

LA-UR-26-26899

Approved for public release; distribution is unlimited.

Title: Source and Tally Plugin Capability Coming in MCNP6.4

Author(s): Jones, Fred Burke

Intended for: Meeting: 2026 MCNP User Symposium, 2026-08-24 - 2026-08-26 (Los Alamos, New Mexico, UNITED STATES)

Issued: 2026-08-27 (rev.1)



Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by Triad National Security, LLC for the National Nuclear Security Administration of U.S. Department of Energy under contract 89233218CNA000001. By approving this article, the publisher recognizes that the U.S. Government retains nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.



Source and Tally Plugin Capability

Coming in MCNP6.4

Fred Jones

on behalf of the MCNP[®] Development Team

MCNP User Symposium

August 24–26, 2026

LA-UR-26-26899, Rev. 1



Managed by Triad National Security, LLC, for the U.S. Department of Energy's NNSA.

8/24/2026

Outline

Introduction

Current source capabilities

Current tally capabilities

Source plugin capability

Tally plugin capability

Demo: source plugin

Demo: tally plugin

Contribute before 6.4

Validation

Conclusions

Additional Slides

Introduction

What is the plugin capability?

- New in MCNP6.4: user-supplied **source** and **tally** routines loaded at **run time** from a shared library.
 - Write in **C, C++, or Fortran**; compile to a shared library (`.so` on Linux, `.dylib` on macOS, `.dll` on Windows); point the MCNP code at it from the input deck. **No rebuild.**
 - A modern alternative to the classic SOURCE/SRCDX and TALLYX subroutine stubs — which required editing and recompiling the MCNP code from source.
- ⇒ Same extension points, delivered as a plug-in library — no source access, no recompilation.

Why: the pain of the current customization path

Today (stub + recompile)

- Get the MCNP source
- Edit `source.F90` / `tallyx.F90`
- Rebuild all of the MCNP code
- Re-validate the custom build

With plugins (6.4)

- Write a small `.c/.f90`
- `gcc -c -fpic my.c` then
`gcc -shared -o my.so my.o`
- Add one input card
- Run the *stock* MCNP executable

Many licensees have *no source access* — so custom TALLYX was simply unavailable. Plugins remove that barrier.

Current source capabilities

The general source: SDEF

- SDEF defines point / volume / surface sources via *source variables* plus *distributions*.
- Default: isotropic point source at the origin, $t = 0$, $E = 14$ MeV, weight 1.
- Companion cards: SI/SP/SB (information / probability / bias), DS (dependent), SC (comment).
- Other source modes: KCODE+KSRC (criticality), SSW/SSR (surface source write/read).

SDEF source variables (selected)

CEL	source cell	AXS	reference axis
SUR	surface (0 = volume)	X,Y,Z	explicit coordinates
ERG	energy (MeV)	WGT	starting weight
TME	time (shakes)	PAR	particle type
DIR	direction cosine μ	CCC	cookie-cutter cell
VEC	reference vector	TR	transformation
POS	position reference	EFF	rejection efficiency
RAD	radial distance	BEM	beam emittance

Every variable can take a scalar value; many can instead be a distribution Dn or depend on another variable. Built-in spectra via SP: Maxwell (-2) , Watt (-3) , Gaussian fusion (-4) , evaporation (-5) , ...

Current tally capabilities

Standard tallies F1–F8

- F1 surface current
 - F2 surface flux
 - F4 cell flux (track length estimate)
 - F5 point / ring detector flux
 - F6 energy deposition / heating (MeV/g)
 - F7 fission energy deposition (neutron-induced)
 - F8 pulse-height (counts / energy distribution)
- Plus mesh tallies (FMESH, TMESH (deprecated)) and radiography (FIP/FIR/FIC).
 - Binning & modifiers: E/T/C, FM, DE/DF, FS/SD, FT.

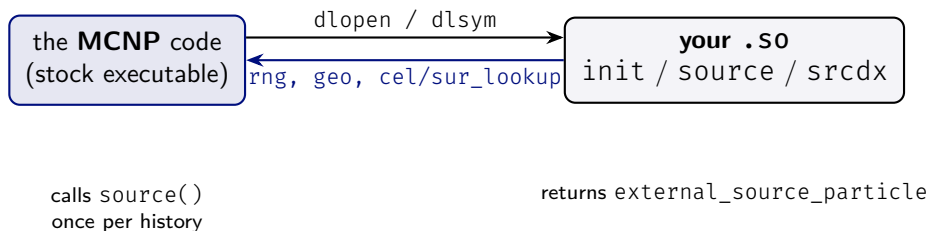
User-defined tallies today: FU + TALLYX

- FUn establishes *user bins* for tally n ; its values are passed to TALLYX and label the output bins.
- TALLYX(t , ib) — user Fortran subroutine called as the last step before a score posts:
 - t = score (may be modified or replaced)
 - ib = loop control: 0 (default) posts one score; <0 posts none; >0 keeps re-entering the user-bin loop until reset to 0 (score into several bins). The target bin is chosen separately.
- **But:** TALLYX is compiled *into* the MCNP code — changing it means editing source and rebuilding the whole code.

⇒ This is exactly what the tally plugin modernizes.

Source plugin capability

Source plugin: how it fits together



The MCNP code loads the library at run time, hands your `init` a set of callbacks, then calls `source()` each history. Your code samples with the MCNP `rng` and returns a particle.

Source plugin: architecture

- **Loader** — `dlopen/dlsym` (Linux/macOS), `LoadLibrary` (Windows).
- **ABI** — `bind(c)` struct `external_source_particle` + function contracts, shared by C and Fortran headers.
- **Functions your library exports** (by symbol name):
 - `init` — *required*, called once; wires up MCNP callbacks
 - `source` — *required*, births one particle per history
 - `srcdx` — *optional*, needed only for anisotropic sources used with detectors or DXTRAN; supplies the source angular probability density (PSC) for the next-event estimate
- **Callbacks the MCNP code hands you:** `rng()` (the MCNP RNG — use it!), `geo()` (geometry lookup), `cel_lookup/sur_lookup`.

Source plugin: init and source (C)

```
int init(external_source_particle (*geofunc)(external_source_particle),
        double (*rngfunc)(void), int64_t (*celfunc)(int64_t),
        int64_t (*surffunc)(int64_t), char *inp) {
    geo = geofunc;  rng = rngfunc;
    cel_lookup = celfunc;  sur_lookup = surfunc;
    if (geo==NULL || rng==NULL || cel_lookup==NULL || sur_lookup==NULL)
        return -1;          // code aborts on nonzero
    return 0;
}

external_source_particle source(void) {
    external_source_particle s = {0};
    s.particle_type = 1;      // neutron
    s.x = 2.0 * rng();        // sample with the MCNP RNG
    s.energy = 14.1;  s.weight = 1.0;
    s = geo(s);              // fills in cell/surface numbers
    return s;
}
```

Tally plugin capability

Tally plugin: what it exposes

- Structs (bind(c)): external_tally_particle (full particle state), external_tally_data (bin/tally context), external_tally_surface.
- **Functions your library exports:**
 - init — *required*, wires up 14 callbacks
 - tallyx(t, ib) — *required*, the score filter
- **Callbacks (read *and* write MCNP state):** rng, get_particle_data, get_tally_data, get_surface_data, get_ddet, xs_lookup, ... plus set_user_bin and set_tally_data.
- Fires only on a tally carrying an FU card.

Tally plugin: tallyx (C)

```
// called for every score to a tally that has an FU card
void tallyx(double *t, int *ib) {
    external_tally_particle particle = get_particle_data();
    external_tally_data tdata = get_tally_data();
    double xs_val = xs_lookup(1, particle.xs_table); // total xs

    *t = *t * xs_val; // modify the score
    set_user_bin(1); // post to user bin 1
}
```

*t is the score (modify or replace); *ib controls the posting loop (0 = one pass, <0 = skip, >0 = repeat). Indices into MCNP arrays are Fortran-based.

Demo: source plugin

Build, link, and use a source plugin

1. Compile to a shared library (Linux/GCC):

```
gcc -c -fpic my_source.c  
gcc -shared -o mcnp_source.so my_source.o
```

2. Point the MCNP code at it from the input deck:

```
extsrc  libname="mcnp_source.so"
```

3. Notes

- Path must be in double quotes, on one line (symlink if it is too long).
- Optional `infile=" ... "` is passed to your `init`.
- `extsrc` is *incompatible* with SDEF, KSRC, SSR — remove other source definitions.

Demo: tally plugin

Build, link, and use a tally plugin

1. Compile to a shared library:

```
gcc -c -fpic my_tally.c  
gcc -shared -o mcnp_tally.so my_tally.o
```

2. Add the card *and* an FU card on the tally:

```
exttal  libname="mcnp_tally.so"  
F4:n 1  
FU4  1          $ activates tallyx on tally 4
```

3. Notes

- Without an FU card, tallyx is never called.
- No OpenMP/MPI in the plugin — the MCNP code owns parallelism; the rng callback is already parallel-safe.

Contribute before 6.4

Help shape the API before release

- MCNP user forum: <https://mcnp.discourse.group/>
- `mcnp_help@lanl.gov` — only if it cannot be shared on the forum.
- When requesting an API addition, include:
 - a detailed description of the requested interface, and
 - ideally, how you would *test* it.
- Example ask (from our own testing): an **energy-explicit cross-section lookup**, $\sigma(\text{mt}, \text{nuclide}, E)$ — today's `xs_lookup` returns the cached collision value, which is stale for uncollided / next-event scores (see Additional Slides).

Ideas for source plugins

- Custom energy/angle spectra not expressible with SDEF distributions.
- Read a phase-space file from another code and birth particles from it.
- Analytic or procedural sources.

Ideas for tally plugins

- Custom filters/weighting beyond FM (e.g. score only on user-defined conditions).
- On-the-fly derived quantities from the full particle state.
- User-bin schemes beyond the fixed FT menu — bin on an arbitrary computed key or combination of particle state, not just the built-in treatments.
- Prototype a proposed tally treatment without touching MCNP source.

Validation

Validation: source plugin vs. SDEF

Test problem

- Isotropic 14.1 MeV point neutron source at the origin.
- Native SDEF vs. a C source plugin — *same geometry, material, tally.*
- Water sphere, F4 neutron flux, 200k histories each.

Every energy bin agrees within **0.6σ** ; total flux differs by **0.07%** — consistent with two independent runs of the same source.

E (MeV)	F4 fluence ($1/\text{cm}^2$)			Δ/σ
	SDEF	plugin		
10^{-3}	2.4971e-4 (0.2%)	2.4956e-4 (0.2%)		-0.21
10^{-1}	1.0520e-5 (0.3%)	1.0536e-5 (0.3%)		+0.33
5	4.5987e-5 (0.3%)	4.5866e-5 (0.3%)		-0.62
14.1	1.3297e-4 (0.2%)	1.3299e-4 (0.2%)		+0.07
total	5.0363e-4 (0.1%)	5.0327e-4 (0.09%)		-0.53

Validation: tally plugin vs. built-in tally

Test problem

- Same water-sphere F4 fluence problem, native SDEF source.
- Plain F4 vs. F4 + FU4 1 routed through a pass-through tallyx plugin.
- Plugin leaves the score unchanged and posts to one user bin.

Because both share the source and RNG stream, the result is **bit-for-bit identical** — every digit, every bin. The only difference is the added “user bin” label.

F4 fluence (1/cm ²)		
<i>E</i> (MeV)	native	plugin
10 ⁻³	2.49708e-4 (0.2%)	2.49708e-4 (0.2%)
10 ⁻¹	1.05203e-5 (0.3%)	1.05203e-5 (0.3%)
5	4.59866e-5 (0.3%)	4.59866e-5 (0.3%)
14.1	1.32971e-4 (0.2%)	1.32971e-4 (0.2%)
total	5.03634e-4 (0.1%)	5.03634e-4 (0.1%)

Conclusions

Conclusions

- MCNP6.4 adds **source** and **tally** plugins: user routines loaded at run time from a shared library.
- Same capability as the old SOURCE/SRCDX/TALLYX stubs — **without recompiling the MCNP code or needing source access.**
- Write in C, C++, or Fortran against a documented, stable ABI.
- **Now is the time** to request the API you need for 6.4.

Questions?

Contact Information

Fred Jones
fbjones@lanl.gov

Introduction

Current source capabilities

Current tally capabilities

Source plugin capability

Tally plugin capability

Demo: source plugin

Demo: tally plugin

Contribute before 6.4

Validation

Conclusions

Additional Slides

Additional Slides

Beyond the basics: two harder reproductions

The core validation used the simplest possible cases. Two more demanding tests exercise more of the API — and one surfaces a real subtlety:

- **Source** — a distributed (volume) source with a sampled Watt fission spectrum, not a fixed-energy point source.
- **Tally** — a tallyx filter that multiplies the flux by a cross section looked up at run time (an FM-style reaction rate).

Additional: Watt-spectrum volume source vs. SDEF

Test problem

- Isotropic source, uniform in a 10 cm sphere, energy from the **Watt fission spectrum** ($a=0.965$, $b=2.29$).
- SDEF RAD=d1 ERG=d2 / SP2 -3 a b vs. a plugin sampling Watt with the MCNP code's algorithm.

Energy-binned F4 fluence agrees within **1.9 σ** across all bins (the one outlier is the >10 MeV tail, $\sim 8\%$ error). No systematic bias.

E (MeV)	F4 fluence (1/cm ²)		
	SDEF	plugin	Δ/σ
10^{-3}	4.6551e-4 (0.09%)	4.6642e-4 (0.05%)	+1.9
10^{-1}	1.7853e-5 (0.2%)	1.7853e-5 (0.2%)	0.0
1	1.8286e-5 (0.4%)	1.8301e-5 (0.4%)	+0.2
5	2.7346e-5 (0.4%)	2.7324e-5 (0.4%)	-0.1
10	4.4230e-6 (1.3%)	4.3892e-6 (1.3%)	-0.4
total	5.9376e-4 (0.07%)	5.9462e-4 (0.04%)	+1.8

Additional: reaction rate via tallyx — a subtlety

Reproduce F4 + FM4 1 1 1 (flux $\times$ total xs of carbon) with a tallyx filter:

```
p = get_particle_data();
*t *= xs_lookup(1, p.xs_table);
```

Collided scores: exact. The error is confined to the *uncollided* contribution.

xs_lookup is only valid at a collision: it returns the value cached at the *last* collision. An uncollided track has `iex=0`, so the lookup reads out of bounds — run-to-run garbage. Native FM re-evaluates σ at the scoring energy. **Lesson:** no energy-explicit lookup is exposed.

F4: energy bins (*blended*)

E (MeV)	native FM	plugin
10^{-3} –10	identical (all bins)	
14.1	correct	garbage (iex=0)

the 14.1 bin mixes uncollided + downscattered >10 MeV.

F5 detector: direct/scattered split (*clean*)

component	native FM	plugin
scattered	8.7490e-5	8.7490e-5
direct (uncoll.)	7.896e-7 (0.3%)	undefined

Additional: anisotropic source with srcdx + detector

Test problem

- Anisotropic source (histogram μ -distribution about VEC) with an F5 point detector.
- Exercises the optional srcdx: the plugin supplies the source **PSC** ($p(\mu)$ toward the detector).

The **direct** F5 line *is* the srcdx/PSC term. It agrees within **0.4 σ** — the hardest part of the source API works.

F5 detector flux (1/cm ²)			
component	SDEF	plugin	Δ/σ
direct (PSC)	7.522e-7 (0.2%)	7.514e-7 (0.2%)	-0.4
scattered	1.6310e-5	1.6352e-5	-
total	1.7062e-5 (0.3%)	1.7103e-5 (0.3%)	+0.6

Additional: collision-count binning (FT INC) via tallyx

Reproduce the built-in FT4 INC treatment (bin the tally by number of collisions) with a `tallyx` filter that routes each score to a user bin:

```
p = get_particle_data();  
b = jbin(3, p.n_collisions_track);  
if (b ≤ 0) { *ib = -1; return; }  
set_user_bin(b);
```

Bit-for-bit identical across all collision-count bins.

Same call the native treatment makes internally (`ibu = jbin(3, ncp)`) — and exercises `set_user_bin` with a *varying* bin.

F4 fluence by # collisions (1/cm ²)		
collisions	native INC	plugin
0	2.79997e-5 (0.2%)	2.79997e-5 (0.2%)
1	2.47506e-5 (0.2%)	2.47506e-5 (0.2%)
2	2.14154e-5 (0.2%)	2.14154e-5 (0.2%)
5	1.35425e-5 (0.3%)	1.35425e-5 (0.3%)
≥ 6	6.16221e-4 (0.3%)	6.16221e-4 (0.3%)
total	7.38132e-4 (0.2%)	7.38132e-4 (0.2%)

A plugin can reproduce an MCNP *special treatment*, not just a plain tally.

Additional: runtime config via infile

The source energy is *not* compiled in — `init` reads it from the file named on the `extsrc` card:

```
extsrc infile="src.cfg"  
       libname="plugin_infile.dylib"
```

```
src.cfg:  energy 5.0
```

Reproduces SDEF ERG=5.0 to **0.3 σ** . Change the energy → edit a text file; **no recompile**.

Syntax gotcha: put `infile=` *before* `libname=` on the card — the parser needs it first.

F4 fluence (1/cm ²)		
<i>E</i> (MeV)	SDEF	plugin
10 ⁻³	3.9179e-4 (0.1%)	3.9149e-4 (0.1%)
1	3.5273e-5 (0.2%)	3.5250e-5 (0.2%)
5	9.8366e-5 (0.2%)	9.8456e-5 (0.2%)
total	5.9281e-4 (0.07%)	5.9263e-4 (0.06%)

Additional: cell-dependent source energy

Test problem

- Uniform volume source in a sphere split by $z = 0$ into two cells; neutrons born in cell 101 are 2 MeV, cell 205 are 14 MeV.
- Plugin uses `geo()` + `cel_lookup()` to set energy from the birth cell.
- Native: SDEF RAD=d1 CEL=d2
ERG=FCEL=d3.

Per-cell F4 agrees within **0.6 σ** ; the 15 MeV bin is strong in cell 205 and negligible in cell 101 — the plugin set energy by cell correctly.

cell	F4 fluence (1/cm ²)			Δ/σ
	SDEF	plugin		
101 (2 MeV)	1.1101e-4 (0.3%)	1.1126e-4 (0.2%)		+0.6
205 (14 MeV)	9.2559e-5 (0.3%)	9.2326e-5 (0.3%)		-0.6

Additional: one score into several bins ($ib>0$)

The tallyx ib flag drives a re-entry loop:
set $ib>0$ to be called again and post the
same score into another user bin.

```
n = get_tally_data().n_tallyx_calls;  
set_user_bin(n); // bin = pass #  
*ib = (n == 1) ? 1 : 0; // 2 passes
```

Each of the two bins independently equals the
plain tally; the user-bin *total* is exactly $2\times$.

F4 fluence ($1/\text{cm}^2$)	
user bin 1	7.38132e-4 (0.2%)
user bin 2	7.38132e-4 (0.2%)
bin total	1.47626e-3 (0.2%)

Additional: reading surface data in tallyx

An F1 surface tally whose tallyx reads the crossed surface's data:

```
p = get_particle_data();  
s = get_surface_data(p.surface_number);  
// s.coefs[0], s.type, s.is_mbody ...
```

For the sphere s0 40 the callback returns type=5, coefs[0]=1600 ($= 40^2$) — exactly the surface definition.

Reading surface data does not perturb the score: the F1 result is **bit-for-bit identical** to the plain tally.

F1 current (per source)		
E (MeV)	native	plugin
14.1	6.89641e-2 (0.8%)	6.89641e-2 (0.8%)
total	7.99725e-1 (0.03%)	7.99725e-1 (0.03%)

Bonus: the ABI is language-blind — a plugin in Rust

The MCNP code dlopens the library and dlsyms init and source — it never asks what language produced them. **Any language with a C ABI works.**

A **Rust** cdylib is a drop-in:

`#[no_mangle]` exports the symbols,
`#[repr(C)]` matches the particle struct.

```
#[no_mangle]
pub extern "C" fn source()
    -> ExternalSourceParticle {
    // isotropic 14.1 MeV neutron,
    // sampled with the MCNP rng()
    geo(s) // fill cell/surface
}
```

F4 fluence (1/cm ²), same source	
C plugin	5.03273e-4 (0.09%)
Fortran plugin	5.03273e-4 (0.09%)
Rust plugin	5.03273e-4 (0.09%)

The isotropic 14.1 MeV point source (validated earlier vs. SDEF), ported to Rust. Same rng call order, same system libm — and the result came out **bit-for-bit identical** across languages, every energy bin and its relative error.

Takeaway: the plugin contract is the C ABI, nothing more. Bring whatever language your workflow needs.