LA-UR-14-23160
Approved for public release; distribution is unlimited.

Title: A Patch to MCNP5 for Multiplication Inference: Description and User Guide

Author(s): Solomon, Clell J. Jr.

Intended for: Report

Issued: 2014-05-05

A Patch to MCNP5 for Multiplication Inference: Description and User Guide

Clell J. (CJ) Solomon

11 Aug. 2011

Abstract

A patch to MCNP5 has been written to allow generation of multiple neutrons from a spontaneous-fission event and generate list-mode output. This report documents the implementation and usage of this patch.

1 Introduction

A patch to MCNP5[2] (RSICC version 1.60) has been written to facilitate multiplication inference simulations. The patch is commonly referred to as the "multiplication patch." The patch is meant to assist in analog simulation of multiplication experiments, and the user must take precaution to ensure that default variance reduction is turned off. The patch implements two features:

1. a source that correlates spontaneous-fission neutrons in time
2. a tally modifier that writes a "list-mode" formatted file of capture events in specified cells *

Each feature is implemented independently in the `source.F90` (MCNP's user defined source) and `tallyx.F90` (MCNP's user defined tally modification), respectively.

In the implementation, a requirement that no other MCNP routines could be modified was enforced. Thus, to make the patch work consistently with both OpenMP threading and OpenMPI parallelism, the `tallyx.F90` routine is required to open a separate file for each MPI task and write

*The list mode formatted file is a text file consisting of two columns, the first for the cell in which a capture event occurs and the second for the time at which the capture event occurs.

to the file in "omp critical" sections. The source was tested extensively to ensure that identical list-mode output is produced for serial runs, threading-only runs, mpi-only runs, and mpi-threaded runs.

2 Modified Source Routines

2.1 Source Implementation Description

The correlated fission source is implemented in the `source.F90` file, a listing of which is available in Appendix A. The `source.F90` file is a source file which allows for a user-defined source to be compiled into MCNP. When MCNP executes, if no `SDEF` or `KSRC` cards are found, the user-defined source routine is executed to produce the source particle (lack of a user-defined source routine for this situation results in an "expire").

Inputs to the user-defined source for the multiplication patch are specified by use of MCNP's `RDUM` card. The `RDUM` card allows a list of arbitrary floating-point (real) values to be read in from the input file and stored in memory globally available throughout the problem. The source routine, reads entries from the `RDUM` card and interprets them, following the prescription in the following subsection, to produce the correlated source. Currently, the position sampling is restricted to one of three possibilities: spherical, cylindrical, and Cartesian.

The `source` routine effectively redefines the value of `NPS` to be the number of starting reactions. For example, given a Pu-240 source emitting 2.15 neutrons per spontaneous fission, an `NPS` value of 10 would emit 21.5 neutrons on average. The user of the multiplication patch must think not in terms of the number of emitted particles, but rather in terms of the number of starting spontaneous-fission reactions. Similarly, all tallies, because they are normalized to `NPS`, should be thought of as normalized to the number of starting reactions.

2.2 Basic Usage

The `source.F90` routine obtains information from the `RDUM` card. The general form of the entries on the `RDUM` card is:

```
RDUM  CELL  ZAID  NPS  E-DIST  T-DIST  POS-SAMPLING-METHOD  POSITION-SAMPLING-INFO
```

All entries are required. As an example, consider the `RDUM` input line

```
c     CELL  ZAID  NPS  E-DIST  T-DIST  POS-SAMPLING-METHOD  POSITION-SAMPLING-INFO
RDUM  1000 94240  1e6     3       9             1            0.0   0.0   0.0   4
```

which indicates that 10^6 **fission events** (not particles) will be started in cell 1000 by spontaneous fission of Pu-240. The energies of the fission neutrons will be sampled from distribution 3, and the starting times of the particles will be sampled from distribution 9. Here, a position sampling method of 1 indicates spherical sampling concentric about $(x, y, z) = (0.0, 0.0, 0.0)$ with a radius sampled from distribution 4. The possible values of the position sampling method and required position sampling information are summarized in Table 1.

Table 1: Summary of position sampling methods and required inputs

Sampling Method	Description	Sampling Information
1	Spherical	`POS-X POS-Y POS-Z R-DIST-#`
2	Cylindrical	`POS-X POS-Y POS-Z AXS-U AXS-V AXS-W R-DIST-# EXT-DIST-#`
3	Cartesian	`XXX-DIST-# YYY-DIST-# ZZZ-DIST-#`

Table 1 illustrates the required input for spherical, cylindrical, and Cartesian source sampling. For spherical sampling, four parameters are required: the x, y, and z coordinates of the sphere center and a distribution number corresponding to the radial sampling distribution. For cylindrical sampling, eight parameters are required: the x, y, and z locations of the cylinder base, the u, v, and w components of the vector pointing along the cylinder axis, the distribution number corresponding to the radial sampling distribution, and the distribution number corresponding to the extent (axial) sampling distribution. For Cartesian sampling, only three parameters are required: the distribution numbers corresponding to the sampling distribution of x, y, and z.

2.3 Multiple Starting Reaction Types

Multiple entries are allowed on the `RDUM` card. Consider for example

```
c     CELL  ZAID  NPS  E-DIST  T-DIST  POS-SAMPLING-METHOD  POSITION-SAMPLING-INFO
RDUM  1000 94240  1e6     3       9             1            0.0   0.0   0.0   4
      2000 99999  1e5     2       9             1            0.0   0.0   0.0   5
```

Here, two different reactions are possible: 1. spontaneous fission of Pu-240 (indicated by the 94240 ZAID) and 2. a single neutron emission (indicated by the 99999 faux-ZAID and explained below). The ZAID/reaction-type is the first thing sampled by the source.F90 routine. The reaction-type is sampled in proportion to its `NPS` value given on the `RDUM` card. For the example give, the spontaneous-fission reaction of Pu-240 will be sampled with a probability of 10-in-11 and the single neutron emission with a probability of 1-in-11.

After the reaction is sampled, a position will attempt to be sampled and tested to determine if it is in the specified cell for that reaction type. If not, the sampled position is rejected and another position is sampled. The process will repeat until a position within the correct cell is sampled or

the maximum rejection limit is reached. The default maximum rejection limit is 10^4, but can be modified by the first entry of the IDUM card to whatever the user desires.

Once a starting position is found, the number of particles to start is obtained from the ZAID number provided. The ZAID must be one of the 39 spontaneously fissioning isotopes listed in Table 2 or 99999, which indicates emission of a single particle by, for example, an (α, n) reaction. The number of spontaneous fission neutrons is sampled from the $\bar{\nu}$ value and the ν-width. The ν-width is obtained either from the Lestone or Terrel data depending on the 5th entry of the PHYS:N card, i.e., 0 or 1 = Lestone and 2 = Terrel.

The NPS card must contain the sum of the individual reaction NPS values. In the example above, the NPS card should be the following

```
NPS 1.1e6
```

Failing to have the correct value on the NPS card is a fatal error in sequential or OpenMP builds. However, because of the MPI design of MCNP, the total number of histories is not available to the slaves and therefore this requirement cannot be verified. Thus, for an MPI build of MCNP, a warning message is issued to the user to verify that the NPS card entry equals the sum of the individual reaction NPSs.

2.4 Other Input Options

Two exceptions exist to the input options described above. First, if the ZAID is negative, then the built-in $\overline{\nu}$ values (given in Table 2) are ignored and the entry immediately following the ZAID is used for the $\overline{\nu}$. Second, if the position sampling method is negative, the absolute value of the position sampling method is used for the sampling method, and the entry immediately following the position sampling method is a transformation number corresponding to a TR card that transforms the source location sampled by the position sampling information. For example, the input

```
c     CELL   ZAID    NU  NPS  E-DIST  T-DIST  POS-SAMPLING-METHOD  TR  POS-SAMPLING-INFO
RDUM  3001 -92238  2.38  1e4    1       2            -3            7     3   4   5
```

indicates spontaneous fission of U-238 with a user specified $\overline{\nu}$ of 2.36. 10^4 starting fission reactions will executed. The sampling method is Cartesian (indicated by $3 = |-3|$) where the x, y, and z locations of the initiating fission reaction are sampled from distributions 3, 4, and 5, respectively, and then transformed by transformation 7. After the transformation, the particle is expected to be contained in cell 3001, and, if it is not, the sampling is rejected and resampled. **The user will get a warning that transformation 7 is not used for anything if transformation 7 is only used to define the source via the RDUM card. This warning can be safely ignored.**

Table 2: Available spontaneously fissioning ZAIDS and corresponding $\bar{\nu}$'s

ZAID	$\bar{\nu}$	ZAID	$\bar{\nu}$	ZAID	$\bar{\nu}$	ZAID	$\bar{\nu}$
90230	2.14	92238	2.01	94244	2.30	99253	3.93
91231	1.93	94238	2.22	96244	2.69	98254	3.89
90232	2.14	94239	2.16	96246	3.18	99254	3.95
92232	1.71	94240	2.16	96248	3.11	100254	3.96
92233	1.76	96240	2.39	98248	3.34	99255	3.97
92234	1.81	94241	2.25	97249	3.60	100255	3.73
92235	1.86	95241	2.27	98249	3.41	100256	4.01
92236	1.91	94242	2.15	96250	3.31	100257	3.85
94236	2.13	96242	2.52	98250	3.53	100258	4.03
93237	2.05	95243	2.42	98252	3.76		

2.5 Starting Reactions Versus Starting Particles

WARNING! WARNING! WARNING!

For this version of the multiplication patch, the starting events **ARE NOT** individual neutrons but instead fission events. Therefore, the value entered on the NPS card should be the number of source fission reactions not source neutrons as has been the case historically. For example, if one wishes to calculate the correct NPS number for 1 g of Pu-240 and a count time of 300 s, historically it would have been

$$1 \text{ g} \times 1040 \text{ n/g/s} \times 300 \text{ s} = 3.12 \times 10^5 \text{ n}.$$

However, with this new implementation the correct value is

$$\frac{1 \text{ g} \times 1040 \text{ n/g/s}}{2.16 \text{ n/fission}} \times 300 \text{ s} = 1.44444 \times 10^5 \text{ fissions},$$

where 2.16 is the $\bar{\nu}$ value for Pu-240 from Table 2.

It is important to note that **all tallies will be normalized to this NPS value**, i.e., the number of fissions rather than the number of source particles. Multiplying the tallies by the appropriate $\bar{\nu}$ should correctly reproduce the tallies per source particle.

WARNING! WARNING! WARNING!

3 List-Mode Tally Modifications

3.1 Tally Implementation Description

The list-mode tally output is implemented in the `tallyx.F90` file, a listing of which is available in Appendix B. The `tallyx.F90` file in MCNP's source provides a user defined tally binning. The `tallyx` routine is called after determining all the other binning of a tally contribution and just before the score of the resulting bin is incremented. The `tallyx` routine can use (if provided) user bins given on the `FU` card.

For the multiplication patch, the `tallyx` routine controls writing of the list-mode output and effectively implements a capture tally (the number of captures in a tally cell per source particle) modification of a type-4 tally. The necessary `FU` user inputs for the type-4 tally are outlined in the following subsection. The list-mode output is a two-column text file, where the first column is the cell in which a capture event occurs, and the second column is the time at which the capture event occurs. This output format is useful for later post-processing by external tools (e.g. the "convert.pl" utility[1]) to infer multiplications. When MCNP5 calls `tallyx.F90` it passes in the current tally value. The `tallyx.F90` routine completely ignores the incoming value and sets the tally score value to zero. It then determines if the tally is contributed at a collision, with what nuclide, and if it is captured. If a capture occurs, then the tally score is set to unity. This process essentially replicates an absorption tally.

To provide the list-mode capability while adhering to the mandate not to alter any MCNP base files, two instances of programming acrobatics were required. First, because track-length tallies are performed before a neutron actually knows what event it is undergoing, the collision mechanics have to be performed twice. In the first pass within the `tallyx` routine, the random number state is cached, the collision mechanics are followed to determine if a capture results, and the random number state is restored. Conveniently, the collision mechanics directly follow the track-length tally contributions, so this random-number acrobatics does not alter the flow of the code. Second, to allow MPI execution, each MPI task has to write its own list-mode file. This, in the first pass through `tallyx` an MPI task will determine whether or not it has opened its list-mode file, and if not do so. Then on subsequent calls to `tallyx` the list-mode writes will be appended to that already open file for that MPI task.

3.2 Basic Usage

The `tallyx.F90` routine handles the list-mode output and tally by modification of a type 4 tally. The user specifies the collision nuclide of interest by supplying its ZAID on the `FU` tally modifier card corresponding to the type 4 tally. For example, a capture estimator for He-3 would be specified for the `F14` tally as

```
F14:n  1001
FU14   2003
```

Additionally, the capture resulting from a specific reaction type can be tallied. The user must append the ZAID on the FU card with a '.' and the MT number of the desired capture reaction, generally 102–117. Because this tally is handled with the FU card, an FM card must also be supplied to prevent MCNP5's purging the necessary cross sections. The effects of the FM card are ignored. For example, if one wishes to specify the (n, p) reaction (MT=103) in He-3 the correct input would be

```
F14:n  1001
FU14   2003.103     $ assuming material 1 has He-3 in it the FM card prevents
FM14   1  1  103    $ purging the (n,p) reaction data from the cross sections
```

In addition to providing the list-mode output, the values produced by the FU modification should replicate the results of a standard type 4 tally modified by an FM card. For example, the tally results of the total absorption density tally normally specified by

```
F4:n  1001
FM4   -1  1  -2
```

should be the same as the tally results produced by

```
F14:n  1001
FU4    2003
```

provided material 1 is completely composed of He-3.

3.3 Tally Normalization

By default, tallies are normalized to the value given on the NPS card. Therefore, **all tallies are normalized per starting reaction NOT per starting particle**. If the user desires the normalization per starting particle, then the tally results should be multiplied by the expected number of starting reactions (the value on the NPS card) and divided by the expected number of particles emitted from those reactions.

For isotope i, the expected number of emitted neutrons $\langle S \rangle_i$ is

$$\langle S \rangle_i = m_i e_i t, \tag{1}$$

where m_i is the mass of the ith isotope, e_i is the number of neutrons emitted per unit mass and time, and t is the time of emission. The number of expected spontaneous reactions $\langle R \rangle_i$ producing the $\langle S \rangle_i$ neutrons is given by

$$\langle R \rangle_i = \frac{m_i e_i t}{\overline{\nu}_i} \tag{2}$$

where $\overline{\nu}_i$ is the expected number of neutrons emitted per spontaneous reaction (this value could be unity).

The adjusted normalization value to normalize per particle is then given by

$$\begin{aligned} \frac{\langle R \rangle}{\langle S \rangle} &= \frac{\sum_i \langle R \rangle_i}{\sum_i \langle S \rangle_i} \\ &= \frac{\sum_i \frac{m_i e_i t}{\overline{\nu}_i}}{\sum_i m_i e_i t} \\ &= \frac{t \sum_i \frac{m_i e_i}{\overline{\nu}_i}}{t \sum_i m_i e_i} \\ &= \frac{\sum_i \frac{m_i e_i}{\overline{\nu}_i}}{\sum_i m_i e_i}. \end{aligned} \tag{3}$$

The value above is the expected value of $1/\overline{\nu}$ weighted against the neutron emission rate from each reaction type.

3.4 List-Mode File(s)

More than one list-mode tally can be specified in a problem by having multiple instances of `FU` cards modifying different type 4 tallies. For a sequential or OpenMP build of MCNP, a list-mode output file will be created with the name 'lmout_####', where #### is the tally number preceded by the appropriate number of zeros. For example, list-mode tally 14 would produce the list-mode output file lmout_0014.

A MPI parallel build of MCNP creates a list-mode output file for each MPI slave task for each list-mode tally. For example, if a four task MPI job is started containing list-mode tally 34, then the files lmout_0034_0001, lmout_0034_0002, and lmout_0034_0003 will be created. The first set

of integers corresponds to the tally number, but the second set corresponds to the MPI slave task writing to the file. Each slave task must write its own separate file to prevent simultaneous IO to each of the files.

The user should concatenate all the files containing the same tally number into a single file. This can be quickly accomplished on a Linux or Unix system with the following command:

```
cat lmout_0034_* > lmout_0034
```

The number of lines in the concatenated lmout file should be the same as the number of lines in the lmout if the problem were run sequentially.

By default the lmout files are APPENDED, NOT OVERWRITTEN. The user should take care to appropriately remove or rename list-mode files between runs in the same directory.

References

[1] B. Temple. User's Manual for the convert.pl PERL Script. Technical Report LA–UR–09–05257, Los Alamos National Laboratory, 2009.

[2] X-5 Monte Carlo Team. MCNP–A General Monte Carlo N-Particle Transport Code, Version 5. Technical Report LA-UR-03-1987, Los Alamos National Laboratory, 2003.

A source.F90

```
!+ $Id: source.F90,v 1.3 2013/02/21 23:41:07 csolomon Exp $
! Copyright LANS/LANL/DOE - see file COPYRIGHT_INFO

subroutine source
  ! dummy subroutine.  aborts job if source subroutine is missing.
  ! if nsr=0, subroutine source must be furnished by the user.
  ! at entrance, a random set of uuu,vvv,www has been defined.  the
  ! following variables must be defined within the subroutine:
  ! xxx,yyy,zzz,icl,jsu,erg,wgt,tme and possibly ipt,uuu,vvv,www.
  ! subroutine srcdx may also be needed.
  use mcnp_global
  use mcnp_debug
#ifdef MULT
  use event_log_mod, only: BANK_N_XN_F
  use erprnt_mod, only: erprnt
  implicit none

  integer(i4knd) :: ib
  real(dknd) :: fi
  logical, save :: s_source_setup = .false.
  logical :: s_do_expire = .false.

  integer, parameter :: s_num_sf_zaids = 39

  integer, parameter, dimension(s_num_sf_zaids) :: s_sf_zaids = &
  & (/ 90230, &
  &    91231, & ! ref LA-8869-MS
  &    90232, &
  &    92232, &
  &    92233, &
  &    92234, &
  &    92235, &
  &    92236, &
  &    94236, &
  &    93237, &
  &    92238, &
  &    94238, &
  &    94239, &
  &    94240, &
  &    96240, &
  &    94241, &
  &    95241, &
  &    94242, &
  &    96242, &
  &    95243, &
  &    94244, &
  &    96244, &
  &    96246, &
  &    96248, &
  &    98248, &
  &    97249, &
  &    98249, &
  &    96250, &
  &    98250, &
  &    98252, &
  &    99253, &
  &    98254, &
  &    99254, &
  &   100254, &
  &    99255, &
  &   100255, &
  &   100256, &
  &   100257, &
  &   100258 /)
```

```
real(dknd), parameter, dimension(s_num_sf_zaids) :: s_sf_nubar = &
& (/ 2.14_dknd, &
&    1.93_dknd, & ! ref LA-8869-MS
&    2.14_dknd, &
&    1.71_dknd, &
&    1.76_dknd, &
&    1.81_dknd, &
&    1.86_dknd, &
&    1.91_dknd, &
&    2.13_dknd, &
&    2.05_dknd, &
&    2.01_dknd, &
&    2.22_dknd, &
&    2.16_dknd, &
&    2.16_dknd, &
&    2.39_dknd, &
&    2.25_dknd, &
&    2.27_dknd, &
&    2.15_dknd, &
&    2.52_dknd, &
&    2.42_dknd, &
&    2.30_dknd, &
&    2.69_dknd, &
&    3.18_dknd, &
&    3.11_dknd, &
&    3.34_dknd, &
&    3.60_dknd, &
&    3.41_dknd, &
&    3.31_dknd, &
&    3.53_dknd, &
&    3.76_dknd, &
&    3.93_dknd, &
&    3.89_dknd, &
&    3.95_dknd, &
&    3.96_dknd, &
&    3.97_dknd, &
&    3.73_dknd, &
&    4.01_dknd, &
&    3.85_dknd, &
&    4.03_dknd /)


integer, parameter :: s_pos_info_max_length = 8

integer, parameter :: s_num_pos_methods = 3
integer, parameter, dimension(s_num_pos_methods) :: s_pos_info_lengths = (/4,8,3/)

integer(i4knd), save :: s_nrxn, s_max_reject
integer(i4knd), allocatable, dimension(:), save :: s_cell, s_zaid, s_pos_method, s_edist,
    s_tdist, s_nps, s_trans
real(dknd), allocatable, dimension(:), save :: s_prob, s_user_nubar
real(dknd), allocatable, dimension(:,:), save :: s_pos_info

integer(i4knd) :: s_ib, s_entry, s_itmp, s_itmp2, s_method, s_npar
real(dknd) :: s_fi, s_dtmp, s_tmp_erg, s_tmp_uuu, s_tmp_vvv, s_tmp_www
character(len=120) :: s_stmp
real(dknd), dimension(3) :: s_uvw

real(dknd) :: s_rnd

integer, external :: namchg

!$OMP CRITICAL (SETUP_SOURCE)
if( .not. s_source_setup )then
  call parseRdum()

  if( (.not. s_do_expire) .and. idum(1) < 0 )then
    call expirx(0,'source','maximum rejection number cannot be less than zero')
```

```
  elseif( idum(1) == 0 )then
    s_max_reject = 10000
  else
    s_max_reject = idum(1)
  endif

  if( (.not. s_do_expire) .and. kpt(2) /= 0 )then
    call expirx(0,'source','multiplication patch requires "mode n" exclusively')
  endif

  if( (.not. s_do_expire) .and. mcal /= 0 )then
    call expirx(0,'source','multiplication patch does not work with multigroup')
  endif

  if( (.not. s_do_expire) .and. wcl(1) /= 0. )then
    call expirx(0,'source','multiplication patch requires analog calculation')
  endif

  s_source_setup = .true.
endif
!$OMP END CRITICAL (SETUP_SOURCE)

xxx = 0_dknd
yyy = 0_dknd
zzz = 0_dknd
erg = 14_dknd
jsu = 0_dknd
wgt = 1_dknd
tme = 0_dknd

if( s_do_expire ) return

s_rnd = rang()
do s_itmp=1, s_nrxn
  if( s_rnd < s_prob(s_itmp) )exit
enddo

icl = namchg(1,s_cell(s_itmp))

call samplePos()

if( s_zaid(s_itmp) == 99999 )then
  s_npar = 1
else
  if( s_zaid(s_itmp) < 0 )then
    s_npar = s_acenus(s_user_nubar(s_itmp))
  else
    do s_itmp2=1, s_num_sf_zaids
      if( s_zaid(s_itmp) == s_sf_zaids(s_itmp2) )exit
    enddo
    s_npar = s_acenus(s_sf_nubar(s_itmp2))
  endif
endif

if( s_npar == 0 )then
  nter = 14

  paxtc(1,1,1) = paxtc(1,1,ipt)-one
  paxtc(2,1,1) = paxtc(2,1,ipt)-wgt
  paxtc(3,1,1) = paxtc(3,1,ipt)-wgt*erg
  paxtc(4,nter,1) = paxtc(4,nter,ipt)-one
  paxtc(5,nter,1) = paxtc(5,nter,ipt)-wgt
  paxtc(6,nter,1) = paxtc(6,nter,ipt)-wgt*erg
  pwb(kpwb+ipt,2,icl) = pwb(kpwb+ipt,2,icl)-wgt
  pac(kpac+ipt,1,icl) = pac(kpac+ipt,1,icl)-one
  pac(kpac+ipt,2,icl) = pac(kpac+ipt,2,icl)-one

  smultc(4) = smultc(4)-wgt
```

```
    rlttc(3,2) = rlttc(3,2)-wgt
  else
    call smpsrc(tme, s_tdist(s_itmp), s_ib, s_fi)

    s_tmp_uuu = uuu
    s_tmp_vvv = vvv
    s_tmp_www = www

    fiml(1) = fim(1,icl)

    npa = 1
    do s_itmp2=1, s_npar-1
      call smpsrc(erg, s_edist(s_itmp), s_ib, s_fi)
      vel = slite*sqrt(erg*(erg+two*gpt(1)))/(erg+gpt(1))
      call isos(s_uvw, lev)
      uuu = s_uvw(1)
      vvv = s_uvw(2)
      www = s_uvw(3)

      call bankit(BANK_N_XN_F)

      paxtc(1,1,1) = paxtc(1,1,ipt)+one
      paxtc(2,1,1) = paxtc(2,1,ipt)+wgt
      paxtc(3,1,1) = paxtc(3,1,ipt)+wgt*erg
      pwb(kpwb+ipt,2,icl) = pwb(kpwb+ipt,2,icl)+wgt
      pac(kpac+ipt,1,icl) = pac(kpac+ipt,1,icl)+one
      pac(kpac+ipt,2,icl) = pac(kpac+ipt,2,icl)+one
      smultc(4) = smultc(4)+wgt
      rlttc(3,2) = rlttc(3,2)+wgt
    enddo
    npa = 0

    uuu = s_tmp_uuu
    vvv = s_tmp_vvv
    www = s_tmp_www
    call smpsrc(erg, s_edist(s_itmp), s_ib, s_fi)
  endif

  return
contains

subroutine parseRdum()
  implicit none

  integer(i4knd) :: i
  logical :: s_done
  real(dknd) :: s_t1

  ! SETUP PASS 1
  s_done = .false.
  s_do_expire = .false.
  s_itmp = 1 ! rdum counter
  s_entry = 1
  do while( .not. s_done )
    select case( s_entry )
      case (1) ! look for a cell
        if( rdum(s_itmp) /= 0_dknd )then
          s_nrxn = s_nrxn + 1
        else
          write(s_stmp,'(a,i4)') 'expected cell entry at rdum entry ', s_itmp
        endif
        s_entry = s_entry + 1
        s_itmp = s_itmp + 1

      case (2) ! look for a zaid or 99999=(alpha,n)
        if( rdum(s_itmp) == 0_dknd )then
          s_nrxn = 0
          write(s_stmp,'(a,i4)') 'expected zaid entry at rdum entry ', s_itmp
```

```
      endif
      s_entry = s_entry + 1
      if( rdum(s_itmp) < 0 )then ! user supplied nu-width
        s_itmp = s_itmp + 2
      else
        s_itmp = s_itmp + 1 ! known ZAID
      endif

    case (3) ! look for expected number of events
      if( rdum(s_itmp) == 0_dknd )then
        s_nrxn = 0
        write(s_stmp,'(a,i4)') 'expected number of events at rdum entry ', s_itmp
      endif
      s_entry = s_entry + 1
      s_itmp = s_itmp + 1

    case (4) ! look for an energy distribution
      if( rdum(s_itmp) == 0_dknd )then
        s_nrxn = 0
        write(s_stmp,'(a,i4)') 'expected energy distribution number at rdum entry ', s_itmp
      endif
      s_entry = s_entry + 1
      s_itmp = s_itmp + 1

    case (5) ! look for a time distribution
      if( rdum(s_itmp) == 0_dknd )then
        s_nrxn = 0
        write(s_stmp,'(a,i4)') 'expected time distribution number at rdum entry ', s_itmp
      endif
      s_entry = s_entry + 1
      s_itmp = s_itmp + 1

    case (6) ! look for a position sampling method
             ! 1 = spherical   2 = cylindrical
             ! 3 = box         4 = rejection
      s_method = int(nint(rdum(s_itmp)),i4knd)
      if( abs(s_method) > s_num_pos_methods .or. s_method == 0 )then
        s_nrxn = 0
        write(s_stmp,'(a,i4)') 'expected position sampling method +/- (1-3) at rdum entry ',
            s_itmp
      endif
      s_entry = s_entry + 1
      if( s_method < 0 )then
        s_itmp = s_itmp + 2 ! transformation applied
      else
        s_itmp = s_itmp + 1 ! no transformation
      endif

    case (7) ! look for position sampling info
      s_entry = 1
      s_itmp = s_itmp + s_pos_info_lengths( abs(s_method) )

  end select

  if( s_nrxn == 0 )then
    s_do_expire = .true.
    s_done = .true.
  elseif( rdum(s_itmp) == 0_dknd .and. s_entry == 1 )then
    s_done = .true.
  endif
enddo

if( s_do_expire )then
  call expirx(0,'source', trim(s_stmp))
  return
endif

allocate( s_cell(s_nrxn), &
```

```
        & s_zaid(s_nrxn),&
        & s_pos_method(s_nrxn),&
        & s_edist(s_nrxn),&
        & s_tdist(s_nrxn),&
        & s_nps(s_nrxn),&
        & s_prob(s_nrxn),&
        & s_trans(s_nrxn),&
        & s_user_nubar(s_nrxn),&
        & s_pos_info(s_nrxn,s_pos_info_max_length) )

s_cell = 0
s_zaid = 0
s_pos_method = 0
s_edist = 0
s_tdist = 0
s_trans = 0
s_nps = 0
s_prob = 0_dknd
s_user_nubar = 0_dknd
s_pos_info = 0_dknd

! READ PASS 2
s_nrxn = 0
s_done = .false.
s_do_expire = .false.
s_itmp = 1 ! rdum counter
s_entry = 1
do while( .not. s_done )
  select case( s_entry )
    case (1) ! look for a cell
      s_nrxn = s_nrxn+1
      s_cell(s_nrxn) = int( nint(rdum(s_itmp)), i4knd )

      do icl = 1, mxa
        if( ncl(icl) == s_cell(s_nrxn) ) exit
      enddo

      if( icl > mxa )then
        write(s_stmp,'(a,i4,a)') 'specified cell ', s_cell(s_nrxn), ' not found in cell cards'
        s_nrxn = 0
      endif

      s_entry = s_entry + 1
      s_itmp = s_itmp + 1

    case (2) ! look for a zaid or 99999=(alpha,n)
      s_zaid(s_nrxn) = int( nint(rdum(s_itmp)), i4knd )
      if( s_zaid(s_nrxn) < 0_dknd )then
        s_user_nubar(s_nrxn) = rdum(s_itmp+1)
        if( s_user_nubar(s_nrxn) <= 0_dknd )then
          write(s_stmp,'(a,i4,a)') 'nu width for ZAID ', abs(s_zaid(s_nrxn)), ' must exceed 0'
          s_nrxn = 0
        endif
        s_itmp = s_itmp + 2
      else
        do i=1, s_num_sf_zaids
          if( s_zaid(s_nrxn) == s_sf_zaids(i) ) exit
        enddo
        if( i > s_num_sf_zaids .and. s_zaid(s_nrxn) /= 99999) then
          write(s_stmp,'(a,i6)') 'unknown zaid ', s_zaid(s_nrxn)
          s_nrxn = 0
        endif
        s_itmp = s_itmp + 1
      endif
      s_entry = s_entry + 1

    case (3) ! look for expected number of events
      s_nps(s_nrxn) = int( nint(rdum(s_itmp)), i4knd )
```

```
  if( s_nps(s_nrxn) <= 0 )then
    write(s_stmp,'(a,i10,a)') 'number of events ', s_nps(s_nrxn), ' must exceed 0'
    s_nrxn = 0
  endif
  s_entry = s_entry + 1
  s_itmp = s_itmp + 1

case (4) ! look for an energy distribution
  s_edist(s_nrxn) = int( nint(rdum(s_itmp)), i4knd)

  do i=1, msd
    if( s_edist(s_nrxn) == ksd(1,i) ) exit
  enddo
  if( i > msd )then
    write(s_stmp,'(a,i4,a)') 'distribution number ', s_edist(s_nrxn), ' not found'
    s_nrxn = 0
  else
    s_edist(s_nrxn) = i
  endif
  s_entry = s_entry + 1
  s_itmp = s_itmp + 1

case (5) ! look for a time distribution
  s_tdist(s_nrxn) = int( nint(rdum(s_itmp)), i4knd)

  do i=1, msd
    if( s_tdist(s_nrxn) == ksd(1,i) ) exit
  enddo
  if( i > msd )then
    write(s_stmp,'(a,i4,a)') 'distribution number ', s_tdist(s_nrxn), ' not found'
    s_nrxn = 0
  else
    s_tdist(s_nrxn) = i
  endif
  s_entry = s_entry + 1
  s_itmp = s_itmp + 1

case (6) ! look for a position sampling method
         ! 1 = spherical   2 = cylindrical
         ! 3 = box         4 = rejection
  s_pos_method(s_nrxn) = int( nint(rdum(s_itmp)), i4knd )
  s_entry = s_entry + 1
  s_itmp = s_itmp + 1
  if( s_pos_method(s_nrxn) < 0 )then
    s_pos_method(s_nrxn) = abs(s_pos_method(s_nrxn))
    s_trans(s_nrxn) = int( nint(rdum(s_itmp)), i4knd )
    do s_itmp2=1, mxtr
      if( -trf(1,s_itmp2) == s_trans(s_nrxn) )exit
    enddo
    if( s_itmp2 > mxtr )then
      write(s_stmp,'(a,i4,a)') 'transformation number ', s_trans(s_nrxn), ' not found'
      s_nrxn = 0
    else
      s_trans(s_nrxn) = s_itmp2
    endif
    s_itmp = s_itmp + 1
  endif

case (7) ! look for position sampling info
  s_pos_info(s_nrxn,1:s_pos_info_lengths(s_pos_method(s_nrxn))) = &
   & rdum(s_itmp:s_itmp + s_pos_info_lengths(s_pos_method(s_nrxn))-1)
  s_entry = 1
  s_itmp = s_itmp + s_pos_info_lengths(s_pos_method(s_nrxn))

  select case (s_pos_method(s_nrxn))
    case (1) ! spherical
      do i=1, msd  ! check radial distribution
        if( s_pos_info(s_nrxn,4) == ksd(1,i) ) exit
```

```
  enddo
  if( i > msd )then
    write(s_stmp,'(a,i4,a)') 'spherical source distribution number ', int(nint(
        s_pos_info(s_nrxn,4)),i4knd), ' not found'
    s_nrxn = 0
  else
    s_pos_info(s_nrxn,4) = i
  endif

case (2) ! cylindrical
  do i=1, msd ! check radial distribution
    if( s_pos_info(s_nrxn,7) == ksd(1,i) ) exit
  enddo
  if( i > msd )then
    write(s_stmp,'(a,i4,a)') 'cylindrical source distribution number ', int(nint(
        s_pos_info(s_nrxn,7)),i4knd), ' not found'
    s_nrxn = 0
  else
    s_pos_info(s_nrxn,7) = i
  endif

  do i=1, msd ! check extent distribution
    if( s_pos_info(s_nrxn,8) == ksd(1,i) ) exit
  enddo
  if( i > msd )then
    write(s_stmp,'(a,i4,a)') 'cylindrical source distribution number ', int(nint(
        s_pos_info(s_nrxn,8)),i4knd), ' not found'
    s_nrxn = 0
  else
    s_pos_info(s_nrxn,8) = i
  endif

  s_t1 = 0d0
  do i=4,6
    s_t1 = s_t1 + s_pos_info(s_nrxn,i)**2
  enddo
  s_t1 = dsqrt(s_t1)
  if( s_t1 < 1d-6 )then
    write(s_stmp,'(a,es12.4,a,es12.4,a,es12.4,a)') &
   & 'cylinder axis (', s_pos_info(s_nrxn,4), ',', s_pos_info(s_nrxn,5), ',',
        s_pos_info(s_nrxn,6), ') unnormalizable'
    s_nrxn = 0
  else
    s_pos_info(s_nrxn,4:6) = s_pos_info(s_nrxn,4:6) / s_t1
  endif

case (3) ! cartesian
  do i=1, msd ! check x distribution
    if( s_pos_info(s_nrxn,1) == ksd(1,i) ) exit
  enddo
  if( i > msd )then
    write(s_stmp,'(a,i4,a)') 'cartesian source distribution number ', int(nint(
        s_pos_info(s_nrxn,1)),i4knd), ' not found'
    s_nrxn = 0
  else
    s_pos_info(s_nrxn,1) = i
  endif

  do i=1, msd ! check y distribution
    if( s_pos_info(s_nrxn,2) == ksd(1,i) ) exit
  enddo
  if( i > msd )then
    write(s_stmp,'(a,i4,a)') 'cartesian source distribution number ', int(nint(
        s_pos_info(s_nrxn,2)),i4knd), ' not found'
    s_nrxn = 0
  else
    s_pos_info(s_nrxn,2) = i
  endif
```

```
          do i=1, msd ! check z distribution
            if( s_pos_info(s_nrxn,3) == ksd(1,i) ) exit
          enddo
          if( i > msd )then
            write(s_stmp,'(a,i4,a)') 'cartesian source distribution number ', int(nint(
                s_pos_info(s_nrxn,3)),i4knd), ' not found'
            s_nrxn = 0
          else
            s_pos_info(s_nrxn,3) = i
          endif

        case (4) ! rejection
          do i=1, msd ! check x distribution
            if( s_pos_info(s_nrxn,1) == ksd(1,i) ) exit
          enddo
          if( i > msd )then
            write(s_stmp,'(a,i4,a)') 'rejection source distribution number ', int(nint(
                s_pos_info(s_nrxn,1)),i4knd), ' not found'
            s_nrxn = 0
          else
            s_pos_info(s_nrxn,1) = i
          endif

          do i=1, msd ! check y distribution
            if( s_pos_info(s_nrxn,2) == ksd(1,i) ) exit
          enddo
          if( i > msd )then
            write(s_stmp,'(a,i4,a)') 'rejection source distribution number ', int(nint(
                s_pos_info(s_nrxn,2)),i4knd), ' not found'
            s_nrxn = 0
          else
            s_pos_info(s_nrxn,2) = i
          endif

          do i=1, msd ! check z distribution
            if( s_pos_info(s_nrxn,3) == ksd(1,i) ) exit
          enddo
          if( i > msd )then
            write(s_stmp,'(a,i4,a)') 'rejection source distribution number ', int(nint(
                s_pos_info(s_nrxn,3)),i4knd), ' not found'
            s_nrxn = 0
          else
            s_pos_info(s_nrxn,3) = i
          endif
      end select

    end select

    if( s_nrxn == 0 )then
      s_do_expire = .true.
      s_done = .true.
    elseif( rdum(s_itmp) == 0_dknd .and. s_entry == 1)then
      s_done = .true.
    endif
  enddo

  do i=1, s_nrxn
    s_prob(i) = real( s_nps(i), dknd ) / real( sum(s_nps), dknd )
  enddo

  do i=s_nrxn, 2, -1
    s_prob(i) = sum(s_prob(1:i))
  enddo

  if( mcnp_opt_mpi )then
    if( mynum == 1)then
      call erprnt(1,3,0,0,0,0,0,0,'"ensure nps equals sum of histories for each reaction"')
```

```
      endif
    else
      if( (.not. s_do_expire) .and. sum(s_nps) /= npp )then
        s_do_expire = .true.
        write(s_stmp,'(a)') 'sum of numbers of histories for each reaction must equal nps'
      endif
    endif

!   do i=1, s_nrxn
!     print *, i
!     print *, 's_cell = ', s_cell(i)
!     print *, 's_zaid = ', s_zaid(i)
!     print *, 's_nps = ', s_nps(i)
!     print *, 's_edist = ', s_edist(i)
!     print *, 's_tdist = ', s_tdist(i)
!     print *, 's_pos_method = ', s_pos_method(i)
!     print *, 's_prob = ', s_prob(i)
!     print *, 's_user_nubar = ', s_user_nubar(i)
!     print *, 's_trans = ', s_trans(i)
!     print *, 's_pos_info = ', s_pos_info(i,1:s_pos_info_max_length)
!   enddo

    if( s_do_expire )then
      call expirx(0,'source', trim(s_stmp))
    endif

    return
  end subroutine parseRdum

  subroutine samplePos()
    implicit none

    real(dknd) :: s_rad, s_ext, s_zu, s_zv, s_zw, s_ph, s_xp, s_yp, s_t1, s_t2, &
    & xx, yy, zz
    real(dknd), dimension(3) :: s_dir

    integer :: s_i, s_j
    logical :: s_do_expire, s_done
    integer, parameter :: s_max_reject = 10000

    s_do_expire = .false.
    select case( s_pos_method(s_itmp) )
      case(1) ! sphere

        s_i = 1
        s_j = 1
        do while( s_i /= 0 .and. s_j <= s_max_reject )
          call smpsrc(s_rad, int(nint(s_pos_info(s_itmp,4)), i4knd), s_ib, s_fi)
          call isos(s_dir,0)
          xxx = s_pos_info(s_itmp,1) + s_dir(1)*s_rad
          yyy = s_pos_info(s_itmp,2) + s_dir(2)*s_rad
          zzz = s_pos_info(s_itmp,3) + s_dir(3)*s_rad

          if( s_trans(s_itmp) /= 0 )then
            xx = xxx
            yy = yyy
            zz = zzz

            xxx = trf( 5,s_trans(s_itmp))*xx + trf( 6,s_trans(s_itmp))*yy + &
            &     trf( 7,s_trans(s_itmp))*zz + trf(15,s_trans(s_itmp))
            yyy = trf( 8,s_trans(s_itmp))*xx + trf( 9,s_trans(s_itmp))*yy + &
            &     trf(10,s_trans(s_itmp))*zz + trf(16,s_trans(s_itmp))
            zzz = trf(11,s_trans(s_itmp))*xx + trf(12,s_trans(s_itmp))*yy + &
            &     trf(13,s_trans(s_itmp))*zz + trf(17,s_trans(s_itmp))
          endif

          call chkcel(icl,2,s_i)
          s_j = s_j + 1
```

```
  enddo

case(2) ! cylinder

  s_i = 1
  s_j = 1
  do while( s_i /= 0 .and. s_j <= s_max_reject )
    call smpsrc(s_rad,int(nint(s_pos_info(s_itmp,7)), i4knd), s_ib, s_fi)
    call smpsrc(s_ext,int(nint(s_pos_info(s_itmp,8)), i4knd), s_ib, s_fi)

    s_zu = s_pos_info(s_itmp,4)*s_ext
    s_zv = s_pos_info(s_itmp,5)*s_ext
    s_zw = s_pos_info(s_itmp,6)*s_ext
    s_ph = rang()*2.*pie
    s_xp = s_rad*cos(s_ph)
    s_yp = s_rad*sin(s_ph)
    s_t1 = sqrt(s_pos_info(s_itmp,4)**2+s_pos_info(s_itmp,5)**2)

    xxx = s_pos_info(s_itmp,1)
    yyy = s_pos_info(s_itmp,2)
    zzz = s_pos_info(s_itmp,3)
    if( s_t1/=0. ) then
      s_t2 = 1./s_t1
      xxx = xxx+s_t2*(s_pos_info(s_itmp,4)*s_pos_info(s_itmp,6)*s_xp-s_pos_info(s_itmp,5)*
          s_yp)+s_zu
      yyy = yyy+s_t2*(s_pos_info(s_itmp,5)*s_pos_info(s_itmp,6)*s_xp+s_pos_info(s_itmp,4)*
          s_yp)+s_zv
      zzz = zzz-s_t1*s_xp+s_zw
    else
      xxx = xxx+s_xp
      yyy = yyy+s_yp
      zzz = zzz+s_zw
    endif

    if( s_trans(s_itmp) /= 0 )then
      xx = xxx
      yy = yyy
      zz = zzz

      xxx = trf( 5,s_trans(s_itmp))*xx + trf( 6,s_trans(s_itmp))*yy + &
      &     trf( 7,s_trans(s_itmp))*zz + trf(15,s_trans(s_itmp))
      yyy = trf( 8,s_trans(s_itmp))*xx + trf( 9,s_trans(s_itmp))*yy + &
      &     trf(10,s_trans(s_itmp))*zz + trf(16,s_trans(s_itmp))
      zzz = trf(11,s_trans(s_itmp))*xx + trf(12,s_trans(s_itmp))*yy + &
      &     trf(13,s_trans(s_itmp))*zz + trf(17,s_trans(s_itmp))
    endif

    call chkcel(icl,2,s_i)
    s_j = s_j + 1
  enddo

case(3) ! box
  s_i = 1
  s_j = 1
  do while( s_i /= 0 .and. s_j <= s_max_reject )
    call smpsrc(xxx,int(nint(s_pos_info(s_itmp,1)), i4knd), s_ib, s_fi)
    call smpsrc(yyy,int(nint(s_pos_info(s_itmp,2)), i4knd), s_ib, s_fi)
    call smpsrc(zzz,int(nint(s_pos_info(s_itmp,3)), i4knd), s_ib, s_fi)

    if( s_trans(s_itmp) /= 0 )then
      xx = xxx
      yy = yyy
      zz = zzz

      xxx = trf( 5,s_trans(s_itmp))*xx + trf( 6,s_trans(s_itmp))*yy + &
      &     trf( 7,s_trans(s_itmp))*zz + trf(15,s_trans(s_itmp))
      yyy = trf( 8,s_trans(s_itmp))*xx + trf( 9,s_trans(s_itmp))*yy + &
      &     trf(10,s_trans(s_itmp))*zz + trf(16,s_trans(s_itmp))
```

```
          zzz = trf(11,s_trans(s_itmp))*xx + trf(12,s_trans(s_itmp))*yy + &
          &     trf(13,s_trans(s_itmp))*zz + trf(17,s_trans(s_itmp))
        endif

        call chkcel(icl,2,s_i)
        s_j = s_j + 1
      enddo

  end select

  if( (.not. s_do_expire) .and. s_i /= 0 )then
    write(s_stmp,'(a,i6)') &
    & 'failed to sample a starting position in cell ', s_cell(s_itmp)
      call expirx(0,'source',s_stmp)
  endif

end subroutine samplePos

function s_acenus(tn)
  implicit none

  real(dknd)                :: s_acenus  ! # of neutrons emitted from this fission.
  real(dknd), intent(in)    :: tn        ! value of nubar for fissionable isotope.
  integer,    parameter     :: nb = 30 ! number of values in the nu-bar shift table.
  integer                   :: i,j,id,il,iu,is,mn,na,nn
  real(dknd)                :: a,b,x1,x2,vb,w,w2

  real(dknd), parameter :: tv(nb) = &  !  values of (tn+0.5)/width_nu.
    & (/ 1.9857d0, 2.0022d0, 2.0192d0, 2.0369d0, 2.0552d0, 2.0743d0, 2.0943d0, &
    &    2.1151d0, 2.1369d0, 2.1599d0, 2.1841d0, 2.2097d0, 2.2369d0, 2.2659d0, &
    &    2.2971d0, 2.3307d0, 2.3673d0, 2.4075d0, 2.4521d0, 2.5023d0, 2.5600d0, &
    &    2.6281d0, 2.7114d0, 2.8199d0, 2.9785d0, 3.2980d0, 3.3597d0, 3.4379d0, &
    &    3.5455d0, 3.7227d0 /)
  real(dknd), parameter :: bs(nb) = &  ! nu shift values for Gaussian multiplicity sampling.
    & (/ 0.0255d0, 0.0245d0, 0.0235d0, 0.0225d0, 0.0215d0, 0.0205d0, 0.0195d0, &
    &    0.0185d0, 0.0175d0, 0.0165d0, 0.0155d0, 0.0145d0, 0.0135d0, 0.0125d0, &
    &    0.0115d0, 0.0105d0, 0.0095d0, 0.0085d0, 0.0075d0, 0.0065d0, 0.0055d0, &
    &    0.0045d0, 0.0035d0, 0.0025d0, 0.0015d0, 0.0005d0, 0.0004d0, 0.0003d0, &
    &    0.0002d0, 0.0001d0 /)

  id = s_zaid(s_itmp) !  zaid number for collision isotope.
  iu = 1; is = 1  !  iu = upper do-loop index and is = do-loop stride.
  if( id==92235 ) then
    iu = 2
  elseif( id==94239 ) then
    iu = 3
    is = 2
  endif

  if( width_nu==zero ) then
    if( ifisnu==0 .or. ifisnu==1 ) then  !  use reevaluation fitting first three factorial
        multiplicity moments.
      ! select fissionable isotope id to be sampled for number of fission neutrons.
      select case( id )
      case( 92233 )
        w = 1.070d0
      case( 92235 )
        w = 1.088d0
      case( 92238 )
        w = 1.116d0
      case( 94239 )
        w = 1.140d0
      case( 94241 )
        w = 1.150d0
      case default
        na = id-(id/1000)*1000  !  get the atomic weight of the fissioning isotope id.
        w = 0.007338d0*(na+1.)-0.64d0
      end select
```

```
    elseif( ifisnu==2 ) then  ! sample using the terrell width values.
      ! select fissionable isotope id to be sampled for number of fission neutrons.
      select case( id )
      case( 92235 )
        ! ignore energy-dependent (1.25 and 4.8 mev) widths because of large errors (0.06).
        w = 1.072d0  ! + or - 0.021  (80 kev)
      case( 92238 )
        w = 1.23d0   ! + or - 0.08  (1.5 mev)
      case( 94239 )
        w = 1.14d0   ! + or - 0.07   (80 kev)
      case( 94240 )
        w = 1.109d0  ! + or - 0.012
      case( 92233 )
        w = 1.041d0  ! + or - 0.041  (80 kev)
      case( 94236 )
        w = 1.11d0   ! + or - 0.11
      case( 94238 )
        w = 1.115d0  ! + or - 0.023
      case( 94242 )
        w = 1.069d0  ! + or - 0.035
      case( 96242 )
        w = 1.053d0  ! + or - 0.013
      case( 96244 )
        w = 1.036d0  ! + or - 0.018
      case( 98252 )
        w = 1.207d0  ! + or - 0.012
      case default
        w = 1.079d0  ! + or - 0.007
      end select
    endif
  else
    ! use user input width for Gaussian sampling for all fissionable isotopes (b=0).
    w = width_nu
  endif

  ! use Gaussian sampling for fission neutron multiplicity.
  do
    x1 = two*rang()-one
    x2 = x1**2+rang()**2
    if( x2<=one ) exit
  enddo
  x1 = x1*sqrt(-two*log(x2)/x2)

  ! find value for nu-bar shift b for non-truncated Gaussian sampling.
  b = zero  ! amount subtracted from nu-bar (tn) for more accurate nu-bar.
  vb = (tn+half)/w
  if( vb<=tv(1) ) then
    ! being below the table will cause a slight over prediction of nu-bar.
    b = bs(1)
  elseif( vb<=tv(nb) ) then
    do i = 2,nb-1
      if( vb<=tv(i) ) exit
    enddo
    b = bs(i-1)+(bs(i)-bs(i-1))*(vb-tv(i-1))/(tv(i)-tv(i-1))
  endif

  ! set the number of neutrons selected from this distribution.
  nn = max(0,int(x1*w+tn-b+half))  ! set negative number of neutrons to zero.
  s_acenus = nn

  ! if( nn==0 ) then  ! terminate 0 fission neutrons to loss to fission.
  !   nter = 14
  ! endif

  return
end function s_acenus

#else
```

```
  implicit real(dknd) (a-h,o-z)

  call expirx(0,'source','you need a source subroutine.')
  return
#endif

end subroutine source
```

B tallyx.F90

```
!+ $Id: tallyx.F90,v 1.1.1.1 2011/02/24 20:24:56 csolomon Exp $
! Copyright LANS/LANL/DOE - see file COPYRIGHT_INFO

subroutine tallyx(t,ib)
  ! dummy for user-supplied tallyx subroutine.
  ! t is the input and output tally score value.
  ! ib controls scoring.  see the user's manual.
  use mcnp_global
  use mcnp_debug
#ifdef MULT
  implicit none
  character(len=5), parameter :: t_filebase = "lmout"
  character(len=256), save :: t_filename
  integer, dimension(9), save :: t_tal_ids = 0
  integer, allocatable, dimension(:), save :: t_last_cell
  real(dknd), allocatable, dimension(:), save :: t_last_time
  real(dknd), allocatable, dimension(:), save :: t_last_tal
  logical, save :: t_threads_allocated = .false.

  real(dknd) :: t
  integer(i4knd) :: ib

  integer(i8knd) :: t_seed, t_count
  integer(i8knd) :: t_i, t_zaid, t_itmp, t_id, t_unit

  real(dknd) :: t_dtmp, t_score

  ! backup variables
  integer(i4knd) :: t_jsu, t_ntyn, t_iex, t_iexp, t_mtp

  logical :: t_capture

  real(dknd), external :: getxs

  !$OMP CRITICAL (SETUP_TALLY)
  do t_id=1, size(t_tal_ids)
    if( t_tal_ids(t_id) == 0 ) exit
    if( t_tal_ids(t_id) == jptal(1,ital) )exit
  enddo
  if( t_id > size(t_tal_ids) )then
    call expirx(0,'tallyx','too_many_list_mode_tallies')
  endif

  if( t_tal_ids(t_id) == 0 )then
    t_tal_ids(t_id) = jptal(1,ital)
    t_unit = 990 + t_id
    if( mcnp_opt_mpi )then
      write(t_filename,'(a,a,i4.4,a,i4.4)') t_filebase, '_', jptal(1,ital), '_', mynum
      open(unit=t_unit, file=trim(t_filename), position='append')
    else
      write(t_filename,'(a,a,i4.4)') t_filebase, '_', jptal(1,ital)
      open(unit=t_unit, file=trim(t_filename), position='append')
    endif

    if( .not. t_threads_allocated )then
      allocate( t_last_cell(ntasks), &
              & t_last_time(ntasks), &
              & t_last_tal(ntasks)   )
      t_last_cell = 0
      t_last_time = 0_dknd
      t_threads_allocated = .true.
    endif

    if( jptal(2,ital) /= 4 )then
      call expirx(0,'tallyx','multiplication_patch_only_works_with_f4_tally')
```

```
  endif

  if( iptal(3,1,ital) == -1 )then
    call expirx(0,'tallyx','no collision nuclide found on FU card')
  endif
else
  t_unit = 990 + t_id
endif
!$OMP END CRITICAL (SETUP_TALLY)

t_score = zero
if( pmf == min( pmf, dls ) )then
  call RN_query( seed=t_seed, count=t_count)

  t_jsu = jsu
  t_ntyn = ntyn
  t_iex = iex
  t_iexp = iexp
  t_mtp = mtp
  call t_colidn

  if( t_capture )then
    t_zaid = nxs(2,iex)
    do ibu=1, iptal(3,4,ital)-1
      if( t_zaid == int( tds(iptal(3,1,ital)+ibu), i4knd) )then
        t_dtmp = tds(iptal(3,1,ital)+ibu) - int( tds(iptal(3,1,ital)+ibu), i4knd)
        if( dabs(t_dtmp) < 1d-6 )then
          t_score = t / pmf
          if( icl /= t_last_cell(ktask+1) .or. tme+pmf/vel /= t_last_time(ktask+1) &
          & .or. ital /= t_last_tal(ktask+1) )then
            !$OMP CRITICAL (WRITE_LIST_MODE)
            write(t_unit,'(i8,2x,f20.5)') ncl(icl), tme+pmf/vel
            !$OMP END CRITICAL (WRITE_LIST_MODE)
            t_last_cell(ktask+1) = icl
            t_last_time(ktask+1) = tme+pmf/vel
            t_last_tal(ktask+1) = ital
          endif
          exit
        else
          t_itmp = int(nint( t_dtmp ), i4knd)
          do while( t_itmp == 0 .or. dabs(t_dtmp - nint(t_dtmp)) > 1d-6 )
            t_dtmp = 10d0 * t_dtmp
            t_itmp = int(nint( t_dtmp ), i4knd)
          enddo

          if( rang() <= getxs(t_itmp)/getxs(101) )then
            t_score = t / pmf
            if( icl /= t_last_cell(ktask+1) .or. tme+pmf/vel /= t_last_time(ktask+1) &
            & .or. ital /= t_last_tal(ktask+1) )then
              !$OMP CRITICAL (WRITE_LIST_MODE)
              write(t_unit,'(i8,2x,f20.5)') ncl(icl), tme+pmf/vel
              !$OMP END CRITICAL (WRITE_LIST_MODE)
              t_last_cell(ktask+1) = icl
              t_last_time(ktask+1) = tme+pmf/vel
              t_last_tal(ktask+1) = ital
            endif
            exit
          endif
        endif
      endif
    enddo
  endif

  jsu = t_jsu
  ntyn = t_ntyn
  iex = t_iex
  iexp = t_iexp
  mtp = t_mtp
```

```
      call RN_init_particle( npstc )
      do t_i=1, t_count
        t_dtmp = rang()
      enddo
    endif

    t = t_score
    return

  contains

  subroutine t_colidn
    ! calculate the collision of a neutron with a nucleus.

    use mcnp_global
    use event_log_mod, only : eventp, EVENTP_COLLISION, BANK_N_XN_F
    use mcnp_debug
    use ra1_mod
    use kadjoint_mod, only : do_any_kadjoint, kadjoint_banking

    implicit real(dknd) (a-h,o-z)
    implicit integer(i4knd) (i-n)
    real(dknd) :: vr(3), uvw(3)

    integer :: id_old

    ! save the incoming direction.
    uold(1) = uuu
    uold(2) = vvv
    uold(3) = www
    jsu = 0
    ntyn = 0
!   if( mcal==2 )  go to 290
    sf = wtfasv
    mk = mat(icl)
    if( mk==0 )  call expirx(1,'colidn','collision in void.')
    m = jmd(mk)
    m1 = m

    call ra_ncheck( ncheck, c )
    if( ncheck<0 ) then
!     if( c>totm )  go to 245
      if( npq(mk)==1 )  go to 20
    else
      ! sample the nuclide, using the cumulative total cross section.
      if( npq(mk)==1 )  go to 20
      c = rang()*totm
    endif
    do m = m1,jmd(mk+1)-2
      c = c-rtc(krtc+5,lme(1,m))*fme(m)
      if( c<0. )  go to 20
    enddo
  20 continue
    iex = lme(1,m)
    iexp = m
    mtp = 2
!   mpan = ipan(icl)+m-m1
!   if( mcal==1 )  go to 300
    iet = lmt(m)
    if( erg>esa(iet) )  iet = 0
    if( rtc(krtc+11,iex)>=0. )  iet = 0

    ! ************************  make photons  ****************************
    ! if( kpt(2)/=0 ) then
    !   if( gwt(icl)/=-1.e6 ) then
    !     if( (jxs(12,iex)/=0 .and. npikmt==0) .or. totgp1/=0. ) then
    !       if( tme<tco(2) ) then
```

```
  !          call acegam
  !        endif
  !        if( kdb/=0 )  return
  !      endif
  !    endif
  ! endif

  ! ***************************  capture  ******************************
  ! calculate fission separately in the kcode case.
!  if( nsr==71 ) then
!    call colidk
!  endif
!  el = eg0
!  if( kdb/=0 )  return
!  pan(kpan+1,1,mpan) = pan(kpan+1,1,mpan)+1.
!  pan(kpan+1,2,mpan) = pan(kpan+1,2,mpan)+wgt

  ! sample analog capture if required.
  if( wc1(1)==0. .or. erg<=emcf(1) ) then
    t1 = rtc(krtc+3,iex)+rtc(krtc+8,iex)
    r = rang()*rtc(krtc+5,iex)
    if( t1<r )then
      t_capture = .false.
    else
      t_capture = .true.
    endif
!    i = 3
!    if( r>=rtc(krtc+3,iex) )  i = 4
!    pan(kpan+1,i,mpan) = pan(kpan+1,i,mpan)+wgt
!    nter = 2*i+6
!    if( krflg/=0 )  call eventp(EVENTP_COLLISION)
!    if( iptr/=0  )  call ptrak(4)
!    if( nsr==71  ) then
!      rlttc(2,1) = rlttc(2,1)+wgt*tme
!      rlttc(2,2) = rlttc(2,2)+wgt
!      if( i==4 )  rlttc(4,1) = rlttc(4,1)+wgt*tme
!      sumktc(2)  = sumktc(2)+wgt*rtc(krtc+10,iex)*rtc(krtc+8,iex)/t1
!    endif
  endif
  return
end subroutine t_colidn

#else
  implicit real(dknd) (a-h,o-z)

  !
  ! print a warning the first time this dummy tallyx is called.
  if(jtlx == 0)call errprn(1,jtlx,0,zero,zero,' ',' ',&
    & 'a tallyx subroutine is ordinarily needed with fu cards.')
  return
#endif
end subroutine tallyx
```