LLNL's Deterministic Transport Access Routines and Data*

Documentation for the Nuclear Data Files (ndf) and the libndf.a access routines

Bret R. Beck
Lawrence Livermore National Laboratory
UCRL-MA-147647

December 3, 2002

*This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract #W-7405-ENG-48.

Contents

List of Tables

y_i	File name	Transportable particle name
1	**ndf1**	Neutron (n)
2	**ndf2**	proton (p)
3	**ndf3**	Deuteron (d)
4	**ndf4**	Triton (t)
5	**ndf5**	Helium 3 (^{3}He)
6	**ndf6**	Helium (^{4}He) also called alpha (α)
7	**ndf7**	gamma (γ)

Table 1: Transportable particles and their corresponding y_i and file name.

1 Introduction

This document describes the routines, provided by the LLNL Computational Nuclear Physics Group, that access the data stored in the **ndfy$_i$** files where y_i is a token to be replaced by a 1, 2, 3, 4, 5, 6 or 7 (e.g., **ndf1**). These files contain nuclear (y_i = 1 to 6) and atomic gamma-ray (y_i = 7) data required by deterministic transport codes. Such codes transport particles through a composite material as particles are destroyed and created through interactions with the composite material. The various **ndfy$_i$** files contain data so that the particles listed in Table 1, called transportable particles, can be transported through various composite materials. Within a **ndfy$_i$** file is a list of targets (also called isotopes) for which there are data for reactions induced by its transportable particle. A composite material is a combination of one or more of these targets. For each target there is a list of nuclear reactions (or atomic gamma-ray reactions for the **ndf7** file) for which there are data for that target. Section 5 describes the type of data stored in a **ndfy$_i$** file for each target.

For example, consider a material composed of only Uranium 238, ^{238}U, for which one would like to transport neutrons. Since only neutrons are being transported, only the **ndf1** file needs to be accessed. The **ndf1** file may contain ^{238}U with data for the nuclear reactions listed in Table 2. Various data, like total cross-section, fission spectrum, etc., are stored for the ^{238}U target. The last reaction listed in the Table 2, capture, creates a γ particle. The energy and angular distribution information about the created γs are available in the **ndf1** file (i.e., transport matrix and deposited energy). In order to transport γs the **ndf7** file would need to be access.

In this document, an incident transportable particle is labeled y_i and an outgoing (appearing on the right-hand-side of the reaction equation) transportable particle is labeled y_o. Some reactions may have more than one type of outgoing transportable particle (e.g., (n, p α) has a proton and an alpha as outgoing particles).

2 A bit of history

Originally, the access routines, called ndf access routines, were a set of LLLTRAN routines, it is claimed, built into a library named **libndf.a**. LLLTRAN was a version of FORTRAN, with extensions, developed at Lawrence Livermore National Laboratory and dates back to at least the 1970s. The ndf routines were incomplete in that some essential knowledge of the data could not be obtained through the access routines; it had to be known or guessed by the code developer. For example, the routine **ndfistab** returns a list of all targets and the number of targets. Memory for the target list is allocated

$n + {}^{238}U \rightarrow n + {}^{238}U$	! Elastic scattering
$n + {}^{238}U \rightarrow n' + {}^{238}U$	! Inelastic scattering
$n + {}^{238}U \rightarrow n + n + {}^{237}U$	! (n,2n)
$n + {}^{238}U \rightarrow n + n + n + {}^{236}U$	! (n,3n)
$n + {}^{238}U \rightarrow n + n + n + n + {}^{235}U$	! (n,4n)
$n + {}^{238}U \rightarrow$ various neutrons + fission fragments	! Fission
$n + {}^{238}U \rightarrow \gamma + {}^{239}U$	! Capture

Table 2: Example of reaction data for ^{238}U in a **ndf1** file.

by the calling routine. If memory for the target list is insufficient a memory overwrite will occur. In the old routines there was no way to determine the number of targets prior to calling **ndfistab** so that appropriate memory could be allocated. Starting sometime in the late 1990's, routines have been added to overcome any deficiencies. For example, a routine **ndfnistab** as been added which returns the number of targets in the opened **ndfy$_\mathbf{i}$** file. C-wrapper routines have also been added and a C-header file ndf.h exist for C programming. The FORTRAN routines have the prefix **ndf** and the C-wrapper routines have the prefix **ndfc**.

Some of the concepts and equations used to calculate the data can also be found in reference [1], in particular chapters VI and VII, and in reference [2].

The data in the **ndfy$_\mathbf{i}$** files is a "processed" form of the data from three nuclear databases developed at LLNL. One database contains information about neutron incident on various targets and is called ENDL (Evaluated Neutron Data Library). Another database contains information about the 5 supported transportable charged particles (p, d, t, ^{3}He, and α) incident on various targets and is called ECPL (Evaluated Charged Particle Library). The last database contains information about gammas (γs) incident on various targets and is called EGDL (Evaluated Gamma Data Library). These databases represent the data in point-wise form. For example, cross-section data is given as 2-column data where the first column is the incident energy in MeV and the second column is the cross-section in barns; as the following lines demonstrate,

```
1.2510000E+01  0.00000E+00
1.4000000E+01  2.00000E-02
2.0000000E+01  2.00000E-02
```

This data is converted from point-wise data into grouped data (see Section 4) to form the **ndfy$_\mathbf{i}$** files and is then called processed data. In the future, the LLNL Nuclear Computations Group may make evaluated data from other databases (e.g., the ENDFB5 database from Brookhaven National Laboratory) available as **ndfy$_\mathbf{i}$** files.

The acronym ENDL as two derivations. At times it will derive from Evaluated Neutron Data Library. At other times it will derive from Evaluated Nuclear Data Libraries. In this latter version, nuclear implies all three databases (ENDL, ECPL and EGDL). Hopefully, the appropriate meaning for this acronym will be clear when it is used at various places in this document.

3 Accessing the library and data at Livermore Computing

The Nuclear Computations Group supports the ndf accessing routines and required nuclear data on LLNL's Livermore Computing Facilities (e.g., GPS and Forest Clusters). When possible the OCF and the SCF ndf libraries and data are identical. Since the data is either in ASCII or a binary independent format it resides in the global directory /usr/gapps/nuclear/data, allowing all LC platforms on OCF or SCF to access the identical data. The directory /usr/gapps/nuclear/data has the following file and sub-directories relevant for using the libndf.a routines:

bdfls) An ASCII file containing the following: (1) lists of group boundaries that can be accessed using **ndfidog**, (2) lists of fluxes used by **ndfgroup**, (3) a list of target masses, (4) a list of target half-lives, (5) some constants and other data of little concern to a user of the ndf accessing routines. See the routines **ndfidog**, **ndfcidog**, **ndfgroup** and **ndfcgroup**.

alpha) A sub-directory containing various processed Evaluated Nuclear Data Library (ENDL) data in the alpha testing stage.

betas) A sub-directory containing various processed ENDL data in the beta testing stage.

current) A sub-directory containing default data accessed by the ndf accessing routines unless a user specifies otherwise (see **ndfaccess** and **ndfinit**).

endl) A sub-directory containing various LLNL ENDL, ECPL and EGDL processed data.

The ndf data in the **alpha**, **betas** and **endl** sub-directories typically resides several sub-directories below these directories. For example, a version of ENDL (Evaluated Neutron Data Library) was released in 1999 and labeled endl99. This data resides inside the **endl99/ndf** sub-directory of the **endl** sub-directory (i.e., in the directory **/usr/gapps/data/nuclear/endl/endl99/ndf**.

The **libndf.a** and **ndf.h** files reside in sub-directories of the **/usr/apps/ndf** directory. The **/usr/apps/ndf** directory has the following sub-directories:

betas) This sub-directory contains various libndf.a and ndf.h files in the beta testing stage. Different releases of libndf.a and ndf.h are signified by a date of the form YYMMDD and reside in a sub-directory named YYMMDD. For example, a version of libndf.a and ndf.h releases on 9-Jan-2002 resides in the sub-directory 020109.

current) A sub-directory containing the version of libndf.a and ndf.h most users should use.

old) A sub-directory containing the previous contents of the **current** sub-directory.

new) A beta version that will likely become the current version.

versions) A sub-directory containing all versions of libndf.a and ndf.h that once resided in the **current** sub-directory. Versions of libndf.a and ndf.h residing in betas may be removed at anytime. However, those residing in the **versions** sub-directory will not be removed, so that a users can, if they wish, always insure that they are getting the same libndf.a version by loading against a libndf.a in a sub-directory in the **versions** sub-directory instead of the **current** sub-directory.

Some releases of libndf.a and ndf.h have a sub-directory named **debug**. This sub-directory contains a libndf.a file compile with the debugging option "-g" and it also contains the source files. For example, the directory **/usr/apps/ndf/versions/020109** contains the sub-directory **debug**.

4 Energy grouping and data collapsing

Much of the data stored in a **ndfy$_i$** file is a function of the incident particle's energy (e.g., total cross-section $\sigma(E)$) (and possibly outgoing transportable particle energy (e.g., transfer matrix)). As is common in deterministic computation, the energies in the **ndfy$_i$** files have been discretized: the multi-energy-group approximation (see [3] page 61) which is called grouping in this document. In grouping, the incident energy range is divided up into ng regions by defining $ng + 1$ energy boundaries. In a **ndfy$_i$** file the boundaries are stored from highest to lowest energy; the data is also stored and retrieved in that order. The incident energy groups are labeled g for $1 \leq g \leq ng$. The outgoing energy groups are label h for $1 \leq h \leq nh$. Only the transfer and fission matrix data are stored as a function of both incident and outgoing energies; their grouping and collapsing are discussed in Section 4.2. All other data is stored as a function of incident energy; their grouping and collapsing are discussed in the next section.

4.1 Grouping and collapsing data only dependent on incident energy

Data that is only a function of incident particle energy is grouped by performing a Legendre-order-flux weighted averaging of the data between group boundaries. That is, for the l^{th} Legendre order Q^l of a quantity $Q(E)$ and the l^{th} Legendre order flux $f^l(E)$, the value of Q^l for group g, Q^l_g, is calculated as,

$$Q^l_g = \frac{\int_{E_{g+1}}^{E_g} f^l(E)\, Q^l(E)\, dE}{\int_{E_{g+1}}^{E_g} f^l(E)\, dE} \quad . \tag{1}$$

Only the transport correcting cross-sections and the interaction transfer matrix have $l > 0$ Legendre orders. All other quantities have only the isotropic $l = 0$ Legendre order; in which case the l-order label is dropped (e.g., σ^0 is written as σ). For example, consider a cross-section $\sigma(E)$ grouped using the 3 groups given by the energy boundaries (20.0, 15.0, 12.3, 0.1 MeV); then the grouped cross-sections are,

$$\sigma_1 = \frac{\int_{15.0}^{20.0} f^0(E)\, \sigma(E)\, dE}{\int_{15.0}^{20.0} f^0(E)\, dE} \tag{2}$$

$$\sigma_2 = \frac{\int_{12.3}^{15.0} f^0(E)\, \sigma(E)\, dE}{\int_{12.3}^{15.0} f^0(E)\, dE} \tag{3}$$

$$\sigma_3 \quad = \quad \frac{\int_{0.1}^{12.3} f^0(E)\,\sigma(E)\,dE}{\int_{0.1}^{12.3} f^0(E)\,dE} \quad . \tag{4}$$

Four quantities are not grouped as given by Eq. 1. One is the group speed v_g which is calculated as,

$$\frac{1}{v_g} = \frac{\int_{E_{g+1}}^{E_g} \frac{f^0(E)}{v(E)}\,dE}{\int_{E_{g+1}}^{E_g} f^0(E)\,dE} \tag{5}$$

since it is $1/v_g$ that appears in transport equations. However, it is v_g which is returned by the access routines. Another quantity not grouped as per Eq. 1 is the group flux f_g^l, which is not weighted,

$$f_g^l = \int_{E_{g+1}}^{E_g} f^l(E)\,dE \quad . \tag{6}$$

The last two quantities not grouped as per Eq. 1 are the transfer and fission matrices. These quantities also depend on outgoing particle energy and are discussed in Section 4.2.

Often the data in a **ndfy$_\mathbf{i}$** file are stored with more energy resolution than a problem requires. In this case, the user can request, by calling **ndfgroup**, that the ndf access routines return the data grouped to a smaller energy group. This smaller energy group must be a subset of the energy group used to generate the data in the **ndfy$_\mathbf{i}$** file. (A subset energy group contains only boundaries of the superset group). The example above has six 1-group subsets (20.0, 15.0), (20.0, 12.3), (20.0, 0.1), (15.0, 12.3), (15.0, 0.1), and (12.3, 0.1), four 2-group subsets (20.0, 15.0, 12.3), (20.0, 15.0, 0.1), (20.0, 12.3, 0.1) and (15.0, 12.3 0.1) and itself as a 3-group subset.

Data mapped to a subset group is referred to as collapsed data in this document and the act of mapping the data is called collapsing. To receive collapsed data one must first call the ndf routine **ndfgroup** (or **ndfcgroup**). The first two arguments of **ndfgroup** are the new group boundaries and the number of new groups (labeled *ncg* is this document). The third argument of this routine is a flux id. The bdfls file is scanned for the requested flux. This flux φ is then grouped onto the old groups and the data is collapsed using this flux as a weight for the old group data. The flux is calculated as,

$$\varphi_{g'}^l = \sum_{g \in g'} \varphi_g^l = \int_{E_{g'+1}}^{E_{g'}} \varphi^l(E)\,dE \quad . \tag{7}$$

where

$$\varphi_g^l = \int_{E_{g+1}}^{E_g} \varphi^l(E)\,dE \quad . \tag{8}$$

Here, g' is a label for the new groups, φ_g^l is the requested flux grouped (see Eq. 8) and $g \in g'$ means to sum over all g for which the boundaries of g fall inclusively between the boundaries of g'.

Collapsing a quantity Q_g^l is calculated as,

$$Q_{g'}^l = \frac{\sum_{g \in g'} \varphi_g^l\, Q_g}{\sum_{g \in g'} \varphi_g^l} \quad . \tag{9}$$

In the above example, collapsing the cross-section to the 2-group (20.0, 15.0, 0.1) yields,

$$\sigma'_1 = \frac{\sigma_1 \varphi^0_1}{\varphi^0_1} = \sigma_1 \tag{10}$$

$$\sigma'_2 = \frac{\sigma_2\, \varphi^0_2 + \sigma_3\, \varphi^0_3}{\varphi^0_2 + \varphi^0_3} \quad . \tag{11}$$

While collapsing to the 1-group (20.0, 12.3) yields,

$$\sigma'_1 = \frac{\sigma_1\, \varphi^0_1 + \sigma_2\, \varphi^0_2}{\varphi^0_1 + \varphi^0_2} \quad . \tag{12}$$

The speed (and flux as given in Eq. 7) is collapsed differently, so as to be consistent with its previous groupings. Speed is collapsed as,

$$\frac{1}{v_{g'}} = \frac{\displaystyle\sum_{g \in g'} \frac{\varphi^0_g}{v_g}}{\displaystyle\sum_{g \in g'} \varphi^0_g} \quad . \tag{13}$$

4.2 Grouping and collapsing data dependent on incident and outgoing energies

This section discusses grouping and collapsing of transfer and fission matrix data. These data are dependent on both incident and outgoing particles' energies. Grouping and collapsing of transfer matrix data are more complicated than is presented here, since either particle number, energy, or number-and-energy of the outgoing particles is conserved during the grouping and collapsing. In this section only particle conserving grouping and collapsing will be discussed. A full discussion of grouping and collapsing of the transfer matrix data can be found in Chapter VI and pages VII-19 to VII-23 of reference [1] and in reference [2].

If $M(E, E')$ is a quantity that is dependent on incident E and outgoing E' energies and $f^l(E)$ is the l-order Legendre flux then the l-order Legendre particle-conserving grouped matrix $M^l_{g,h}$ is calculated as,

$$M^l_{g,h} = \frac{\displaystyle\int_{E_{g+1}}^{E_g} \int_{E'_{h+1}}^{E'_h} f^l(E)\, M^l(E, E')\, dE\, dE'}{\displaystyle\int_{E_{g+1}}^{E_g} f^l(E)\, dE} \quad . \tag{14}$$

Collapsing this data to new incident and outgoing particle groups, label as g' and h', is calculated as,

$$M^l_{g',h'} = \frac{\displaystyle\sum_{g \in g'} \sum_{h \in h'} \varphi^l_g\, M^l_{g,h}}{\displaystyle\sum_{g \in g'} \varphi^l_g} \quad . \tag{15}$$

Here $g \in g'$ means to sum over all g for which the boundaries of g fall inclusively between the boundaries of g' and $h \in h'$ means to sum over all h for which the boundaries of h fall inclusively between the boundaries of h' except that the end points of h' are extended to include the end points of h so as to conserve outgoing particle number. For example, collapsing the outgoing particle's group from

(20.0, 15.0, 12.3, 0.1) to (20.0, 15.0, 12.3) will result in the outgoing particle's collapse group being (20.0, 15.0, 0.1) when collapsing. The incident particle's collapse group is set by calling **ndfgroup** or **ndfcgroup**. For the interaction transfer matrix data and the fission transfer matrix data the outgoing particle's collapse group is also that set by calling **ndfgroup** or **ndfcgroup**. For transfer matrix data for $y_i \neq y_o$ the outgoing particle's collapse group is specified through arguments to the routines **dfpmat** and **ndfcpmat**.

5 Types of data in an ndfy$_i$ file

Each **ndfy$_i$** file contains a global data section, and a target specific data section for each target. The global data section contains target independent data and brief information about each target in the file.

5.1 Global data

This section describes the data in the global data section and the ndf access routines used to retrieve this data.

Date: Date is an integer date in the form YYMMDD (e.g., 991031 for 31-Oct-1999). Originally this date was the day on which the file was processed. Currently, as of about 1-Oct-2000, the date is used to uniquely identify a file. Dates in files generated before about 1-Oct-2000 are not guaranteed to be unique. See routines **ndfinit**, **ndfcinit** and **ndfcopen**.

Group id: The incident particle's group id from the bdfls file used to generate the **ndfy$_i$** file. See routines **ndfgid** and **ndfcgid**.

Number of groups: The number of incident particle groups used to generate the **ndfy$_i$** file. This is designated as ng in this document. See routines **ndfngroups**, **ndfcngroups**, **ndfngroup** and **ndfcngroup**.

Group boundaries: The incident particle's group boundaries used to generate the **ndfy$_i$** file. See routines **ndfgp** and **ndfgid**. This data has units of MeV.

Group speeds: The incident particle's group averaged speeds (see Eqs. 5 and 13). See routines **ndfsp** and **ndfcsp**. This data has units of cm/sh where sh $= 10^{-8}$ second.

Group flux: The grouped flux used to generate the **ndfy$_i$** file (see Eqs. 6, 7 and 8). Typically, the **ndfy$_i$** file is generated with a flat flux with only the $l = 0$ Legendre order specified. There is no routine which returns the flux used to generate the **ndfy$_i$** file. If **ndfgroup** has been called then **ndfflxw**, **ndfflxw_l**, **ndfwsp**, **ndfcflxw**, **ndfcflxw_l** or **ndfcwsp** can be called to obtained the grouped flux for the flux specified with the flux id argument of **ndfgroup**.

yo descriptor: The list of transportable outgoing particles and relevant information. The number of transportable outgoing particles is obtained from **ndfnyos** or **ndfcnyos** and the list is obtained from **ndfyos** or **ndfcyos** (also see **ndfyo** or **ndfcyo**). When a **ndfy$_i$** file is generated the transfer matrix for transportable incident particle going to a transportable outgoing particle is calculated. During this calculation the processing code is instructed, through a flag labeled $iecflg$, to conserve

either transportable outgoing particle number, energy, or number-and-energy. (Traditionally, conservation of number (iecflg = 0) is used for neutrons, conservation of number-and-energy (iecflg = 3) is used for charge particles and conservation of energy (iecflg = 1) is used for gammas.) This flag is stored in the file and used when collapsing. It can be accessed using the routine **ndfyo_info** or **ndfcyo_iecflg**. When the transportable outgoing particle is the same as the transportable incident particle (i.e., $y_i = y_o$) then the matrix generated has dimension $ng \times ng$. When the transportable outgoing particle is different than the transportable incident particle (i.e., $y_i \neq y_o$) then the matrix generated has dimension $nh \times ng$. The value of nh can be accessed using **ndfyo_info** or **ndfcyo_nego**. There exist an $iecflg$ and nh for each transportable outgoing particle type.

Target list: The integer list of targets in the **ndfy$_\mathbf{i}$** file. The targets are listed in terms of their ZA = 1000 × Z + A (e.g., ^{238}U as ZA = 92238). See routines **ndfnistab**, **ndfistab**, **ndfcnistab** and **ndfcistab**.

5.2 Target specific data

This section describes the data in the target specific data section and the ndf access routines used to retrieve this data. To access this data for a specific target one must first call **ndfiso** to select the target.

Atomic mass: The mass of the target in atomic mass units (AMU). Historically, this was called weight, hence the accessing routines are call **ndfatw** and **ndfcatw** for ATomic Weight (atw).

lMax: The maximum Legendre order that data is calculated to and stored in the **ndfy$_\mathbf{i}$** file. This is only relevant for the transfer matrix when the incident particle and outgoing particle are the same (i.e., $y_i = y_o$) and for the transport correcting cross-sections. (When $y_i \neq y_o$ the outgoing particle distribution is assumed to be isotropic; hence, only the $l = 0$ transfer matrix is stored). See routines **ndfmxorder**, **ndfcmxorder**, **ndfmxorder_tc** and **ndfcmxorder_tc**.

Total cross-section: The total cross-section is the sum of all the reaction cross-sections described in item **Reaction specific information** below and has units of barn. The access routines for this data are **ndftotal**, **ndfsig**, **ndfctotal** and **ndfcsig**.

Transport correcting cross-section: Deterministic transport codes that use a Legendre expansion truncate the expansion due to resource limitations. A truncation at order l_T requires a correction from the next order $l_T + 1$. This correction is called the transport correcting cross-section in this document and is label tc_l in Table 3 with $l = l_T + 1$. Four transport correcting cross-section methods are supported by the ndf accessing routines. Table 3 describes the four methods. The default method is Pendlebury/Underhill. Each transport correcting cross-section can be calculated for Legendre order 1 to lMax_{tc} where lMax_{tc} is given in column 3 of Table 3. The LLNL exact method has two caveats; (1) the lMax + 1 exact correction is really the LLNL approximate method, (2) if the i^{th} group total cross-section is less than the exact $\mathrm{tc}_l(i)$ then the exact $\mathrm{tc}_l(i)$ is replaced by the approximate $\mathrm{tc}_l(i)$. See routines **ndfsig**, **ndftrcorr**, **ndfcorrec**, **ndfncorrec**, **ndfcsig**, **ndfctrcorr**, **ndfccorrec** and **ndfcncorrec**. This data has units of barn.

Reaction specific information: Each target contains a list of all reactions included in its processing as well as cross-section and particles produced (including non-transportable particles) for each

Code	Name	lMax_{tc}	$\text{tc}_l(i)$ (for i = 1 to ng)
0	None	lMax + 1	0
1	Pendlebury/Underhill	lMax	$\text{TM}_l(i,i)$
2	LLNL (exact)	lMax	$\sum_{j=1}^{ng} f_l(j)\,\text{TM}_l(i,j)/f_l(i)$
	LLNL (approximate)	lMax + 1	$\sum_{j=1}^{ng} \text{TM}_l(j,i)$
3	Ferguson	lMax	$\sum_{j=i}^{N_3} \text{TM}_l(j,i)$

Table 3: List of the four transport correcting algorithm options available through the ndf access routines. TM_l is the l^{th} Legendre order interaction transfer matrix (see item **Transfer matrices**), $f_l(E)$ is the l^{th} Legendre order flux and N_3 is the lesser of $i+3$ and ng. lMax_{tc} is the highest Legendre order available for the specified algorithm.

reaction. A reaction is identified by a unique combination of C, S, Q, X1, X2, X3, and Q_{eff}-values. The C-value identifies the type of reaction (e.g., C = 10 identifies the reaction as elastic scattering, C = 11 identifies the reaction as inelastic scattering and C = 15 identifies the reaction as fission). Some reaction types are sub-divided as given by the S-, X1-, X2-, and X3-values. As example, for many targets the C = 11, inelastic scattering, reaction is sub-divided into several level excitations (S = 1, X1 = level excitation MeV) and the rest (S = 0). The Q-value is the mass difference of the reaction. Q_{eff} is the threshold-energy for the reaction, and is typically, but not always, $Q_{calculated}$ - X1. $Q_{calculated}$ is calculated during the processing stage from a table of masses, while Q is taken from input data files that may contain slightly different masses; whence Q_{eff} is close but not always equal to Q - X1. All energy values have units of MeV.

The following two examples illustrate the type of reactions found in a **ndfy$_{\text{i}}$** file. (Note, sometimes there is a difference between the actual particles produced and those found in a **ndfy$_{\text{i}}$** file. During the processing of a **ndfy$_{\text{i}}$** file, the particles produced (i.e., particles occurring on the right hand side of the $\rightarrow$ in Tables 4 and 6) are determined. If a product particle is not in the list of targets in the **ndfy$_{\text{i}}$** file then the processing code substitutes for it a nearby target that is in the **ndfy$_{\text{i}}$** file.) Tables 4 and 5 are an example of a neutron incident on a ^{9}Be target. This **ndf1** file does not contain the targets ^{9}Li, ^{8}Li and ^{6}He, so they were replaced by the targets ^{7}Li, ^{7}Li and α respectively. The Q-values are calculated with the actual product particles and not the substituted ones.

Tables 6 and 7 are an example of a neutron incident on a ^{10}B target. Multiple C = 11 entries are present; four with S = 1 for the excitation levels 0.717, 1.74, 2.154 and 3.59 MeV and one with S = 0 for the rest of the inelastic reaction cross-section data.

The access routines allow one to obtain information about the number of reactions, the list of C-values, the summed cross-section and Q-value for a reaction type, and the particles produced for a specific reaction as described below.

Number of reactions: The number of reactions for a specific target is obtained by calling **ndfnreact** or **ndfcnreact**. For Tables 5 and 7 the number of reactions are 8 and 11 respectively. Also see routines **ndfreact**, **ndfnrxs**, **ndfrxs**, **ndfrxslist**, **ndfrxslevel**, **ndfcreact**, **ndfcnrxs**, **ndfcrxs**, **ndfcrxslist** and **ndfcrxslevel**.

n + ^{9}Be → n + ^{9}Be	! C = 10, Elastic scattering
n + ^{9}Be → n + n + α + α	! C = 12, (n,2n)
n + ^{9}Be → p + ^{9}Li	! C = 40, (n,p)
↪ n + ^{9}Be → p + ^{7}Li	! ^{9}Li replaced with ^{7}Li
n + ^{9}Be → d + ^{8}Li	! C = 41, (n,d)
↪ n + ^{9}Be → d + ^{7}Li	! ^{8}Li replaced with ^{7}Li
n + ^{9}Be → t + ^{7}Li	! C = 42, (n,t)
n + ^{9}Be → α + ^{6}He	! C = 45, (n,α)
↪ n + ^{9}Be → α + α	! ^{6}He replaced with α
n + ^{9}Be → γ + ^{10}Be	! C = 46, Capture

Table 4: Example of the list of reactions for ^{9}Be in a **ndf1** file. The ↪ indicates lines that are a duplicate of the line above them except that the product particle substituted by the processing code is listed instead the actual product particle.

C	S	Q	X1	X2	X3	Q_{eff}
10	0	0.	0.	0.	0.	0.
12	0	-1.5728	0.	0.	0.	-1.5728
40	1	-12.83	0.	0.	0.	-12.824
41	1	-14.66	0.	0.	0.	-14.664
42	1	-10.44	0.	0.	0.	-10.439
42	1	-10.44	0.478	0.	0.	-10.917
45	1	-0.6	0.	0.	0.	-0.60175
46	0	6.82	0.	0.	0.	6.82

Table 5: Actual reaction data for the reactions in Table 4 as stored in a **ndf1** file.

n + ^{10}B → n + ^{10}B	! C = 10, Elastic scattering
n + ^{10}B → n' + ^{10}B	! C = 11, Inelastic scattering
n + ^{10}B → n + d + α + α	! C = 23, (n,n d α)
n + ^{10}B → n + n + p + α + α	! C = 31, (n,2n p α)
n + ^{10}B → t + α + α	! C = 43, (n,t α)
n + ^{10}B → α + ^{7}Li	! C = 45, (n,α)
n + ^{10}B → γ + ^{11}B	! C = 46, Capture

Table 6: Example of the list of reaction for ^{10}B in a **ndf1** file.

C	S	Q	X1	X2	X3	Q_{eff}
10	0	0.	0.	0.	0.	0.
11	0	0.	0.	0.	0.	0.
11	1	0.	0.717	0.	0.	-0.717
11	1	0.	1.74	0.	0.	-1.74
11	1	0.	2.154	0.	0.	-2.154
11	1	0.	3.59	0.	0.	-3.59
23	0	-6.02	0.	0.	0.	-6.02
31	0	-8.158	0.	0.	0.	-6.02
43	0	0.33	0.	0.	0.	0.33
45	0	2.8	0.	0.	0.	2.8
46	0	11.45	0.	0.	0.	11.45

Table 7: Actual reaction data for the reactions in Table 6 as stored in a **ndf1** file.

Reaction cross-section: Reaction specific, for a specified C-value, cross-section data are obtained by calling **ndfrxs** or **ndfcrxs**. Also see routines **ndfnrxs**, **ndfrxslist**, **ndfrxslevel**, **ndfcnrxs**, **ndfcrxslist** and **ndfcrxslevel**. This data has units of barn.

Reaction particles produced: The particles produced for a specified Reaction, C-value, are obtained by calling **ndfprod** or **ndfcprod**. As example, for the C = 40 reaction for the data in Table 4, **ndfnprod** would return 2 and **ndfprod** would return the lists ZAList = { 2, 3007 } and MList = { 1, 1 } since two particles are produced (i.e., p and ^{7}Li), each with multiplicity 1. Also see routines **ndfnprod**, and **ndfcnprod**.

Transportable particles produced, (i.e., y_os): Each target produces transportable particles. For example, the ^{9}Be target of Table 4 and the ^{10}B target of Table 6 each produce 6 transportable particles (n, p, d, t, α and γ). The number of difference transportable particle types produced is obtained by calling **ndfnyos** or **ndfcnyos**. A list of transportable particle types produced is obtained by calling **ndfyos** or **ndfcyos**. Also see routines **ndfnppyos**, **ndfppyos**, **ndfcnppyos**, and **ndfcppyos** which are similar but do not include the incident transportable particle in the number of particles produced or the list.

Transfer matrices: The formulas for the transfer matrices will not be presented here; instead, see chapter VI of reference [1] and reference [2]. In summary, three methods can be used to calculate a transfer matrix, depending on whether particle number, energy or number/energy of the outgoing particles is to be conserved during the processing (see item **yo descriptor** in Section 5.1). Only the particle number conserving transfer matrix is outlined here. The particle number conserving transfer matrix for outgoing particle y_o is defined as,

$$J^{l}_{\mathrm{y}_o,g\to h} = \sum_{r,\mathrm{y}_o} \left(\frac{\displaystyle\int_{E_{g+1}}^{E_g} \int_{E'_{h+1}}^{E'_h} f^l(E)\,\sigma_r(E)\,M_{r,\mathrm{y}_o}(E)\,\pi^{l}_{r,g\to h}\,dE\,dE'}{\displaystyle\int_{E_{g+1}}^{E_g} f^l(E)\,dE} \right) \quad . \tag{16}$$

Here, h is the outgoing particle's group designation, $\sum_{r,\mathrm{y}_o}$ means to sum only reactions that produce y_o as an outgoing particle, $\sigma_r(E)$ and $M_{r,\mathrm{y}_o}(E)$ are the cross-section and multiplicity

respectively for the reaction (e.g., for the C = 13, (n,3n), reaction and yo = 1, $M_{13,1}(E) = 3$) and $\pi^l_{r,g\rightarrow h}$ is the grouped, l^{th} Legendre coefficient of the normalized probability of the incident particle of energy E producing the required outgoing particle with energy E' at angle θ.

The transfer matrices are divided into two types. The first type is called the interaction transfer matrix and is the transfer matrix when the incident transportable particle type is the same as the outgoing transportable particle type (i.e., $y_i \rightarrow y_o$ for $y_i = y_o$). The interaction transfer matrix is calculated to Legendre order l = lMax + 1 during processing, so that the lMax + 1 LLNL approximate transport correcting cross-sections can be calculated, but is only stored to order l = lMax in the **ndfy$_\mathbf{i}$** file. These matrices are accessed with routines **ndfsig**, **ndftransfer**, **ndfcsig** and **ndfctransfer**. The second type of transfer matrix is called the production transfer matrix. This matrix is the transfer matrix when the incident transportable particle type is different than the outgoing transportable particle type (i.e., $y_i \rightarrow y_o$ for $y_i \neq y_o$) and is only calculated for Legendre order $l = 0$. These matrices are accessed with routines **ndfnppyos**, **ndfppyos**, **ndfpmat**, **ndfppmatrix**, **ndfcnppyos**, **ndfcppyos**, **ndfcpmat** and **ndfcppmatrix**. This data has units of barn.

The multiplicity factor in Eq. 16 often leads to confusion with users of the ndf access routines as they assume that the total cross-section must be greater than or equal to the $l = 0$ interaction transfer matrix integrated over outgoing particle energy; since the sum is only over reactions that produce a specific outgoing particle. To understand the confusion, consider a simple problem with a neutron incident on target X that only has the two reactions C = 10 (n,n) and C = 13 (n,3n). Let both reactions' cross-sections be isotropic, independent of energy and 1 barn. Thus, the $l = 0$ interaction transfer matrix integrated over outgoing particle energy is 4 barns ($1 \times 1.$ barn + $3 \times 1.$ barn) which is greater than the total cross-section of 2 barns.

Fission $< \nu\sigma_f >_g$: For neutrons produced by fission, C = 15, the multiplicity is dependent on energy and not an integer (since it is an average multiplicity over many possible fission channels). The average number of neutrons produced by fission $< \nu\sigma_f >_g$ is stored for targets that fission and is calculated as,

$$< \nu\sigma_f >_g = \frac{\int_{E_{g+1}}^{E_g} \bar{\nu}(E)\,\sigma_f(E) f^0(E)\,dE}{\int_{E_{g+1}}^{E_g} f^0(E)\,dE} \tag{17}$$

where $\bar{\nu}(E)$ is the number of neutrons produced by fission average over possible fission channels and σ_f is the fission cross-section. This data only occurs for neutron as incident particle ($y_i = 1$) and when the neutron multiplicity returned by **ndfprod** or **ndfcprod** is 0. This data is accessed with routines **ndffisx** and **ndfcfisx**, and has units of barn.

Fission neutron transfer matrix: For targets in a **ndf1** file that have fission data the $l = 0$ fission transfer matrix is calculated. It is calculated just like the $l = 0$ interaction transfer matrix except only the fission reaction data is used. See routines **ndffisx ndfifsp**, **ndffsp**, **ndfcfisx**, **ndfcifsp** and **ndfcfsp**. This data has units of barn.

Energy conservation and energy data: To conserve energy, particles that are not transported by a code have to have their energy deposited locally. The amount of energy that must be deposited locally and how to obtain it from a **ndfy$_\mathbf{i}$** file is outlined below.

During the processing of a **ndfy$_i$** file, the average kinetic energy of an incident particle in group g is calculated as,

$$< E\,\sigma >_g = \sum_r \left(\frac{\int_{E_{g+1}}^{E_g} E\,\sigma_r(E)\,f^0(E)\,dE}{\int_{E_{g+1}}^{E_g} f^0(E)\,dE} \right) = \frac{\int_{E_{g+1}}^{E_g} E\,\sigma(E)\,f^0(E)\,dE}{\int_{E_{g+1}}^{E_g} f^0(E)\,dE} \quad . \tag{18}$$

Here, $\sigma_r(E)$ is the cross-section for a given reaction and $\sigma(E)$ is the total cross-section.

The average energy due to the mass differences between the before and after reaction particles, called the production energy (see item **ep** below), is calculated as,

$$< Q\,\sigma >_g = \sum_r Q_r \left(\frac{\int_{E_{g+1}}^{E_g} \sigma_r(E)\,f^0(E)\,dE}{\int_{E_{g+1}}^{E_g} f^0(E)\,dE} \right) = \sum_r Q_r\,\sigma_{r,g} \tag{19}$$

where Q_r is the sum of the incident particle's and target's masses minus the sum of all outgoing particles' masses (this includes non-transportable particles). The sum of Eqs. 18 and 19 is the total available energy (see item **emax** below),

$$< E\,\sigma >_{g,Available} = < E\,\sigma >_g + < Q\,\sigma >_g \quad . \tag{20}$$

During the processing the average energy deposited to a transportable outgoing particles (see item **ed(y$_o$)** below) is calculated as,

$$< E'\,\sigma >_{g,\mathrm{y}_o} = \sum_r \left(\frac{\int_{E_{g+1}}^{E_g} E'_{r,\mathrm{y}_o}(E)\,\sigma_r(E)\,f^0(E)\,dE}{\int_{E_{g+1}}^{E_g} f^0(E)\,dE} \right) = \frac{\int_{E_{g+1}}^{E_g} E'_{r,\mathrm{y}_o}(E)\,\sigma(E)\,f^0(E)\,dE}{\int_{E_{g+1}}^{E_g} f^0(E)\,dE} \quad . \tag{21}$$

Here E'_{r,y_o} is the average energy deposited to a transportable particle of type y$_o$ for reaction r. The amount of energy to be locally deposited is calculated as the total available energy, Eq 20, minus the energy deposited to all transportable outgoing particles that are being transported.

emax: This is the total available energy (Eq. 20) and has units of MeV-barn. This data is accessed using **ndfemax** or **ndfcemax**.

ep: This is the production energy (Eq. 19) and has units of MeV-barn. This data is accessed using **ndfep** or **ndfcep**.

ed(y$_o$): This is the average energy deposited to transportable outgoing particle y$_o$ (Eq. 21) and has units of MeV-barn. This data is accessed using **ndfsig** or **ndfcsig** when (y$_o$ = y$_i$) and is accessed using **ndfpmat** or **ndfcpmat** when (y$_o \neq$ y$_i$). Also see routines **ndfed**, **ndfnyos**, **ndfyos**, **ndfced**, **ndfcnyos** and **ndfcyos**.

6 Routine description format and Data format

Section 7.4 is an alphabetical list of the FORTRAN ndf accessing routines and section 7.6 is an alphabetical list of its C-wrapper routines. The description of each FORTRAN routine contains four parts. The first part labeled **Purpose** is a brief statement about the routine. The second part labeled **FORTRAN Calling** presents the routine and its arguments, if any. If the routine has arguments then a table containing the following information about each argument immediately follows;

Argument type	Argument name	Input/Output	Unit	Description.

Argument type is the FORTRAN declaration of the argument in the ndf routine, or function return type if the routine is a FORTRAN function.

Argument name is its name and dimension, if any, as declared in the ndf routine. All multi-dimensional data are stored as 1-dimensional data. An argument name defined as 'V(c × r)' is a serial representation of a 2-dimensional matrix named 'V' and of size c × r. The fastest varying dimension is c. For example, the production transfer matrices are a function of outgoing and incident particles' energies. With nch outgoing energy groups and ncg incident energy groups the production transfer matrices are defined as J($nch \times ncg$). Accessing the i^{th} outgoing energy group and the j^{th} incident energy group is done serially as $J(nch * j + i)$.

Input/Output is one of the following: (i) the argument is "read from" in the ndf routine, (o) the argument is "written to" in the ndf routine, (f) the routine is a function which returns this argument type or (u) the argument is maintained for historical reasons but is not used.

Unit is the unit for this argument (N/A is used for unit less arguments).

Description is a short summary of the type of data.

The third part is a description of the routine. The next part, if present, if **Fatal Message** and describes reason why the routine will abort execute of the program. The final part list other routines closely related to this routine and is labeled **Related routines**. The first routine listed is the routine that must be called before the described routine is called. If this must routine is not called, then the described routine will print a fatal message. If the first routine listed is "None" then no other ndf accessing routine must be called prior to calling the described routine. The must routine may require another routine to be called before it is called. For example, the routine **ndfatw** requires the routine **ndfiso**, the routine **ndfiso** requires the routine **ndfbuff** and the routine **ndfbuff** requires the routine **ndfinit**.

7 ndf access routines

This section describes the FORTRAN ndf access routines and their C-wrapper routines. The description of **ndfaccess** outlines the search path used by **ndfinit** (**ndfcinit**) to file the requested **ndfy$_{\mathbf{i}}$** file. In general, **ndfaccess** (**ndfcaccess**) must be called before **ndfinit** (**ndfcinit**). If the transport correction type is to be set, then **ndftrcorr** (**ndfctrcorr**) must be call after each call to **ndfinit** (**ndfcinit**) as **ndfinit** (**ndfcinit**) resets the transport correction type to the default. After **ndfinit** (**ndfcinit**) one must call **ndfbuff**) (**ndfcbuff**). (If **ndfcopen** is used instead of **ndfcinit** then one must not call

ndfcbuff.) Before any target specific information can be obtained, **ndfiso** (**ndfciso**) must be called to select the desired target. To open another file **ndfclose** (**ndfcclose**) must first be called to close the current file.

7.1 Example of FORTRAN usage

The following FORTRAN example demonstrates how to open a file and loop through all the targets to print out their ZA, mass and the total cross-section for each reaction type.

```
Program ZA

Integer i, yi, iZA, nZAs, Date, ReqdMem, ncg, gid, fid, tcType
Integer Dummy, IsoErr, ZAs(250)     !250 should be large enough for ZAs.
Integer C, iC, nC, CList(250)       !250 should be large enough for CList
Pointer ( pR8, R8 )                 !This may not work on all platforms.
Real*8 Mass, R8, Q
Real*8 cgb(250), cs(250)            !250 should be large enough.
Character Name*4, Path*256

yi = 1                              !Neutron as incident particle.
gid = 93                            !Group id for collapsing.
fid = 0                             !Flux id for collapsing.
tcType = 1                          !Pendlebury/Underhill transport correction

Call ndfinit( yi, Name, Date, ReqdMem ) !Open the ndf1 file.
Print *, 'Opening ndf file ', Name, ' with date = ', Date
Call ndfinfo( Path )                !Get full path of opened ndf file.
Print *, 'path = ', Path
Call ndf_malloc( pR8, 8 * ReqdMem ) !Get memory (an undocumented routine).
Call ndfbuff( pR8 )                 !Pass work memory to ndf.
Call ndftrcorr( tcType )            !Set the transport correction method.
Call ndfidog( gid, cgb, ncg, Dummy ) !Get group from bdfls file.
Call ndfgroup( cgb, ncg, fid )      !Set collapsing.
Call ndfistab( ZAs, nZAs )          !Get list of targets.

Do iZA = 1, nZAs                    !Loop over target list.
   Call ndfiso( ZAs(iZA), IsoErr )  !Select next target.
   Call ndfatw( Mass )              !Get targets mass.
   Print *, 'Processing ZA = ', ZAs(iZA), '.  Mass = ', Mass, ' AMU.'
   Call ndfreact( CList, nC )       !Get list of reactions for current target.
   C = -1
   Do iC = 1, nC                    !Loop over reactions.
      If( C .ne. CList(iC) ) Then   !Do only if different C-value.
         C = CList(iC)
         Call ndfrxs(C, cs, Q, 0)   !Total cross-section for reaction C.
```

```
            Print *, 'C = ', C, '  Q = ', Q
            Print '( 51pe18.10 )', ( cs(i), i = 1, ncg )
         EndIf
      EndDo
   EndDo

   Call ndfclose( )                    !Close the ndf file.
   End
```

The ndf routine **ndf_malloc** is used for internal testing and is not guaranteed to work on all systems. The above example uses "Cray pointers" (the statement starting with "Pointer") which are not supported by all FORTRAN compilers. The following FORTRAN code will **not** work as the routine **ndfbuff** requires a pointer to a pointer (probably requiring a FORTRAN compiler that supports "Cray pointers").

```
   Real*8 R8(1000000)

   Call ndfinit( yi, Name, Date, ReqdMem )
   Call ndfbuff( R8 )                  ! This does not work.
```

7.2 Example of C usage

The following C example demonstrates how to open a file and loop through all the targets to print out their ZA, mass and the total cross-section for each reaction type.

```
#include <stdio.h>
#include <ndf.h>

main( ) {

    int yi = 1, iZA, nZAs, *ZAs, i;
    int ncg, gid = 93, fid = 0, iC, nC, C, *CList;
    double cgb[250], cs[250], Q;         /* 250 should be large enough. */
    char name[5], Path[256];
    CorrectionTypes tcType = endf_LLNL;  /* LLNL transport correction. */

    printf( "\nOpening ndf file ndf%d with date = %d\n",
             yi, ndfcopen( yi, name ) );
    ndfcinfo( Path, sizeof( Path ) );
    printf( "Path = %s\n", Path );
    ndfctrcorr( tc_Type );               /* Set the transport correction method. */
    ncg = ndfcidog( gid, cgb );          /* Get group from bdfls file. */
    ndfcgroup( ncg, cgb, fid );          /* Set collapsing. */
```

```
    nZAs = ndfcistab( &ZAs );            /* Get list of targets. */
    for( iZA = 0; iZA < nZAs; iZA++ ) {  /* Loop over target list. */
        ndfciso( ZAs[iZA] );             /* Select next target. */
        printf( "\nProcessing ZA = %d.  Mass = %e AMU.\n",
                ZAs[iZA], ndfcatw( ) );
        nC = ndfcreact( &CList ); /* Get list of reactions for current target. */
        C = -1;
        for( iC = 0; iC < nC; iC++ ) {   /* Loop over reactions. */
           if( C != CList[iC] ) {        /* Do only if different C-value. */
              C = CList[iC];
              Q = ndfcrxs( C, 0, cs );   /* Total cross-section for reaction C. */
              printf( "C = %2d  Q = %e\n", C, Q );
              for( i = 0; i < ncg; i++ ) {
                  printf( "%18.10e\n", cs[i] );
              }
           }
        }
    }
    ndfcclose( );                        /* Close the ndf file. */
}
```

7.3 Summary of FORTRAN routines

ndfaccess	To select a **ndfy$_i$** data file other then the default.
ndfatw	Returns the mass for the current target in AMU.
ndfbuff	Provide memory allocated by the user to the ndf routines.
ndfclose	Closes the opened **ndfy$_i$** file.
ndfcorrec	Returns the transport-correcting cross-section for the current target.
ndfed	Returns the deposited energy for the requested transportable outgoing particle for the current target.
ndfemax	Returns the total available energy for the current target.
ndfep	Returns the production energy for the current target.
ndffism	Returns the fission matrix for the current target.
ndffisx	Returns the fission $< \nu\sigma >_g$ for the current target.
ndfflxw	Returns the collapsed flux weights for $l = 0$.
ndfflxw_l	Returns the collapsed flux weights for the requested Legendre order.
ndffsp	Returns the normalized fission spectrum for the current target.
ndfgid	Returns the group id of the uncollapsed energy boundaries of the incident particle.
ndfgp	Returns the uncollapsed energy boundaries of the incident particle.
ndfgroup	Provides the user supplied information needed for collapsing of the data.
ndfidog	Reads an energy group from a bdfls file.
ndfifsp	Returns a flag indicating whether or not fission data is present.
ndfinfo	Returns the path of the opened **ndfy$_i$** file.
ndfinit	Opens an **ndfy$_i$** file.

ndfiso	Selects a target from the opened **ndfy$_i$** file.
ndfistab	Returns the list of targets in the opened **ndfy$_i$** file.
ndfmxorder	Returns the maximum Legendre order in the opened **ndfy$_i$** file.
ndfmxorder_tc	Returns the maximum Legendre order allowed by the current transport correction method.
ndfncorrec	Returns the length of data in 'double' words required by **ndfcorrec**.
ndfngroup	Returns the number of collapsed groups.
ndfngroups	Returns the number of uncollapsed groups.
ndfnistab	Returns the number of targets in the opened **ndfy$_i$** file.
ndfnmaxgps	Returns the largest group size used in processing the opened **ndfy$_i$** file.
ndfnppyos	Returns the number of transportable outgoing particles for which a production transfer matrix exist for the current target.
ndfnprod	Returns the number of outgoing particles produced by the requested reaction for the current target.
ndfnreact	Returns the number of reactions for the current target.
ndfnrxs	Returns the number of reactions of type C for the current target.
ndfnyos	Returns the number of transportable outgoing particles for the current target.
ndfpmat	Returns the collapsed production transfer matrix and the corrected, collapsed, deposited energy.
ndfppmatrix	Returns the uncollapsed production transfer matrix for the requested outgoing particle for the current target.
ndfppyos	Returns the list of transportable outgoing particles for which there are production transfer matrices for the current target.
ndfprod	Returns the ZA and multiplicity lists for particles produced by the requested reaction for the current target.
ndfreact	Returns a list of C-values for the reactions for the current target.
ndfrxs	Returns the requested reaction's cross-section data for the current target.
ndfrxslevel	Returns S-, Q-, X1-, X2-, X3-, Q_{eff}-values and the cross-section for the requested level of the requested reaction for the current target.
ndfrxslist	Returns a list of S-, Q-, X1-, X2-, X3- and Q_{eff}-values for the requested reaction for the current target.
ndfsig	Returns the total cross-section (transport corrected), interaction transfer matrix ($y_i \rightarrow y_o$ for $y_i = y_o$ transport corrected), energy deposited by y_o and the transport correcting cross-section for the requested Legendre order for the current target.
ndfsp	Returns the group speeds for the opened **ndfy$_i$** file.
ndftotal	Returns the uncorrected total cross-section for the current target.
ndftransfer	Returns the uncollapsed, uncorrected interaction transfer matrix for the requested Legendre order for the current target.
ndftrcorr	Sets the desired transport correction method.
ndfwsp	Returns the collapsed normalized $l = 0$ flux.
ndfyo	Returns the j^{th} particle's id from the particle directory of the global data section.
ndfyos	Returns a list of transportable outgoing particles with energy deposit data for the current target.

ndfyo_gid	Returns the group id for the requested outgoing particle.
ndfyo_info	Returns the number of energy groups, nh, and the conservation flag, iecflg, for the requested transportable outgoing particle.

7.4 FORTRAN routines

ndfaccess

Purpose:

To select a **ndfy$_i$** data file other then the default.

FORTRAN Calling:

Call ndfaccess(yi, libnam, Version, grptype, subpath)

Integer	yi	i	N/A	Id of the incident particle (i.e., y_i)
Character*(*)	libnam	i	N/A	Evaluated data library name (e.g., 'endl')
Character*(*)	Version	i	N/A	Version of evaluated data (e.g., '991129')
Character*(*)	grptype	i	N/A	Suffix added to file name (e.g., '175')
Character*(*)	subpath	o	N/A	Returned string of sub-directory

In default mode, **ndfinit** looks for the requested **ndfy$_i$** data file in the current working directory. If the file is not found, **ndfinit** looks in the directory pointed to by the environment variable NDFPATH. If NDFPATH is not defined or the requested **ndfy$_i$** data file does not exist in the directory pointed to by NDFPATH, then **ndfinit** looks in the directory

/usr/gapps/data/nuclear/current/ndf

for the requested file. This final location can be changed by calling **ndfaccess** before calling **ndfinit**. If **ndfaccess** is called first, then **ndfinit** looks for the file

/usr/gapps/data/nuclear/*libnam*/*Version*/**ndf/ndfy$_i$**.*grptype*,

assuming the file was not found in the current working directory or in the directory pointed to by NDFPATH. For example,

```
Call ndfaccess( 1, "endl", "endl94", "230", subpath )
```

will cause **ndfinit** to look for the file

/usr/gapps/data/nuclear/endl/endl94/ndf/ndf1.230 .

Not calling **ndfaccess** is equivalent to calling it as

```
Call ndfaccess( 1, "current", "", "", subpath )
```

.

A call to **ndfaccess** only modifies the search path for the specified incident particle. Thus, to get neutron (yi = 1) and gamma (yi = 7) data from the same location, **ndfaccess** must be called twice (once with yi = 1 and once with yi = 7). Calling **ndfaccess** with different parameters for different incident particles is valid. This allows one to mix and match data for different incident particles.

Fatal Message(s): Prints a fatal message if yi is invalid.

Related routines: None: ndfinit

ndfatw

Purpose:

Returns the mass for the current target in AMU.

FORTRAN Calling:
Call ndfatw(mass)

Real*8	mass	o	AMU	Atomic mass in AMU for the current target

This routine returns the current target's mass in atomic mass units (AMU).

Related routines: ndfiso:

ndfbuff

Purpose:
Provide memory allocated by the user to the ndf routine.

FORTRAN Calling:
Call ndfbuff(Pointer)

Real*8	Pointer	i	N/A	Pointer to pointer to memory allocated by the user

Most ndf routines require the user to allocate memory to read and process the data. The last argument in the **ndfinit** routine is the amount of memory in 8-byte words that is needed to process the data. After calling **ndfinit** the user must allocate the required memory and pass the pointer to **ndfbuff** before calling most ndf routines. As example,

```
Integer Date, ReqdMem
Character Name*(4)
Real*8 DummyArray
Pointer ( p, DummyArray(*) )            ! This may not work on all systems.

Call ndfinit( 1, Name, Date, ReqdMem )
p = GetBytes( 8 * ReqdMem )             ! User routine to allocate memory.
Call ndfbuff( p )
```

Most ndf routines cannot be called until **ndfinit**, **ndfbuff** and then **ndfiso** are called.
Related routines: ndfinit:

ndfclose

Purpose:
Closes the opened **ndfy$_{\mathbf{i}}$** file.

FORTRAN Calling:
Call ndfclose( )

To properly close the opened **ndfy$_{\mathbf{i}}$** file **ndfclose** must be called. Only one **ndfy$_{\mathbf{i}}$** file can be opened at a time.

Fatal Message(s): Calling **ndfinit** twice without calling **ndfclose** between the two **ndfinit** calls will cause a fatal message to be printed.

Related routines: ndfinit:

ndfcorrec

Purpose:

Returns the transport-correcting cross-section for the current target.

FORTRAN Calling:

Call ndfcorrec(tccs)

Real*8	tccs(ncg × (lMax + 1))	o	barn	The transport-correcting cross-section

For each target, the total cross-section and the diagonal of the $y_i \rightarrow y_i$ interaction transfer matrix are transport corrected. This routine returns the transport-correcting cross-section; that is, the cross-section that is subtracted from the total cross-section and the diagonal of the interaction transfer matrix to produce the transport-corrected total cross-section and the transport corrected interaction transfer matrix. There are *ncg* values (see **ndfngroup**) for each l-order (l = 0, ..., lMax). The data are arranged serially with the first *ncg* points being the $l = 0$ data, the second being the $l = 1$ data and so on. The last *ncg* points are only valid when the transport correction method is e_ndfnone or e_ndfLLNL (see **ndfmxorder_tc** or **ndfcmxorder_tc**). Note, as per Table 3 the $l = 0$ data is tc_1, the $l = 1$ data is tc_2, ... and the $l =$ lMax data is tc_{lMax+1}.

Related routines: ndfiso: ndfgroup, ndfngroup, ndfsig, ndftrcorr, ndfmxorder, ndfmxorder_tc

ndfed

Purpose:

Returns the deposited energy for the requested transportable outgoing particle for the current target.

FORTRAN Calling:

Call ndfed(ed, yo)

Real*8	ed(ncg)	o	MeV-barn	yo's deposit energy
Integer	yo	i	N/A	Id of the transportable outgoing particle

The deposit energy for yo is returned in *ed*, where *yo* is the id of the transportable outgoing particle (e.g., $yo = 1$ is a neutron). The are *ncg* data points returned (see **ndfngroup**). If the requested yo does not exist for the current target **ndfed** prints a fatal message. Use **ndfnyos** and **ndfyos** to determine the allowable yos. In general, **ndfsig**, **ndfpmat**, **ndfsig**, **ndfpmat** should be used to obtain the deposition energy for each particle being transported.

Fatal Message(s): Prints a fatal message if the requested yo does not exist for the current target.

Related routines: ndfiso: ndfnyos, ndfyos, ndfnppyos, ndfppyos

ndfemax

Purpose:
Returns the total available energy for the current target.

FORTRAN Calling:
Call ndfemax(em)

Real*8	em(ncg)	o	MeV-barn	Available energy

The total available energy is returned in *em*. There are *ncg* data points (see **ndfngroup**).

Related routines: ndfiso:

ndfep

Purpose:
Returns the production energy for the current target.

FORTRAN Calling:
Call ndfemax(ep)

Real*8	ep(ncg)	o	MeV-barn	Production energy

The production energy is returned in ep. There are *ncg* data points (see **ndfngroup**).

Related routines: ndfiso:

ndffism

Purpose:
Returns the fission matrix for the current target.

FORTRAN Calling:
Call ndffism(fm, iFlag)

Real*8	fm(ncg × ncg)	o	barn	Fission matrix
Integer	iFlag	o	N/A	Indicates whether or not data is present

The fission matrix, if present (iFlag = 0 is returned), for the current target is returned in fm. There are $ncg \times ncg$ data points (see **ndfngroup**). If no fission data is present then iFlag = 1 is returned. This data only exist for neutron as an incident particle (i.e., **ndf1**) and only for some of its targets.

Related routines: ndfiso: ndffisx, ndffsp

ndffisx

Purpose:
Returns the fission $< \nu\sigma >_g$ for the current target.

FORTRAN Calling:
Call ndffisx(fnubar, iFlag)

Real*8	fnubar(ncg)	o	barn	Fission nubar data
Integer	iFlag	o	N/A	Indicates whether or not data is present

The average number of neutrons produced by fission $< \nu\sigma >_g$, if present (iFlag = 0 is returned), for the current target is returned in fnubar. There are *ncg* data points (see **ndfngroup**). If no fission data is present then iFlag = 1 is returned. This data only exist for neutron as an incident particle (i.e., **ndf1**) and only for some of its targets.

Related routines: ndfiso: ndffism, ndffsp

ndfflxw

Purpose:
Returns the collapsed flux weights for the $l = 0$ Legendre order.

FORTRAN Calling:
Call ndfflxw(flux)

Real*8	flux(ncg)	o	N/A	$l = 0$ collapsed flux weights

The $l = 0$ Legendre order, collapsed flux weights are returned in flux. This routine can only be called after **ndfgroup** has been called. This is equivalent to calling **ndfflxw_l** from FORTRAN as,

```
Call ndfflxw\_l( 0, flux )
```

(see routine **ndfflxw_l**).
Related routines: ndfgroup: ndfflxw_l

ndfflxw_l

Purpose:
Returns the collapsed flux weights for the requested Legendre order.

FORTRAN Calling:
Call ndfflxw_l(l, flux)

Integer	l	i	N/A	Requested Legendre order
Real*8	flux(ncg)	o	N/A	l^{th} Legendre order collapsed flux weights

The l^{th} Legendre order, collapsed flux weights are returned in flux. There are *ncg* data points (see **ndfngroup**). This routine can only be called after **ndfgroup** has been called.

Fatal Message(s): Prints a fatal message if the requested l-order is invalid.

Related routines: ndfgroup: ndfflxw

ndffsp

Purpose:
Returns the normalized fission spectrum for the current target.

FORTRAN Calling:
Call ndffsp(FissSpec)

Real*8	FissSpec(ncg)	o	N/A	Normalized fission spectrum

The lowest energy group of the incident neutron of the fission matrix, if present, for the current target is normalized and returned in FissSpec. There are *ncg* data points (see **ndfngroup**). If no fission data is present then this routine prints a fatal message. If the lowest energy data are all zero than FissSpec is filled with *ncg* ones. This data only exist for neutron as an incident particle (i.e., **ndf1**) and only for some of its targets.

Fatal Message(s): Prints a fatal message if the current target has no fission matrix.

Related routines: ndfiso: ndfifsp, ndffism, ndffisx

ndfgid

Purpose:
Returns the group id of the uncollapsed energy boundaries of the incident particle.

FORTRAN Calling:
gid = ndfgid( )

Integer	gid	f	N/A	Uncollapsed group id for the incident particle

When an **ndfy$_{\mathbf{i}}$** file is generated the data is grouped along the energy of the incident particle. The boundaries for energy grouping are specified using an energy group found in the bdfls file. The id of this energy group is returned by **ndfgid**.

Related routines: ndfinit: ndfgp

ndfgp

Purpose:
Returns the uncollapsed energy boundaries of the incident particle.

FORTRAN Calling:
Call ndfgp(gb)

Real*8	gb(ng + 1)	o	MeV	Uncollapsed energy boundaries for the incident particle

The uncollapsed energy boundaries of the incident particle are returned. There are $ng + 1$ data points returned since ng groups requires $ng + 1$ boundaries.

Related routines: ndfbuff: ndfgid

ndfgroup

Purpose:
Provides the user supplied information needed for collapsing of the data.

FORTRAN Calling:
Call ndfgroup(cgb, ncg, fid)

Real*8	cgb(ncg + 1)	i	MeV	Boundaries for the collapsed group
Integer	ncg	i	N/A	Number of collapsed groups
Integer	fid	i	N/A	bdfls id of the flux to use for collapsing

The ndf routines allow collapsing of the data to a smaller energy group using a flux from the bdfls file for weighting of the uncollapsed group data. Collapsing is initiated by calling **ndfgroup**. This smaller collapsing energy group must be a subset of the original energy group; that is, every boundary in the smaller group must have a corresponding boundary in the original group. The smaller group can be a user specified group that is a subset of the original group. The user can use **ndfidog** to read in a group from a bdfls file. The flux used to weight the collapsing is given by the fid argument. If fid = 0 then the flux in the **ndfy$_i$** file is used; otherwise, the flux with id = fid in the bdfls file is used.

Fatal Message(s): Prints a fatal message if the collapsing group is not a subset of the group used to generate the file, if the bdfls file cannot be found, or if the requested flux is not found in the bdfls file.

Related routines: ndfbuff: ndfidog, ndfngroup

ndfidog

Purpose:
Reads an energy group from a bdfls file.

FORTRAN Calling:
Call ndfidog(gid, gb, ngs, Dummy)

Integer	gid	i	N/A	Id of the energy group to input from the bdfls file
Real*8	gb(ngs + 1)	o	MeV	Boundaries of returned group
Integer	ngs	o	N/A	Number of returned groups
Real*8	Dummy	u	N/A	Not used

The group with id = gid is read in from the bdfls file and returned in gb. There are $ngs + 1$ data points returned in gb. This routine prints a fatal message if the bdfls file cannot be opened or the requested group id is not found in the bdfls file.

Fatal Message(s): Prints a fatal message if the requested group is not found in the bdfls file, or if the bdfls file cannot be opened.

Related routines: None: ndfgroup

ndfifsp

Purpose:
Returns a flag indicating whether or not fission data is present.

FORTRAN Calling:
Flag = ndfifsp( )

Integer	Flag	f	N/A	Flag indicating whether or not ndffsp will print a fatal message

This routine allows the user to determine whether or not fission data is present for the current target, without calling **ndffsp** which will print a fatal message if no fission spectrum data is present for the current target. It returns -1 if no fission spectrum data is present, 0 if the fission spectrum for the lowest incident-energy-group is zero, and 1 otherwise.

Related routines: ndfiso: ndffsp

ndfinfo

Purpose:
Returns the path of the opened **ndfy$_i$** file.

FORTRAN Calling:
Call ndfinfo(path)

Character*(*)	path	o	N/A	Path of the opened **ndfy$_i$** file

The path of the opened **ndfy$_i$** file is accessible by calling **ndfinfo**. Calling this routine is meaningful only after an **ndfy$_i$** file has been opened by calling **ndfinit**. The string is truncated if it is longer than the length of path.

Related routines: ndfinit:

ndfinit

Purpose:
Opens an **ndfy$_i$** file.

FORTRAN Calling:
Call ndfinit(yi, Name, Date, ReqdMem)

Integer	yi	i	N/A	Id of the incident particle
Character*(*)	Name	o	N/A	File name (e.g., '**ndf1**')
Integer	Date	o	N/A	Date in the **ndfy$_i$** file
Integer	ReqdMem	o	N/A	Memory in 8-byte words needed by ndf routines

This routine opens a **ndfy$_i$** file for input, it must be called before any other ndf routine, except **ndfaccess** and **ndfidog**. The file to be opened is determined by the first parameter which specifies the incident particle's id (e.g., $yi = 1$ for neutron). The name of the file to be opened is 'ndf' + { yi

converted to a character } (e.g., for $yi = 3$ the file name is **ndf3**). See **ndfaccess** for the directories **ndfinit** searches to find the **ndfy$_i$** file. *Name* : is the 4 character string name of the file (e.g., '**ndf5**'). Date is a date in the form YYMMDD (e.g., 991031 for 31-Oct-1999) stored in the file. Originally this date was the day on which the file was processed. Currently, as of about 1-Oct-2000, the date is used to uniquely identify a file. Dates in files generated before about 1-Oct-2000 are not guaranteed to be unique. ReqdMem is the amount of memory in 8-byte words that the user must allocate and pass to **ndfbuff**. This memory is used internally by the ndf accessing routines to read in the data and for work space. Most ndf routines cannot be called until **ndfinit**, **ndfbuff** and then **ndfiso** are called.

Fatal Message(s): Prints a fatal message if yi is invalid, if the file could not be opened, or if a file is already opened.

Related routines: None: ndfaccess, ndfbuff, ndfinfo, ndfclose

ndfiso

Purpose:

Selects a target from the opened **ndfy$_i$** file.

FORTRAN Calling:

Call ndfiso(ZA, Flag)

Integer	ZA	i	N/A	ZA = (1000 × Z + A) of the target to select
Integer	Flag	o	N/A	Flag indicating if target was found in file

This routine is used to select the target for which data is requested. Most ndf routines cannot be called until **ndfinit**, **ndfbuff** and then **ndfiso** are called. If the requested ZA is found then Flag = 0, otherwise Flag = 1. Only one target is selected at a time. Whenever **ndfiso** is called the previous target's data, if any, is lost. For the **ndf7** file (i.e., $y_i = 7$) all ZAs except 99120, fission target, are converted to 1000 × Z (i.e., natural isotope) before the target is selected (e.g., ZA = 6012 if converted to 6000). The assumption being that all isotopes for a given Z have the same electron shell structure and therefore the same photon interaction characteristics.

Related routines: ndfbuff: ndfistab, ndfnistab

ndfistab

Purpose:

Returns the list of targets in the opened **ndfy$_i$** file.

FORTRAN Calling:

Call ndfistab(ZAList, nZAs)

Integer	ZAList(nZAs)	o	N/A	List of targets (ZAs)
Integer	nZAs	o	N/A	Number of ZAs in ZAList

This routine returns the list of all targets in the opened **ndfy$_i$** file. The number of ZAs returned can also be obtained by calling **ndfnistab**.

Related routines: ndfbuff: ndfiso, ndfnistab

ndfmxorder

Purpose:

Returns the maximum Legendre order in the opened **ndfy$_{\mathbf{i}}$** file.

FORTRAN Calling:

Call ndfmxorder(lMax)

Integer	lMax	o	N/A	Maximum Legendre order

Each **ndfy$_{\mathbf{i}}$** file contains some data represented as a Legendre expansion (e.g., interaction transfer matrix). The expansions are truncated, and the last Legendre order present for the current target can be obtained by calling **ndfmxorder**. Typically, the maximum order is the same for all targets.

Related routines: ndfiso: ndfcorrec, ndfmxorder_tc, ndfsig, ndftrcorr

ndfmxorder_tc

Purpose:

Returns the maximum Legendre order allowed by the current transport correction method.

FORTRAN Calling:

lm = ndfmxorder_tc( )

Integer	lm	f	N/A	Maximum Legendre order allowed by transport correction method

Each **ndfy$_{\mathbf{i}}$** file contains some data represented as a Legendre expansion (e.g., interaction transfer matrix). The Legendre expansion is truncated, and the last Legendre order present for the current target can be obtained by calling **ndfmxorder**. This maximum Legendre order can be used when the transport correction method is "No correction" or the "Legacy LLNL correction". For the "Pendlebury/Underhill correction" and the "Ferguson correction" the maximum Legendre order is lMax - 1. This routine returns the correct maximum Legendre order allowed by the current transport correction method.

Related routines: ndfiso: ndfcorrec, ndfmxorder, ndfsig, ndftrcorr

ndfncorrec

Purpose:

Returns the length of data in 'double' words required by **ndfcorrec**.

FORTRAN Calling:

Call ndfncorrec(len)

Integer	len	o	N/A	Length of data required by **ndfcorrec**

The length of the data returned by **ndfcorrec** can be obtained by calling this routine. This can be useful if memory for the transport-correcting cross-section returned by **ndfcorrec** must be allocated before calling **ndfcorrec**. Calling **ndfncorrec** is equivalent to the following FORTRAN code,

```
Call ndfmxorder( lMax )
len = ( lMax + 1 ) * ndfngroup( )
```

For e_ndfPendlebury and e_ndfFerguson transport correction methods, the amount of useful data is 'lMax * ndfngroup( )'; however, the number returned by **ndfncorrec** is the requirement amount of double' words required by **ndfcorrec**, which is always '(lMax + 1) * ndfngroup( )'.

Related routines: ndfiso: ndfcorrec, ndfngroup, ndfmxorder

ndfngroup

Purpose:
Returns the number of collapsed groups.

FORTRAN Calling:
n = ndfngroup( )

Integer	n	o	N/A	Number of collapsed groups

If **ndfgroup** has been called then the number of groups for the incident particle's collapsed group is returned. Else, the number of groups for the incident particle's uncollapsed group is returned.

Related routines: ndfinit: ndfgroup, ndfngroups

ndfngroups

Purpose:
Returns the number of uncollapsed groups.

FORTRAN Calling:
Call ndfngroups(ng)

Integer	ng	o	N/A	Number of uncollapsed groups

The number of groups for the incident particle's uncollapsed group is returned.

Related routines: ndfinit: ndfgroup, ndfngroup

ndfnistab

Purpose:
Returns the number of targets in the opened **ndfy$_i$** file.

FORTRAN Calling:

nZAs = ndfnistab( )

Integer	nZAs	f	N/A	Number of targets in the opened **ndfy$_i$** file

The number of targets in the opened **ndfy$_i$** file can be obtained by calling this routine. This can be useful if memory for the target list must be allocated before calling **ndfistab**.

Related routines: ndfbuff: ndfiso, ndfistab

ndfnmaxgps

Purpose:

Returns the largest group size used in processing the opened **ndfy$_i$** file.

FORTRAN Calling:

Call ndfnmaxgps( )

'return value'	f	N/A	Largest group size used in processing the opened **ndfy$_i$** file

The groups used in grouping the various transportable particles can be difference and can have difference sizes. This routine returns the size of the largest group and may be useful if memory must be allocated for a production matrix.

Related routines: ndfbuff: ndfngroup, ndfngroups, ndfpmat, ndfppmatrix

ndfnppyos

Purpose:

Returns the number of transportable outgoing particles for which a production transfer matrix exist for the current target.

FORTRAN Calling:

nyos = ndfnppyos( )

Integer	nyos	f	N/A	Number of transportable outgoing particles

The number of transportable outgoing particles with a production transfer matrix for the current target can be obtained by calling this routine. This can be useful if memory for the transportable outgoing particle with production transfer matrices list must be allocated before calling **ndfppyos**. The difference between **ndfnyos** and **ndfnppyos** is 1, since **ndfnyos** includes the incident particle; whereas, **ndfnppyos** does not (as the incident particle's transfer matrix, called the interaction transfer matrix, is obtained using **ndfsig** instead of **ndfpmat**).

Related routines: ndfiso: ndfppyos, ndfnyos, ndfyos, ndfpmat

ndfnprod

Purpose:

Returns the number of outgoing particles produced for the requested reaction for the current target.

FORTRAN Calling:
n = ndfnprod(C)

Integer	C	i	N/A	C-value for the requested reaction
Integer	n	f	N/A	Number of outgoing particles produced for reaction C

This routine can be useful if memory for the outgoing produced particle's ZA and multiplicity lists must be allocated before calling **ndfprod**. See **ndfprod** for more details. If the requested C-value is not in the reaction list for the current target then 0 is returned. Note, **ndfprod** returns a list of all particles produced and not just the transportable ones.

Related routines: ndfiso: ndfprod

ndfnreact

Purpose:
Returns the number of reactions for the current target.

FORTRAN Calling:
n = ndfnreact( )

Integer	n	f	N/A	Number of reactions for the current target

For each target there is a list of reactions. A reaction is a unique combination of C-, S-, Q-, X1-, X2-, X3-, and Q_{eff}-values (see **Reaction specific information** in section 5.2). This routine returns the number of unique reactions for the current target. This can be useful if memory for the reaction list must be allocated before calling **ndfreact**.

Related routines: ndfiso: ndfreact

ndfnrxs

Purpose:
Returns the number of reactions of type C for the current target.

FORTRAN Calling:
n = ndfnrxs(C)

Integer	C	i	N/A	C-value for the requested reaction
Integer	n	f	N/A	Number of reactions of type C

For each target there is a list of reactions. A reaction is a unique combination of C-, S-, Q-, X1-, X2-, X3-, and Q_{eff}-values (see **Reaction specific information** in section 5.2). This routine returns the number of reactions of type C for the current target.

Related routines: ndfiso: ndfrxs, ndfrxslist, ndfrxslevel

ndfnyos

Purpose:
Returns the number of transportable outgoing particles for the current target.

FORTRAN Calling:
n = ndfnyos( )

Integer	n	f	N/A	Number of transportable outgoing particles

For each target there is a list of possible transportable outgoing particles. For example, with neutron as incident particle, a specific target in a **ndfy$_i$** file may have neutrons, protons, deuterons and gammas as transportable outgoing particles. In this case **ndfnyos** would return 4. (See routine **ndfnppyos** for more details.)

Related routines: ndfiso: ndfyos, ndfed, ndfppyos, ndfnppyos

ndfpmat

Purpose:
Returns the collapsed production transfer matrix and the corrected, collapsed, deposited energy.

FORTRAN Calling:
Call ndfpmat(matrix, ed, yo, ch, nch)

Real*8	matrix(nch × ncg)	o	barn	$y_i \rightarrow y_o$ production transfer matrix for $y_i \neq y_o$
Real*8	ed(ncg)	o	MeV-barn	Corrected energy deposited by yo
Integer	yo	i	N/A	Requested outgoing particle ($y_o \neq y_i$)
Real*8	ch(nch + 1)	i	MeV	Outgoing particle's collapsed group boundaries
Integer	nch	i	N/A	Number of groups in ch

This routine returns the collapsed production transfer matrix (i.e., $y_i \rightarrow y_o$ for $y_i \neq y_o$) and the corrected deposited energy for y_o = yo. The outgoing particle's deposition energy is collapsed to *ch*. To obtain ncg call **ndfngroup**. If $iecflg = 3$, see **ndfyo_info**, the deposited energy returned by **ndfed** is corrected so that the outgoing particle's number and energy are conserved. This corrected energy is returned in ed. This routine returns all zeros in matrix and energy if the requested *yo* is not present for the current target. Use **ndfsig** or **ndftransfer** to obtain the interaction transfer matrix (i.e., $y_i \rightarrow y_o$ for $y_i = y_o$).

Fatal Message(s): Prints a fatal message if yo is not present in the particle directory of the global data section.

Related routines: ndfiso: ndfnppyos, ndfppyos, ndfyo_gid, ndfed, ndfppmatrix, ndfsig

ndfppmatrix

Purpose:

Returns the uncollapsed production transfer matrix for the requested outgoing particle for the current target.

FORTRAN Calling:

Call ndfppmatrix(yo, matrix, nh)

Integer	yo	i	N/A	Requested outgoing particle ($y_o \neq y_i$)
Real*8	matrix(nh × ng)	o	barn	$y_i \rightarrow y_o$ production transfer matrix for $y_i \neq y_o$
Integer	nh	o	N/A	Number of groups representing the outgoing particle's energy groups

The uncollapsed production transfer matrix (i.e., $y_i \rightarrow y_o$ for $y_i \neq y_o$) for the requested transportable outgoing particle for the current target is returned. See **ndfppyos** for a list of transportable outgoing particle for the current target.

Fatal Message(s): Prints a fatal message if the requested transportable outgoing particle is not present.

Related routines: ndfiso: ndfpmat, ndfnppyos, ndfppyos

ndfppyos

Purpose:

Returns the list of transportable outgoing particles for which there are production transfer matrices for the current target.

FORTRAN Calling:

Call ndfppyos(yoList, nyos)

Integer	yoList(nyos)	o	N/A	List of outgoing particles for the current target
Integer	nyos	o	N/A	Number of outgoing particles returned

For each target there is a list of possible transportable outgoing particles. For example, with neutron as incident particle, a specific target in a **ndfy$_\mathbf{i}$** file may have neutrons, protons, deuterons and gammas as transportable outgoing particles. In this case **ndfyos** would return $yo = \{ 2, 3, 7 \}$ and $n = 3$, since the transfer matrix for $y_o = y_i$ is not included. The transfer matrix for $y_o = y_i$, called the interaction transfer matrix, is obtained using **ndfsig** or **ndftransfer**.

Related routines: ndfiso: ndfnppyos, ndfpmat, ndfppmatrix

ndfprod

Purpose:

Returns the ZA and multiplicity lists for particles produced by the requested reaction for the current target.

FORTRAN Calling:

Call ndfprod(C, n, ZAList, MList)

Integer	C	i	N/A	C-value for the reaction
Integer	n	o	N/A	Number of outgoing particles produced for reaction C
Integer	ZAList(n)	o	N/A	List of ZAs
Integer	MList(n)	o	N/A	List of multiplicities

For each incident particle, target and reaction combination in a **ndfy$_i$** file, there is a list of outgoing particles produced and their multiplicities. For example, when a neutron is incident on ^{239}Pu, ZA = 94239, a possible reaction is two neutrons out, C = 12:

$$n + {}^{239}Pu \rightarrow n + n + {}^{238}Pu \quad .$$

For this case, **ndfprod** will return n = 2, ZAList = {1, 94238} and MList = {2, 1}. This means that two neutron (i.e., ZAList(1) = 1 and MList(1) = 2) and one ^{238}Pu (i.e., ZAList(2) = 94238 and MList(2) = 1) are produced. Note, in this reaction there is no mention of the gammas produced as they are not included in the particle production list. Information about the gammas may be given in the production transfer matrix and the energy deposited to the outgoing particle (see **ndfppyos**, **ndfpmat** and **ndfed**).

For neutrons, the multiplicity may be zero. In this case, the outgoing neutron information is obtained by calling **ndffisx**. For example, calling **ndfprod** for a typical fission reaction, C = 15, of target ^{239}Pu may return n = 2, ZAList = {1, 99120} and MList = {0, 2} (ZA = 99120 is the special ZA code for prompt fission products and ZA = 99125 is the special ZA code for delayed fission products). In this example 2 prompt fission products are produced and MList(1) = 0 implies that **ndffisx** must be called to get the multiplicity data for neutrons.

Fatal Message(s): Prints a fatal message if the requested C-value is not in the reaction list for the current target.

Related routines: ndfiso: ndfnprod

ndfreact

Purpose:

Returns a list of C-values for the reactions for the current target.

FORTRAN Calling:

Call ndfreact(CList, n)

Integer	CList(n)	o	N/A	C-value for each reaction
Integer	n	o	N/A	Number of reactions returned

For each target there is a list of reactions. A reaction is a unique combination of C-, S-, Q-, X1-, X2-, X3-, and Q_{eff}-values (see **Reaction specific information** in section 5.2). This routine returns the C-value for each reaction and the number of reactions for the current target. For example, for one target in a **ndf1** file the following was returned; CList = { 10, 11, 11, 11, 11, 11, 11, 11, 12, 13, 14, 15, 46, 46 } and n = 14.

Related routines: ndfiso: ndfnreact, ndfrxs, ndfrxslevel, ndfrxslist

ndfrxs

Purpose:

Returns the requested reaction's collapsed cross-section data for the current target.

FORTRAN Calling:

Call ndfrxs(C, cs, Q, n)

Integer	C	i	N/A	C-value of the requested reaction
Real*8	cs(ncg)	o	barn	Collapsed cross-section for this reaction
Real*8	Q	o	MeV	Q-value of reaction
Integer	n	i	N/A	Number of reactions with this C-value to include

For each unique combination of C-, S-, Q-, X1-, X2-, X3-, and Q_{eff}-values (see **Reaction specific information** in section 5.2), for a given target in a **ndfy$_i$** file, the cross-section is stored. For a specific reaction type, specified by the C-value, **ndfrxs** will sum the cross-sections for the first n of these reactions and return it in cs. If $n = 0$ then all reactions with this C-value are summed. The Q-value, (i.e. Q and not Q_{eff} in the list above) of the last reaction included in the sum is returned in Q. To obtain ncg call **ndfngroup**.

Related routines: ndfiso: ndfnrxs, ndf, ndfrxslevel, ndfrxslist

ndfrxslevel

Purpose:

Returns S-, Q-, X1-, X2-, X3-, Q_{eff}-values and the collapsed cross-section for the requested level of the requested reaction for the current target.

FORTRAN Calling:

err = ndfrxslevel(C, L, S, Q, X1, X2, X3, Qeff, cs)

Integer	C	i	N/A	C-value for the requested reaction
Integer	L	i	N/A	Level for requested C-value
Integer	S	o	N/A	S-value for reaction
Real*8	Q	o	MeV	Q-value for reaction
Real*8	X1	o	Varies	X1-value for reaction
Real*8	X2	o	Varies	X2-value for reaction
Real*8	X3	o	Varies	X3-value for reaction
Real*8	Qeff	o	MeV	Q_{eff}-value for reaction
Real*8	cs(ncg)	o	barns	collapsed cross-section for reaction
Integer	err	f	N/A	1 if requested C and L are present, 0 otherwise

The meanings and units for X1-, X2-, and X3-values depend on the S-value. There are *ncg* data points return in cs (see **ndfngroup**). If no reaction for the requested C-value and level exist then err = 0 is returned. L is an index for the level in the range 1 to the value returned by **ndfnrxs**.

Related routines: ndfiso: ndfrxs, ndfnrxs, ndfrxslist

ndfrxslist

Purpose:

Returns a list of S-, Q-, X1-, X2-, X3- and Q_{eff}-values for the requested reaction for the current target.

FORTRAN Calling:

n = ndfrxslist(C, S, Q, X1, X2, X3, Qeff)

Integer	C	i	N/A	C-value for the requested reaction
Integer	S(n)	o	N/A	List of S-values for reaction
Real*8	Q(n)	o	MeV	List of Q-values for reaction
Real*8	X1(n)	o	Varies	List of X1-values for reaction
Real*8	X2(n)	o	Varies	List of X2-values for reaction
Real*8	X3(n)	o	Varies	List of X3-values for reaction
Real*8	Qeff(n)	o	MeV	List of Q_{eff}-values for reaction
Integer	n	f	N/A	Number of reactions of type C

The meanings and units for X1-, X2-, and X3-values depend on the S-value. If no reaction for the requested C-value exist then n = 0 is returned. As example, if **ndfrxslist** is called with C = 42 for the data of Table 4 then the following is returned: S = { 1, 1 }, Q = { -10.44, -10.44 }, X1 = { 0., 0.478 }, X2 = { 0., 0. }, X3 = { 0., 0. }, Qeff = { -10.439, -10.917 } and n = 2. If memory must be allocated before calling **ndfrxslist**, the routine **ndfnrxs** can be called before **ndfrxslist** to obtain n.

Related routines: ndfiso: ndfrxs, ndfnrxs, ndfrxslevel

ndfsig

Purpose:

Returns the collapsed total cross-section (transport corrected), the collapsed interaction transfer matrix ($y_i \rightarrow y_o$ for $y_i = y_o$ transport corrected), the collapsed energy deposited by $y_o = y_i$ and the collapsed transport correcting cross-section for the requested Legendre order for the current target.

FORTRAN Calling:

Call ndfsig(tcs, ed, tm, tc, l)

Real*8	tcs(ncg)	o	barn	Collapsed transport corrected total cross-section
Real*8	ed(ncg)	o	MeV-barn	Collapsed corrected energy deposited by incident particle type
Real*8	tm(ncg × ncg × (l + 1))	o	barn	Collapsed transport corrected interaction transfer matrix
Real*8	tc(ncg)	o	barn	Collapsed transport correcting cross-section
Integer	l	i	N/A	Requested Legendre order

This routine first calls other ndf routines to calculate the transport correcting cross-section for Legendre order $l + 1$. Then, the transport corrected, total cross-section and the transport corrected interaction transfer matrix ($y_i \rightarrow y_o$ for $y_i = y_o$) are calculated. All interaction transfer matrices for

Legendre order 0 to l inclusive are transport corrected and returned. If $iecflg = 3$, see **ndfyo_info**, the deposited energy returned by **ndfed** is corrected so that outgoing particle number and energy are conserved. (see **ndfpmat**) To obtain ncg call **ndfngroup**.

Fatal Message(s): Prints a fatal message if the request l-order is invalid.

Related routines: ndfiso: ndftotal, ndfed, ndfyo_info

ndfsp

Purpose:

Returns the collapsed group speeds for the opened **ndfy$_i$** file.

FORTRAN Calling:

Call ndfsp(speeds)

Real*8	speeds(ncg)	o	cm/sh	Collapsed group speeds for the opened **ndfy$_i$** file

This routine returns the collapsed group speeds for the opened **ndfy$_i$** file. There are *ncg* data points returned (see **ndfngroup**).

Related routines: ndfinit:

ndftotal

Purpose:

Returns the collapsed uncorrected total cross-section for the current target.

FORTRAN Calling:

Call ndftotal(tcs)

Real*8	tcs(ncg)	o	barn	Collapsed uncorrected total cross-section for the current target

This routine returns the collapsed uncorrected total cross-section for the current target. There are *ncg* data points returned (see **ndfngroup**).

Related routines: ndfiso: ndfsig

ndftransfer

Purpose:

Returns the uncollapsed, uncorrected interaction transfer matrix for the requested Legendre order for the current target.

FORTRAN Calling:

Call ndftransfer(tm, l)

Real*8	tm(ng × ng)	o	barn	Uncollapsed, uncorrected interaction transfer matrix for Legendre order l
Integer	l	i	N/A	Requested Legendre order

The uncollapsed, uncorrected interaction transfer matrix (i.e., $y_i \rightarrow y_i$ for $y_i = y_o$) for the requested Legendre order for the current target is returned.

Fatal Message(s): Prints a fatal message if the requested Legendre order is not present.

Related routines: ndfiso: ndfpmat, ndfmxorder

ndftrcorr

Purpose:

Sets the desired transport correction method.

FORTRAN Calling:

Call ndftrcorr(c)

Integer	c	i	N/A	Desired transport correction method

The ndf routines allow for 4 difference transport correction methods. This routine sets the transport correction method to c, as described in the following table.

c	Maximum l	Correction method
0	lMax	No correction
1	lMax - 1	Pendlebury/Underhill correction (default)
2	lMax	Legacy LLNL correction
3	lMax - 1	Ferguson correction

In this table the second column is the maximum Legendre order the user can request. The transport correcting cross-section can be calculated to one greater than this Legendre order. For example, if lMax = 3 and the correction method is Pendlebury/Underhill then **ndfsig** can only be called for l-order up to 2. This routine must be called after each call to **ndfinit**, as **ndfinit** resets the internal flag to the default.

Fatal Message(s): Prints a fatal message if an invalid transport correct method is requested.

Related routines: ndfinit:

ndfwsp

Purpose:

Returns the collapsed normalized $l = 0$ flux.

FORTRAN Calling:

Call ndfwsp(w)

Real*8	w(ncg)	o	N/A	Collapsed normalized $l = 0$ flux

This routine returns the collapsed normalized $l = 0$ flux. There are *ncg* data points returned (see **ndfngroup**). It can only be called after **ndfgroup** has been called.

Fatal Message(s): Prints a fatal message if the $l = 0$ flux is all zeros.

Related routines: ndfgroup:

ndfyo

Purpose:
Returns the j^{th} particle's id from the particle directory of the global data section.

FORTRAN Calling:
yo = ndfyo(j)

Integer	j	i	N/A	The index of the requested particle's id
Integer	yo	f	N/A	The j^{th} particle id from particle directory

This routine can be used to loop over the particle directory of the global data section. To step through the particles in the particle directory, start with $j = 1$ (FORTRAN indexing is used) and increment it in a while loop until *yo* = -1 is returned. For example,

```
n = 0
yo = 0
Do While( yo .ne. -1 )
    yo = ndfyo( n + 1 )          ! FORTRAN indexing.
    If( yo .ne. -1 ) n = n + 1
EndDo
Print *, 'n = ', n
```

prints the number of particles in the particle directory of the global data section.

Related routines: ndfbuff:

ndfyos

Purpose:
Returns a list of transportable outgoing particles with energy deposit data for the current target.

FORTRAN Calling:
Call ndfyos(yo, n)

Integer	yo(n)	o	N/A	List of transportable outgoing particles with energy deposit data
Integer	n	o	N/A	Number of transportable outgoing particles with energy deposit data

For each target there is a list of possible transportable outgoing particles. For example, with neutron as incident particle, a specific target in a $\mathbf{ndfy_i}$ file may have neutrons, protons, deuterons and gammas as transportable outgoing particles. In this case **ndfyos** would return *yo* = { 1, 2, 3, 7 } and $n = 4$.

Related routines: ndfiso: ndfnyos, ndfed, ndfppyos, ndfnppyos

ndfyo_gid

Purpose:

Returns the group id for the requested outgoing particle.

FORTRAN Calling:

gid = ndfyo_gid(yo)

Integer	yo	i	N/A	The requested outgoing particle
Integer	gid	f	N/A	The group id used for the outgoing particle

The uncollapsed production transfer matrix is grouped in incident and outgoing particle energies. The uncollapsed grouping for the outgoing particle energies is contained in the bdfls file and has the group id = gid.

Related routines: ndfiso:

ndfyo_info

Purpose:

Returns the number of energy groups, nh, and the conservation flag, iecflg, for the requested transportable outgoing particle.

FORTRAN Calling:

Call ndfyo_info(yo, nh, iecflg)

Integer	yo	i	N/A	Requested transportable outgoing particle
Integer	nh	o	N/A	Number of groups representing the transportable outgoing particle's energy grouping
Integer	iecflg	o	N/A	Particle and/or Energy conservation flag

This routine returns the number of groups representing the outgoing particle's energy grouping in the uncollapsed production transfer matrix. If outgoing particle's energy grouping is of size nh then in FORTRAN the uncollapsed production transfer matrix would be equivalent to the declaration,

`Real*8 matrix( nh, ng )` .

However, ndf treats all matrices as one dimensional vectors. The routine also returns $iecflg$ which is a flag indicating the conservation mode used to process the production transfer matrix for this outgoing particle. The meaning of $iecflg$ is as follows,

iecflg	Description
0	Conserve particles
1	Conserve energy, used for gammas ($y_o = 7$)
2	Conserve energy for the $l = 0$ Legendre order
3	Conserve particles and energy

If the requested yo is not present then nh = -1 and iecflg = -1.

Related routines: ndfbuff: ndfpmat, ndfppmatrix, ndfppyos

7.5 Summary of C routines

ndfcaccess	To select a **ndfy$_i$** data file other then the default.
ndfcatw	Returns the mass for the current target in AMU.
ndfcbuff	Provide memory allocated by the user to the ndf routines.
ndfcclose	Closes the opened **ndfy$_i$** file.
ndfccorrec	Returns the transport-correcting cross-section for the current target.
ndfced	Returns the deposited energy for the requested transportable outgoing particle for the current target.
ndfcemax	Returns the total available energy for the current target.
ndfcep	Returns the production energy for the current target.
ndfcfism	Returns the fission matrix for the current target.
ndfcfisx	Returns the fission $< \nu\sigma >_g$ for the current target.
ndfcflxw	Returns the collapsed flux weights for $l = 0$.
ndfcflxw_l	Returns the collapsed flux weights for the requested Legendre order.
ndfcfsp	Returns the normalized fission spectrum for the current target.
ndfcgid	Returns the group id of the uncollapsed energy boundaries of the incident particle.
ndfcgp	Returns the uncollapsed energy boundaries of the incident particle.
ndfcgroup	Provides the user supplied information needed for collapsing of the data.
ndfcidog	Reads an energy group from a bdfls file.
ndfcifsp	Returns a flag indicating whether or not fission data is present.
ndfcinfo	Returns the path of the opened **ndfy$_i$** file.
ndfcinit	Opens an **ndfy$_i$** file.
ndfciso	Selects a target from the opened **ndfy$_i$** file.
ndfcistab	Returns the list of targets in the opened **ndfy$_i$** file.
ndfcmxorder	Returns the maximum Legendre order in the opened **ndfy$_i$** file.
ndfcmxorder_tc	Returns the maximum Legendre order allowed by the current transport correction method.
ndfcncorrec	Returns the length of data in 'double' words required by **ndfcorrec**.
ndfcngroup	Returns the number of collapsed groups.
ndfcngroups	Returns the number of uncollapsed groups.
ndfcnistab	Returns the number of targets in the opened **ndfy$_i$** file.
ndfcnmaxgps	Returns the largest group size used in processing the opened **ndfy$_i$** file.
ndfcnppyos	Returns the number of transportable outgoing particles for which a production transfer matrix exist for the current target.
ndfcnprod	Returns the number of outgoing particles produced for the requested reaction for the current target.
ndfcnreact	Returns the number of reactions for the current target.
ndfcnrxs	Returns the number of reactions of type C for the current target.
ndfcnyos	Returns the number of transportable outgoing particles for the current target.
ndfcopen	Opens an **ndfy$_i$** file. Replaces **ndfcinit** and **ndfbuff**.
ndfcpmat	Returns the collapsed production transfer matrix and the corrected, collapsed,

	deposited energy
ndfcppmatrix	Returns the uncollapsed production transfer matrix for the requested outgoing particle for the current target.
ndfcppyos	Returns the list of transportable outgoing particles for which there are production transfer matrices for the current target.
ndfcprod	Returns the ZA and multiplicity lists for particles produced by the requested reaction for the current target.
ndfcreact	Returns a list of C-values for the reactions for the current target.
ndfcrxs	Returns the requested reaction's cross-section data for the current target.
ndfcrxslevel	Returns S-, Q-, X1-, X2-, X3-, Q_{eff}-values and the cross-section for the requested level of the requested reaction for the current target.
ndfcrxslist	Returns a list of S-, Q-, X1-, X2-, X3- and Q_{eff}-values for the requested reaction for the current target.
ndfcsig	Returns the total cross-section (transport corrected), interaction transfer matrix ($y_i \rightarrow y_o$ for $y_i = y_o$ transport corrected), energy deposited by $y_o = y_i$ and the transport correcting cross-section for the requested Legendre order for the current target.
ndfcsp	Returns the group speeds for the opened **ndfy$_i$** file.
ndfctotal	Returns the uncorrected total cross-section for the current target.
ndfctransfer	Returns the uncollapsed, uncorrected interaction transfer matrix for the requested Legendre order for the current target.
ndfctrcorr	Sets the desired transport correction method.
ndfcwsp	Returns the collapsed normalized $l = 0$ flux.
ndfcyo	Returns the j^{th} id from the particle directory of the global data section.
ndfcyos	Returns a list of transportable outgoing particles with energy deposit data for the current target.
ndfcyo_gid	Returns the group id for the requested particle type.
ndfcyo_iecflg	Returns the conservation flag, iecflg, for the requested transportable outgoing particle.
ndfcyo_nego	Returns the number of energy groups, nh, for the requested transportable outgoing particle.

7.6 C wrappers for the FORTRAN routines

ndfcaccess

C declaration:

void ndfcaccess(int yi, char *libnam, char *Version, char *grptype, char *subpath, int ls);

yi	i	N/A	Id of the incident particle
libnam	i	N/A	Evaluated data library name
Version	i	N/A	Version of evaluated data
grptype	i	N/A	Suffix added to file name
subpath	o	N/A	Returned string of directory
ls	i	N/A	Length of 'subpath' space (i.e. sizeof(*subpath))

This routine is a C wrapper for the FORTRAN routine **ndfaccess** with one additional argument. This argument, 'ls', informs **ndfcaccess** about the length of space in bytes reserved for 'subpath'. The following C codes demonstrates it usage.

```
char subpath[64];
ndfcaccess( 1, "endl", "current", "230", subpath, sizeof( subpath ) );
```

If there is not enough space in 'subpath' to hold the sub-directory name and the trailing NULL byte, then the name is truncated with a trailing NULL.

ndfcatw

C declaration:

double ndfcatw(void);

'return value'	f	AMU	Atomic mass in AMU for the current target

This routine is a C wrapper for the FORTRAN routine **ndfatw**.

ndfcbuff

C declaration:

void ndfcbuff(void *p);

p	i	N/A	Pointer to memory allocated by the user

This routine is a C wrapper for the FORTRAN routine **ndfbuff**.

ndfcclose

C declaration:

void ndfcclose(void);

This routine is a C wrapper for the FORTRAN routine **ndfclose**. It also frees memory allocated by other ndf C wrapper routines. For example, the memory allocated by **ndfcopen**. Note, **ndfcclose** only frees memory allocated by the C wrapper routines. If a user calls **ndfinit**, allocates memory and calls **ndfbuff**, then **ndfcclose** will not frees the memory allocated by the user, but will free other memory that the C wrapper routines may have allocated.

ndfccorrec

C declaration:

void ndfccorrec(double *tccs);

tccs[ncg × (lMax + 1)]	o	barn	The transport-correcting cross-section

This routine is a C wrapper for the FORTRAN routine **ndfcorrec**.

ndfced

C declaration:

void ndfced(int yo, double *ed);

yo	i	N/A	Id of the transportable outgoing particle
ed[ncg]	o	MeV-barn	outgoing particle's energy deposited

This routine is a C wrapper for the FORTRAN routine **ndfed**. Note that the arguments are in reverse order as compared to **ndfed**.

ndfcemax

C declaration:

void ndfcemax(double *em);

em[ncg]	o	MeV-barn	Available energy

This routine is a C wrapper for the FORTRAN routine **ndfemax**.

ndfcep

C declaration:

void ndfcep(double *ep);

ep[ncg]	o	MeV-barn	Production energy

This routine is a C wrapper for the FORTRAN routine **ndfep**.

ndfcfism

C declaration:

int ndfcfism(double *fm);

fm[ncg × ncg]	o	barn	Fission matrix
'return value'	f	N/A	Indicates whether or not data is present

This routine is a C wrapper for the FORTRAN routine **ndffism**. The error flag returned as the second argument in **ndffism** is returned as **ndfcfism**'s return value.

ndfcfisx

C declaration:

int ndfcfisx(double *fnubar);

fnubar[ncg]	o	barn	Fission nubar data
'return value'	f	N/A	Indicates whether or not data is present

This routine is a C wrapper for the FORTRAN routine **ndffisx**. The error flag returned as the second argument in **ndffisx** is returned as **ndfcfisx**'s return value.

ndfcflxw

C declaration:

void ndfcflxw(double *flux);

flux[ncg]	o	N/A	$l = 0$ collapsed flux weights

This routine is a C wrapper for the FORTRAN routine **ndfflxw**.

ndfcflxw_l

C declaration:

void ndfcflxw_l(int l, double *d);

l	i	N/A	Requested Legendre order
flux[ncg]	o	N/A	l order collapsed flux weights

This routine is a C wrapper for the FORTRAN routine **ndfflxw_l**.

ndfcfsp

C declaration:

void ndfcfsp(double *FissSpec);

FissSpec[ncg]	o	N/A	Normalized fission spectrum

This routine is a C wrapper for the FORTRAN routine **ndffsp**.

ndfcgid

C declaration:

int ndfcgid(void);

'return value'	f	N/A	Group id for the incident particle

This routine is a C wrapper for the FORTRAN routine **ndfgid**.

ndfcgp

C declaration:

void ndfcgp(double *gb);

gb[ng + 1]	o	MeV	Uncollapsed energy boundaries for the incident particle

This routine is a C wrapper for the FORTRAN routine **ndfgp**.

ndfcgroup

C declaration:

void ndfcgroup(int ncg, double *cgb, int fid);

ncg	i	N/A	Number of collapsed groups
cgb[ncg + 1]	i	MeV	Boundaries for the collapsed group
fid	i	N/A	bdfls id of the flux to uses for collapsing

This routine is a C wrapper for the FORTRAN routine **ndfgroup**. Note that the first two arguments are reversed as compared to the arguments of **ndfgroup**.

ndfcidog

C declaration:

int ndfcidog(int gid, double *gb);

gid	i	N/A	Id of the energy group to input from the bdfls file
gb['return value' + 1]	o	MeV	Boundaries of returned group
'return value'	f	N/A	Number of returned groups

This routine is a C wrapper for the FORTRAN routine **ndfidog**. Note that **ndfidog**'s third argument is returned as **ndfcidog**'s return value and **ndfidog**'s forth argument is not used by **ndfcidog**.

ndfcifsp

C declaration:

int ndfcifsp(void);

'return value'	f	N/A	Flag indicating whether or not ndffsp will print a fatal message

This routine is a C wrapper for the FORTRAN routine **ndffsp**.

ndfcinfo

C declaration:

void ndfcinfo(char *path, int ls);

path[ls]	o	N/A	Full path name of the current open **ndfy$_{\mathbf{i}}$** file.
ls	i	N/A	Length of path space (i.e., sizeof(*path)).

This routine is a C wrapper for the FORTRAN routine **ndfinfo** with one additional argument. This argument, 'ls', informs **ndfcinfo** about the length of space in bytes reserved for 'path'. The following C codes demonstrates it usage.

```
char subpath[64];
ndfcinfo( path, sizeof( path ) );
```

The path is truncated with a trailing NULL if path is not long enough.

ndfcinit

C declaration:

int ndfcinit(int yi, char *Name, int *ReqdMem);

yi	i	N/A	Id of the incident particle
Name[5]	o	N/A	File name (e.g., "**ndf1**")
ReqdMem	o	N/A	Memory in 8-byte words needed by ndf routines
'return value'	f	N/A	Date in the **ndfy$_i$** file

This routine is a C wrapper for the FORTRAN routine **ndfinit**. Note that **ndfinit**'s third argument is **ndfcinit**'s return value and **ndfinit**'s forth argument is **ndfcinit**'s third argument. This routine and **ndfcbuff** can, and probably should, be replaced by **ndfcopen**.

ndfciso

C declaration:

int ndfciso(int ZA);

ZA	i	N/A	ZA = (1000 × Z + A) of the target to select
'return value'	f	N/A	Flag indicating if target was found in file

This routine is a C wrapper for the FORTRAN routine **ndfiso**. The error flag returned as the second argument in **ndfiso** is returned as **ndfciso**'s return value.

ndfcistab

C declaration:

int ndfcistab(int **ZAList);

(*ZAList)['return value']	o	N/A	List of targets (ZAs)
'return value'	f	N/A	Number of ZAs in ZAList

This routine is a C wrapper for the FORTRAN routine **ndfistab**. This routine allocates the memory for the ZA list, calls **ndfistab**, and then returns a pointer to the allocated memory in 'ZAList'. The number of targets in 'ZAList' is returned as **ndfcistab**'s returned value. The user must not free the allocated memory, as this is done by **ndfcclose**. Multiple calls to **ndfcistab** are allowed.

ndfcmxorder

C declaration:

int ndfcmxorder(void);

'return value'	f	N/A	Maximum Legendre order

This routine is a C wrapper for the FORTRAN routine **ndfmxorder**. Note that **ndfmxorder**'s first argument is returned as **ndfcmxorder**'s return value.

ndfcmxorder_tc

C declaration:
int ndfcmxorder_tc(void);

'return value'	f	N/A	Maximum Legendre order allowed by transport correction method

This routine is a C wrapper for the FORTRAN routine **ndfmxorder_tc**.

ndfcncorrec

C declaration:
int ndfcncorrec(void);

'return value'	f	N/A	Length of data returned by **ndfccorrec**

This routine is a C wrapper for the FORTRAN routine **ndfncorrec**.

ndfcngroup

C declaration:
int ndfcngroup(void);

'return value'	f	N/A	Number of collapsed groups

This routine is a C wrapper for the FORTRAN routine **ndfngroup**. If collapsing has been initiated then the number of collapsed groups for the incident particle is returned. Else, the number of uncollapsed groups is returned.

ndfcngroups

C declaration:
int ndfcngroups(void);

'return value'	f	N/A	Number of uncollapsed groups

This routine is a C wrapper for the FORTRAN routine **ndfngroups**.

ndfcnistab

C declaration:
int ndfcnistab(void);

'return value'	f	N/A	Number of targets in the opened $\mathbf{ndfy_i}$ file

This routine is a C wrapper for the FORTRAN routine **ndfnistab**.

ndfcnmaxgps

C declaration:
int ndfcnmaxgps(void);

'return value'	f	N/A	Largest group size used in processing the opened **ndfy$_i$** file

This routine is a C wrapper for the FORTRAN routine **ndfnmaxgps**.

ndfcnppyos

C declaration:

int ndfcnppyos(void);

'return value'	f	N/A	Number of transportable outgoing particles

This routine is a C wrapper for the FORTRAN routine **ndfnppyos**.

ndfcnprod

C declaration:

int ndfcnprod(int C);

C	i	N/A	C-value for the requested reaction
'return value'	f	N/A	Number of outgoing particles produced for reaction C

This routine is a C wrapper for the FORTRAN routine **ndfnprod**. If the requested C-value is not in the reaction list for the current target then 0 is returned.

ndfcnreact

C declaration:

int ndfcnreact(void);

'return value'	f	N/A	Number of reactions for the current target

This routine is a C wrapper for the FORTRAN routine **ndfnreact**.

ndfcnrxs

C declaration:

int ndfcnrxs(int C);

C	i	N/A	C-value for the requested reaction
'return value'	f	N/A	Number of reactions of type C

This routine is a C wrapper for the FORTRAN routine **ndfnrxs**.

ndfcnyos

C declaration:

int ndfcnyos(void);

'return value'	f	N/A	Number of transportable outgoing particles with energy deposit data

This routine is a C wrapper for the FORTRAN routine **ndfnyos**.

ndfcopen

C declaration:

int ndfcopen(int yi, char *Name);

yi	i	N/A	Id of the incident particle
Name[5]	o	N/A	File name (e.g., "**ndf1**")
'return value'	f	N/A	Date in the **ndfy$_i$** file

This routine replaces the routines **ndfinit** and **ndfbuff**. Effectively, it calls **ndfinit**, allocates the requested memory and then calls **ndfbuff**. The date returned by **ndfinit** is returned as **ndfcopen**'s return value and the file name returned by **ndfinit** is returned in the **ndfcopen** argument 'Name'. A call to **ndfcclose** will free the allocated memory. Do not use **ndfclose** with this routine.

ndfcpmat

C declaration:

void ndfcpmat(int yo, int n, double *g, double *pm, double *ed);

yo	i	N/A	Requested outgoing particle ($y_o \neq y_i$)
nch	i	N/A	Number of groups in ch
ch[nch + 1]	i	MeV	Outgoing particle's collapsed group boundaries
matrix[nch $\times$ ncg]	o	barn	$y_i \rightarrow y_o$ production transfer matrix for $y_i \neq y_o$
energy[ncg]	o	MeV-barn	Corrected energy deposited by outgoing particle

This routine is a C wrapper for the FORTRAN routine **ndfpmat**. Note that the argument order is different than that of **ndfpmat**.

ndfcppmatrix

C declaration:

int ndfcppmatrix(int yo, double *pm);

yo	i	N/A	Requested outgoing particle ($y_o \neq y_i$)
matrix['return value' $\times$ ng]	o	barn	Uncollapsed $y_i \rightarrow y_o$ production transfer matrix for $y_i \neq y_o$
'return value'	f	N/A	Number of groups representing the outgoing particle energy grouping

This routine is a C wrapper for the FORTRAN routine **ndfppmatrix**. Note that **ndfppmatrix**'s last argument is returned as **ndfcppmatrix**'s return value.

ndfcppyos

C declaration:

int ndfcppyos(int **yoList);

(*yoList)['return value']	o	N/A	List of outgoing particle ids for the current target
'return value'	f	N/A	Number of outgoing particle ids returned

This routine is a C wrapper for the FORTRAN routine **ndfppyos**. Note that **ndfcppyos**'s last argument is returned as **ndfppyos**'s return value, and that **ndfcppyos** returns a pointer to the list. Currently, the list is static memory and is not and should not be freed. Selecting a new target and calling this routine will overwrite the old data.

ndfcprod

C declaration:

int ndfcprod(int C, int **ZAList, int **MList);

C	i	N/A	C-value for the requested reaction
(*ZAList)['return value']	o	N/A	List of targets (i.e., ZAs)
(*MList)['return value']	o	N/A	List of multiplicities
'return value'	f	N/A	Number of outgoing particles produced for reaction C

This routine is a C wrapper for the FORTRAN routine **ndfprod**. Note that **ndfprod**'s second argument is returned as **ndfcprod**'s return value. If the requested C-value is not in the reaction list for the current target then a fatal message is printed. ZAList and MList are the list of ZA's and their multiplicities. Memory for ZAList and MList are allocated by **ndfcprod** and freed when a new target is selected with **ndfciso** or when the **ndfy$_{\mathbf{i}}$** file is closed with **ndfcclose**. If you use **ndfcprod** you must use **ndfciso** and **ndfcclose** instead of **ndfiso** and **ndfclose** to insure that memory is properly freed.

ndfcreact

C declaration:

int ndfcreact(int **C);

(*C)['return value']	o	N/A	List of C-values for each reaction
'return value'	f	N/A	Number of reactions returned

This routine is a C wrapper for the FORTRAN routine **ndfreact**. Note that **ndfreact**'s second argument is returned as **ndfcreact**'s return value. Memory for C is allocated by **ndfcreact** and freed when a new target is selected with **ndfciso** or when the **ndfy$_{\mathbf{i}}$** file is closed with **ndfcclose**. If you use **ndfcreact** you must use **ndfciso** and **ndfcclose** instead of **ndfiso** and **ndfclose** to insure that memory is properly freed.

ndfcrxs

C declaration:

double ndfcrxs(int C, int n, double *cs);

C	i	N/A	C-value for the requested reaction
n	i	N/A	Number of reactions with this C-value to include
cs[ncg]	o	barn	Collapsed cross-section for this reaction
'return value'	f	MeV	Q-value for last reaction

This routine is a C wrapper for the FORTRAN routine **ndfrxs**. Note that **ndfreact**'s third argument (i.e., the Q-value) is returned as **ndfcreact**'s return value. Also, **ndfrxs**'s second and last arguments are **ndfcrxs**'s third and second arguments, respectively.

ndfcrxslevel

C declaration:

int ndfcrxslevel(int C, int L, double *S, double *Q, double *X1, double *X2, double *X3, double *Qeff, double *cs);

C	i	N/A	C-value for the requested reaction
L	i	N/A	Level for the requested C-value
*S	o	N/A	S-value for reaction
*Q	o	MeV	Q-value for reaction
*X1	o	Varies	X1-value for reaction
*X2	o	Varies	X2-value for reaction
*X3	o	Varies	X3-value for reaction
*Qeff	o	MeV	Q_{eff}-value for reaction
cs[ncg]	o	barns	Collapsed cross-section for reaction
'return value'	f	N/A	1 if requested C and L are present, 0 otherwise

This routine is a C wrapper for the FORTRAN routine **ndfrxslevel**.

ndfcrxslist

C declaration:

int ndfcrxslist(int C, double *S, double *Q, double *X1, double *X2, double *X3, double *Qeff);

C	i	N/A	C-value for the requested reaction
S['return value']	o	N/A	List of S-values for reaction
Q['return value']	o	MeV	List of Q-values for reaction
X1['return value']	o	Varies	List of X1-values for reaction
X2['return value']	o	Varies	List of X2-values for reaction
X3['return value']	o	Varies	List of X3-values for reaction
Qeff['return value']	o	MeV	List of Q_{eff}-values for reaction
'return value'	f	N/A	Number of reactions of type C

This routine is a C wrapper for the FORTRAN routine **ndfrxslist**.

ndfcsig

C declaration:

void ndfcsig(int l, double *tcs, double *ed, double *tm, double *tc);

l	i	N/A	Requested Legendre order
tcs[ncg]	o	barn	Transport corrected total cross-section
ed[ncg]	o	MeV-barn	Energy deposited by incident particle type
tm[ncg × ncg × (l + 1)]	o	barn	Transport corrected interaction transfer matrix
tc[ncg]	o	barn	Transport correcting cross-section

This routine is a C wrapper for the FORTRAN routine **ndfsig**. Note that **ndfcsig**'s first argument is **ndfsig**'s last, with **ndfcsig**'s other arguments being one greater than **ndfsig**'s arguments.

ndfcsp

C declaration:

void ndfcsp(double *speeds);

speeds[ncg]	o	cm/sh	Group speeds for the opened **ndfy$_i$** file

This routine is a C wrapper for the FORTRAN routine **ndfsp**.

ndfctotal

C declaration:

void ndfctotal(double *tcs);

tcs[ncg]	o	barn	Uncorrected collapsed total cross-section for the current target

This routine is a C wrapper for the FORTRAN routine **ndftotal**.

ndfctransfer

C declaration:

void ndfctransfer(int l, double *d);

l	i	N/A	Requested Legendre order
d[ng × ng]	o	barn	Uncollapsed interaction transfer matrix for Legendre order l

This routine is a C wrapper for the FORTRAN routine **ndftransfer**.

ndfctrcorr

C declaration:

void ndfctrcorr(CorrectionTypes c);

c	i	N/A	Requested transport correction method

This routine is a C wrapper for the FORTRAN routine **ndftrcorr**. CorrectionTypes is a C enum with valid values of *e_ndfnone, e_ndfPendlebury, e_ndfLLNL* and *e_ndfFerguson.*

ndfcwsp

C declaration:

void ndfcwsp(double *w);

w[ncg]	o	N/A	Collapsed normalized $l = 0$ flux

This routine is a C wrapper for the FORTRAN routine **ndfwsp**.

ndfcyo

C declaration:

int ndfcyo(int j);

j	i	N/A	Index of the requested id
'return value'	f	N/A	The j_{th} id from the particle directory

This routine is a C wrapper for the FORTRAN routine **ndfyo**.

ndfcyos

C declaration:

int ndfcyos(int **yoList);

(*yoList)['return value']	o	N/A	List of transportable outgoing particles with energy deposit data
'return value'	f	N/A	Number of transportable outgoing particles with energy deposit data

This routine is a C wrapper for the FORTRAN routine **ndfyos**. Note that **ndfyos**'s last argument is returned as **ndfcyos**'s return value, and that **ndfcyos** returns a pointer to the list. Currently, the list is static memory, and is not and should not be freed. Selecting a new target and calling this routine will overwrite the old data.

ndfcyo_gid

C declaration:

int ndfcyo_gid(int yo);

yo	i	N/A	The id of the requested particle
'return value'	f	N/A	Group id used for particle yo

This routine is a C wrapper for the FORTRAN routine **ndfyo_gid**.

ndfcyo_iecflg

C declaration:

int ndfcyo_iecflg(int yo);

yo	i	N/A	Requested transportable outgoing particle
'return value'	f	N/A	Particle and/or Energy conservation flag

This routine is a C wrapper for the FORTRAN routine **ndfyo_info**. Note, **ndfcyo_iecflg** only returns the iecflg value from **ndfyo_info**, which is **ndfcyo_iecflg**'s return value.

ndfcyo_nego

C declaration:

int ndfcyo_nego(int yo);

yo	i	N/A	Requested transportable outgoing particle
'return value'	f	N/A	Number of groups representing transportable outgoing particle's energy grouping

This routine is a C wrapper for the FORTRAN routine **ndfyo_info**. Note, **ndfcyo_nego** only returns the nh value from **ndfyo_info**, which is **ndfcyo_nego**'s return value.

7.7 Files and their routines.

This section is mainly a reference for the developers of the ndf accessing routines.

File	routines
ndf_cvt.F	ndf_cvt_aasection ndf_cvt_cdtod ndf_cvt_pptab ndf_cvt_isotab ndf_cvt_bsection ndf_cvt_getint
ndf_ffromc.F	ndffaccess ndffinfo ndffinit ndf_copyi2c ndf_copyc2i
ndf_filestuff.F	ndf_inquire ndf_uf_open ndf_uf_close ndf_uf_read
ndf_stringstuff.F	ndf_strlen ndf_copystring ndf_catstring
ndfaccess.F	ndfaccess ndfsetsubpath ndfgetsubpath ndfsetgetsubpath ndfsetgrptype ndfgetgrptype ndfsetgetgrptype
ndfatw.F	ndfatw
ndfbuff.F	ndfbuff ndfsetgetbuff ndfsetbuff ndfgetbuff
ndfcast.F	ndfcast
ndfclose.F	ndfclose
ndfcoll0.F	ndfcoll0
ndfcoll1.F	ndfcoll1
ndfcoll2.F	ndfcoll2
ndfcoll3.F	ndfcoll3
ndfcollapse.F	ndfcollapse
ndfcopy.F	ndfcopy ndficopy
ndfcorrec.F	ndfcorrec ndfncorrec
ndfed.F	ndfed
ndfemax.F	ndfemax
ndfep.F	ndfep

ndffatal.F	ndffatal ndfcrashifnotinit ndfcrashifnotbuff ndfcrashifnotiso ndfcrashifnotgrp ndfreaderror
ndffism.F	ndffism
ndffisx.F	ndffisx
ndfflx.F	ndfflx
ndfflxi.F	ndfflxi
ndfflxm.F	ndfflxm
ndfflxp.F	ndfflxp
ndfflxw.F	ndfflxw ndfflxw_l
ndffreeioc.F	ndffreeioc
ndffsp.F	ndffsp ndfifsp
ndfgmap.F	ndfgmap
ndfgp.F	ndfgp
ndfgroup.F	ndfgroup ndfsetgrp ndfgetgrp ndfngroup
ndfidog.F	ndfidog
ndfimat.F	ndfimat
ndfinfo.F	ndfinfo
ndfinit.F	ndfinit ndfgid ndfisfileopen ndfsetpathndf ndfgetpathndf ndfsetgetpathndf
ndfiso.F	ndfiso ndfsetgetiso ndfsetiso ndfgetiso
ndfistab.F	ndfistab ndfnistab
ndfmap.F	ndfmap
ndfmatrx.F	ndfmatrx
ndfmxorder.F	ndfmxorder ndfmxorder_tc
ndfngroups.F	ndfngroups
ndfowfl.F	ndfowfl
ndfpath.F	ndfpath
ndfpmat.F	ndfpmat
ndfppmatrix.F	ndfppmatrix
ndfppyos.F	ndfppyos ndfnppyos ndfyo_gid ndfyo ndfnmaxgps ndfyo_info
ndfprod.F	ndfprod ndfnprod
ndfreact.F	ndfreact ndfnreact
ndfread.F	ndfread
ndfrxs.F	ndfrxs ndfnrxs, ndfrxslevel, ndfrxslist
ndfsig.F	ndfsig
ndfskip.F	ndfskip
ndfsp.F	ndfsp
ndftccalc.F	ndftccalc
ndftcor.F	ndftcor
ndftcs.F	ndftcs
ndftotal.F	ndftotal
ndftran.F	ndftran

ndftransfer.F	ndftransfer
ndftrcorr.F	ndftrcorr
ndfwsp.F	ndfwsp
ndfyos.F	ndfyos ndfnyos

ndf_c2f.c

void ndfcaccess(int yi, char *libnam, char *cVersion, char *grptype, char *subpath, int is);

double ndfcatw(void);

void ndfcbuff(void *p);

void ndfcclose(void);

void ndfccorrec(double *d);

int ndfcncorrec(void)l

void ndfced(int yo, double *d);

void ndfcemax(double *d);

void ndfcep(double *d);

int ndfcfism(double *d);

int ndfcfisx(double *d);

void ndfcflxw(double *d);

void ndfcflxw_l(int l, double *d);

void ndfcfsp(double *d);

int ndfcgid(void);

void ndfcgp(double *d);

void ndfcgroup(int n, double *d, int fid);

int ndfcidog(int gid, double *d);

int ndfcifsp(void);

void ndfcinfo(char *path, int RetDateSize);

int ndfcinit(int yi, char *name, int *m);

int ndfciso(int ZA);

int ndfcistab(int **ZAList);

int ndfcnistab(void);

int ndfcmxorder(void);

int ndfcmxorder_tc(void);

int ndfcngroup(void);

int ndfcngroups(void);

int ndfcnmaxgps(void);

int ndfcopen(int yi, char *name);

void ndfcpmat(int yo, int n, double *g, double *pm, double *ed);

int ndfcppmatrix(int yo, double *pm);

int ndfcppyos(int **YoList);

int ndfcnppyos(void);

int ndfcprod(int C, int **ZA, int **M);

int ndfcnprod(int C);

int ndfcreact(int **C);

int ndfcnreact(void);

double ndfcrxs(int C, int n, double *d);

int ndfcrxslevel(int C, int Level, double *S, double *Q, double *X1, double *X2, double *X3, double *Qeff, double *cs);

int ndfcrxslist(int C, double *S, double *Q, double *X1, double *X2, double *X3, double *Qeff);

int ndfcnrxs(int C);

void ndfcsig(int l, double *tcs, double *ed, double *tm, double *tc);

void ndfcsp(double *d);

void ndfctotal(double *d);

void ndfctransfer(int l, double *d);

void ndfctrcorr(CorrectionTypes t);

void ndfcwsp(double *d);

int ndfcyo(int i);

int ndfcyo_iecflg(int yo);

int ndfcyo_nego(int yo);

int ndfcyos(int **YoList);

int ndfcnyos(void);

int ndfcyo_gid(int yo);

ndf_cfilestuff.c

int ndfuopen_(char *name);

int ndfuopen(char *name);

int ndfuclose_(void);

int ndfuclose(void);

int ndfuread_(void *p, int *Size, int *Offset);

int ndfuread(void *p, int *Size, int *Offset);

void ndfuprintopenedfilename_(void);

void ndfuprintopenedfilename(void);

int ndf_cinquire_(char *File);

int ndf_cinquire(char *File);

ndf_cie.c

int ndf_cie_iscray_(void);

int ndf_cie_iscray(void);

int ndf_cie_getb_(unsigned char *i, int *n);

int ndf_cie_getb(unsigned char *i, int *n);

void ndf_cie_8bto2ints_(void *i8);

void ndf_cie_8bto2ints(void *i8);

double ndf_cie_8btodouble_(unsigned char *dp);

double ndf_cie_8btodouble(unsigned char *dp);

ndfmemory.c

void ndf_malloc(void **ptr, int *nBytes);

void ndf_malloc_(void **ptr, int *nBytes);

void ndf_free(void **ptr);

void ndf_free_(void **ptr);

References

[1] R.J. Howerton, R.E. Dye, P.C. Giles, J.R. Kimlinger, S.T. Perkins and E.F. Plechaty, *Omega Documentation*, UCRL-50400 Vol 25 (1983)

[2] G.W. Hedstrom, *An explanation of ndfgen*, PD-211 (2001)

[3] E.E. Lewis and W.F. Miller, Jr., *Computational Methods of Neutron Transport*, American Nuclear Society, Inc., La Grange Park, Illinois (1993)