+05,
2.1184E+05, 2.1183E+05, 2.1184E+05, 2.0913E+05, 1.9234E+05, 1.9233E+05,
1.9234E+05, 1.9234E+05, 1.9233E+05, 1.7840E+05, 1.7271E+05, 1.7271E+05,
1.7271E+05, 1.7271E+05, 1.6986E+05, 1.6259E+05, 1.6259E+05, 1.6259E+05,
1.6259E+05, 1.6259E+05, 1.6342E+05, 1.6356E+05, 1.6356E+05, 1.6356E+05,
1.6356E+05, 1.5175E+05, 1.3570E+05, 1.3570E+05, 1.3570E+05, 1.3570E+05,
1.3570E+05, 1.3554E+05, 1.3554E+05, 1.3554E+05, 1.3554E+05, 1.3554E+05,
1.3184E+05, 1.2902E+05, 1.2902E+05, 1.2902E+05, 1.2902E+05, 1.2699E+05,
1.1435E+05, 1.1435E+05, 1.1435E+05, 1.1435E+05, 1.1435E+05, 1.0523E+05,
1.0150E+05, 1.0149E+05, 1.0150E+05, 1.0149E+05, 9.5928E+04, 8.1656E+04,
8.1657E+04, 8.1656E+04, 8.1656E+04, 8.1656E+04, 9.2951E+04, 9.4913E+04,
9.4913E+04, 9.4913E+04, 9.4913E+04, 8.6281E+04, 7.4521E+04, 7.4521E+04,
7.4521E+04, 7.4521E+04, 7.4521E+04, 6.1552E+04, 6.1482E+04, 6.1482E+04,
6.1482E+04, 6.1482E+04, 5.6493E+04, 5.2666E+04, 5.2666E+04, 5.2666E+04,
5.2666E+04, 5.4200E+04, 6.3842E+04, 6.3842E+04, 6.3842E+04, 6.3841E+04,
6.3841E+04, 5.7042E+04, 5.4245E+04, 5.4245E+04, 5.4245E+04, 5.4245E+04,
5.3570E+04, 5.1834E+04, 5.1834E+04, 5.1834E+04, 5.1834E+04, 5.1834E+04,
4.8436E+04, 4.7843E+04, 4.7842E+04, 4.7842E+04, 4.7843E+04, 4.5208E+04,
4.1608E+04, 4.1608E+04, 4.1608E+04, 4.1608E+04, 4.1608E+04, 3.7355E+04,
3.7329E+04, 3.7329E+04, 3.7329E+04, 3.7329E+04, 3.8387E+04, 3.9202E+04,
3.9202E+04, 3.9201E+04, 3.9202E+04, 3.9728E+04, 4.3059E+04, 4.3059E+04,
4.3059E+04, 4.3059E+04, 4.3059E+04, 2.9553E+04, 2.3978E+04, 2.3978E+04,
2.3978E+04, 2.3978E+04, 2.5107E+04, 2.8026E+04, 2.8026E+04, 2.8026E+04,
2.8026E+04, 2.8025E+04, 2.2712E+04, 2.1778E+04, 2.1778E+04, 2.1778E+04,
2.1778E+04, 2.2753E+04, 2.4090E+04, 2.4090E+04, 2.4090E+04, 2.4090E+04,
2.4090E+04, 1.8954E+04, 1.8918E+04, 1.8918E+04, 1.8918E+04, 1.8918E+04,
1.7435E+04, 1.6289E+04, 1.6289E+04, 1.6289E+04, 1.6289E+04, 1.6196E+04,
1.5601E+04, 1.5601E+04, 1.5601E+04, 1.5601E+04, 1.5601E+04, 1.4146E+04,
1.3543E+04, 1.3543E+04, 1.3543E+04, 1.3543E+04, 1.3819E+04, 1.4535E+04,
1.4535E+04, 1.4535E+04, 1.4535E+04, 1.4535E+04, 1.4109E+04, 1.4033E+04,
1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04,
1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04,
1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04,
1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4033E+04, 1.4034E+04,
1.4033E+04, 1.4033E+04, 1.4034E+04, 1.4033E+04, 1.4033E+04, 1.1922E+04,
1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04,
1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04,
1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04,
1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04,
1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04,
1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04,
1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04, 1.1922E+04,
1.1922E+04, 1.1923E+04, 1.1922E+04, 1.1922E+04, 1.1923E+04, 1.1922E+04,
1.1922E+04, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00, 0.0000E+00,
```

```
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,  0.0000E+00,
            0.0000E+00 ]
# reverse the order of the flux to assending in energy
flux.reverse()
#  convert 709 group spectrum to 1102 groups to match TENDL2019 group structure
outflux = pf.groupconvert.bylethargy(m, pf.groups.G709(), flux, pf.groups.G1102())
ip.setflux(pf.groups.G1102(), outflux )
ip.setfluxwallloading(1.0)
ip.setfluxname("FNS Inconel Flux")
#
# material deatails
#
# set density (g/cm3)
ip.setdensity(8.42)
# set atoms threshold, MIND keyword
ip.setatomsthreshold(1.0e3)
# set total mass (kg)
ip.setmasstotal(1.0E-3)
# set material my weight percentage, MASS keyword
# atomic numbers for elements to be included, using utilities to convert from element names
atomtic_numbers = [ pf.util.z_from_element(m, 'Ni'),
                    pf.util.z_from_element(m, 'Mn'),
                    pf.util.z_from_element(m, 'Fe'),
                    pf.util.z_from_element(m, 'Cr') ]
# percentge of each element
percent = [ 75.82, 0.39, 7.82, 15.97 ]
# input materials
ip.setmass(  atomtic_numbers, percent )
#
# set irradiation schedule
#
# 5 min irradiation
ip.appendschedule(5.0*pf.util.MIN_TO_SEC(), 1.116E+10)
# cooling steps
```

```
    cooltimes = [36, 15, 16, 15, 15, 26, 33, 36, 53, 66, 66,
                 97, 127, 126, 187, 246, 244, 246, 428, 606, 607 ]
    for i in cooltimes:
        ip.appendschedule(i, 0.0)
    return ip

# run fisapct
def run_fispact(i, nd):
    # initialise output data object
    o = pf.OutputData(m)
    # Run fispact for the inputs and nuclear data set
    pf.process(i, nd, o, m, op=computefunc)
    return o

# fispact process call back
def computefunc(p, index, total):
    print(" [{}/{}]  processing {}".format(index, total, p))

# extract heat from output
def print_heat_after_irrad(o):
    # start at 2nd index to avoid initial data input
    time = 0
    print(" ")
    print(" TIME(yrs) ", "  HEAT(kW/kg)")
    for i in range(o.getnrofinventoryentries() ):
        # get cooling time
        coolingtime = o.getinventoryvalue(i, pf.INVENTORY_COOL_TIME())
        irradtime = o.getinventoryvalue(i, pf.INVENTORY_IRRAD_TIME())
        if coolingtime > 0.0 or irradtime > 0.0:
            ## sum time
            time += coolingtime
            # get total heating from inventory
            total_heat = o.getinventoryvalue(i, pf.INVENTORY_TOTAL_HEAT())
            # get total mass from inventory
            mass = o.getinventoryvalue(i, pf.INVENTORY_TOTAL_MASS())
            print(" {:.4E}   {:.4E} ".format(time/pf.util.YEAR_TO_SEC(), total_heat/mass) )

# extract halflife and heating of nuclides after irradiation
def print_halflife_and_heat(nd, o):
    print(" ")
    print(" DOMINANT NUCLIDES AFTER IRRADIATION")
    print(" HALFLIFE(yrs) ", " HEAT(kW/kg)", " NUCLIDE")
    # get the nuclide list and mass for the first time step
    mass = o.getinventoryvalue(1, pf.INVENTORY_TOTAL_MASS())
    # sort the inventory by decay heat so the dominants can be found
    dom_zai, dom_heat = o.getsortedinventory(1, pf.INVENTORY_TOTAL_HEAT())
    # extract the 20 nuclides with the greatest total heating
    nrofnuclides = len(dom_zai)
    for i in range(nrofnuclides-21, nrofnuclides, 1):
        # call routine which extract nulide heat
        heating = dom_heat[i] / mass
        # call routine which extract nulide half life
        half = extract_halflife_from_nuc(nd, dom_zai[i])
        if half != -1:
            print( " {:.4E}      {:.4E}    {} ".format(
                        half/pf.util.YEAR_TO_SEC(), heating,
```

```
                                pf.util.nuclide_from_zai(m, dom_zai[i])) )


# extract half life from nd - generic routine
def extract_halflife_from_nuc(nd, nuclide_zai):
    halflife = -1
    # get zai of all nuclides with decay data
    decay_zais = nd.getdecayzais()
    for i in range(len(decay_zais)):
        if decay_zais[i] == nuclide_zai:
            # get half life and unc for nuclide if found
            halflife = nd.getdecayhalflife(i)
            break
    return halflife


# extract heating - generic routine
def extract_heating_from_nuc(o, inv_index, nuclide_zai):
    # check if nuclide present at a given time step
    if (o.findinventoryexists(inv_index, nuclide_zai)):
        # get index of nuclide in inventory nuclide list
        nuclide_index = o.findinventoryindex(inv_index, nuclide_zai)
        # get nuclide list at given time step
        nuclide_list = o.getinventorynuclides(inv_index)
        # get heating value for that nuclide from inventory
        heating = nuclide_list[nuclide_index].totalheat
        return heating

#-----------------------------------------------

runname = "FNSInconel_PyAPI_tendl19"
# read commandine arguments
nd_base_path = readargs()
# load global static data
pf.initialise(m)
# load the desired nuclear data from binary
nd = load_nuclear_data(nd_base_path)
# set the required input data
i = set_input_data(runname)
#  run fisapct and fill output data
o = run_fispact(i, nd)
# get heating from output
print_heat_after_irrad(o)
print_halflife_and_heat(nd, o)
# write complete inventory to json output
pf.io.to_file(o, m, "{}.json".format(runname))
# tear down methods
pf.finalise(m)
# write logs to file
pf.io.to_file(m, "{}.log".format(runname))
```

F.1.1 Terminal Output

```
TIME(yrs)     HEAT(kW/kg)
0.0000E+00    4.3240E-07
1.1408E-06    3.7809E-07
2.7569E-06    3.5810E-07
4.8800E-06    3.3822E-07
7.4784E-06    3.2086E-07
1.0552E-05    3.0463E-07
1.4450E-05    2.7889E-07
1.9393E-05    2.5003E-07
2.5477E-05    2.2259E-07
3.3241E-05    1.8847E-07
4.3096E-05    1.5428E-07
5.5042E-05    1.2732E-07
7.0062E-05    9.7018E-08
8.9107E-05    6.9310E-08
1.1214E-04    5.1012E-08
1.4111E-04    3.3858E-08
1.7787E-04    2.1722E-08
2.2236E-04    1.6125E-08
2.7464E-04    1.2959E-08
3.4049E-04    1.0370E-08
4.2554E-04    8.8376E-09
5.2982E-04    8.0412E-09

DOMINANT NUCLIDES AFTER IRRADIATION
HALFLIFE(yrs)    HEAT(kW/kg)   NUCLIDE
1.9398E-01       4.5842E-11     Co58
8.6825E-07       5.0456E-11     Co63
8.5558E-08       1.4440E-10     Mn58m
2.0661E-06       1.9508E-10     Mn58
1.0381E-03       2.0703E-10     Co58m
1.8823E-04       3.0807E-10     Co61
1.1027E-05       5.1556E-10     Ti51
7.9664E-05       5.8303E-10     Cr49
1.6180E-05       6.4704E-10     Fe53
1.1370E-05       7.5152E-10     Fe61
4.0958E-03       1.0561E-09     Ni57
6.7305E-06       1.4868E-09     Cr55
2.7062E-06       1.8364E-09     Mn57
9.5064E-09       2.9888E-09     Co64
1.9901E-05       4.4556E-09     Co60m
2.6447E-05       7.2372E-09     Co62m
2.9418E-04       7.2896E-09     Mn56
1.5781E-06       9.1686E-09     V54
3.0801E-06       2.7177E-08     V53
2.9280E-06       3.2810E-08     Co62
7.1203E-06       3.3343E-07     V5
```