

tallysetup

November 24, 2009
13:39

Contents

1 Define tallies to be used in the run	1
--	---

1 Define tallies to be used in the run

\$Id: tallysetup.web,v 1.14 2006/01/19 20:05:23 dstotler Exp \$

A formatted input file (associated with the symbolic name *tally_infile* in the *degas2.in* file) describes the tallies, including the dependent variables, independent variables, and geometry-related information. This description is used by this code to set the corresponding variables in the *tally* class and writes the latter out to a netCDF file (associated with *tallyfile* in the *degas2.in* file).

The examples directory contains separate input files for each example. Because all of these tallies were previously hardcoded in *tallysetup*, there is not much variation between these files. Furthermore, some will contain unused tallies (warnings will be generated by *tallysetup*). This situation may eventually change.

The input file follows the usual DEGAS 2 conventions:

- Whitespace (lines, spaces, tabs) will be ignored.
 - The example input files use a regular tabbing structure to improve readability.
- Comments are lines beginning with a # sign.
- Although some guidelines must be followed, order of entries is not important.

As described in the documentation associated with *tally.hweb*, there are three types of tallies in DEGAS 2:

tl_type_sector A simple event score carried out when a flight crosses a sector. This can be either a material surface or a purely diagnostic sector.

tl_type_test A tally based solely on the properties of the test flight itself. The most familiar example is the neutral density.

tl_type_reaction Tallies associated with reactions and sources. The most important examples are the sources of background particles, momenta, and energy arising from each reaction or source in the problem.

Correspondingly, the input file for this code is divided up into three sections. The lines in the input file indicating the start of these sections are:

SECTOR TALLIES

TEST TALLIES

REACTION TALLIES

Under each of these headings, the input file will list and describe all of the tallies of each type that are to be used in DEGAS 2.

One of the assumed orderings is that the next line (ignoring whitespace and comments) after one of these headings (or following the last line of the previous tally) is the name of the tally. While these are arbitrary in general, some names have been hardwired into post-processing tools. So, new tallies can use whatever names are convenient. However, the user should exercise care in renaming the existing tallies. Note that there is no way for the code to check that a name has been specified. If this line is omitted, critical default and initialization settings are skipped. The end result will likely be bizarre errors. Tallies based on scalar and vector quantities are handled the same way in *tallysetup*. If the word “vector” appears in the name of the tally, the code will assume that it is of the latter type. It will expect the dependent variable to also contain the string “vector” (e.g., *momentum_vector*).

Following the tally name are up to five subsections which will describe the tally. The order of these subsections below the name is unimportant. Each subsection will consist of the header line and one or more lines containing the actual input information.

DEPENDENT VARIABLE is given by a single line. This is the physical quantity that will be added to the tally when it is updated. The complete list of valid strings is defined in subroutine *init_var0_list*. Some of these make sense only for certain types of tallies. For example, *mass_out* represents the mass of a test particle going out through a sector and is only defined for tallies of the sector type. This subsection cannot be omitted. If a particular dependent variable is not available for the current problem, a warning message is printed and the tally is skipped (*emission_rate_H_alpha* is an example of one such dependent variable).

GEOMETRY OPERATOR specifies an object which defines the geometry and other properties to be associated with this tally. For sector type tallies, this would be the name of a diagnostic group. The most essential information provided to the tally by the diagnostic group is the numbers of the sectors at which this tally will be compiled. There is no default geometry for sector type tallies and a **GEOMETRY OPERATOR** subsection must be provided. For reaction type tallies, this subsection contains a single line with the name of a detector. If this section is omitted, the geometry is assumed to be compiled over the whole volume. This is also the case for test type tallies, for which there is presently no appropriate geometry operator. If the geometry operator for a tally is not defined in the current geometry, that tally will be skipped; a corresponding warning will be printed.

INDEPENDENT VARIABLES lists on one or more lines all of the independent variables for this tally. Each independent variable must be specified on a separate line. If this section is omitted, the tally will be scored as a global scalar. The list of valid independent variables is set in subroutine *set_var_list*. Some of these will make sense only for tallies of a particular type. For example, using **diagnostic** is a reasonable independent variable only for sector type tallies. However, the scoring process in DEGAS 2 has been set up in such a way that unanticipated combinations may be proven useful. The knowledgeable user is encouraged to be creative. The order of the independent variables will be retained throughout the calculation and in the output files. The first independent variable will be the “most rapidly varying”.

Note that the **problem_sp** independent variable stands for “problem species” and represents the union of the background and test species lists, in that order.

CONVERSIONS specifies one or more conversions to be applied prior to output. The most useful examples are scaling by the species mass and scaling by the zone volume. For example, the neutral density requires both of these to arrive at a quantity with dimensions of inverse volume (the raw score would just have units of mass). The list of valid conversions is contained in the array *cv_name* and is set in subroutine *set_conversion_packages* (see also the comments below associated with the macros which set the integers associated with the conversion packages).

ESTIMATORS specify the type of estimator(s) to be used with this tally. Since sector type tallies necessarily use only an event estimator, this subsection is not needed there. For tallies of type test, only a single estimator is required. A single line containing a T specifies a track-length estimator, a C designates a collision (based on all reactions) estimator. The default is track-length estimator.

For tallies of type reaction, different estimators can be used for each of several “reaction groups”. These groups are largely analogous, but less formal, than the reaction type specified by *rc_reaction_type*. The ordered list defining these groups is given by macros below (see *pr_rc_ionize* and subsequent lines. The association with *rc_reaction_type* is made within an if-then-else clause in subroutine *init_tally* (the section begins with the comment “Associate ‘reaction groups’ with *rc_reaction_type* types.”). For example, electron impact excitations and de-excitations can be handled in the same way as electron impact ionizations since in both cases the particle’s velocity vector is left unchanged by the collision processing subroutine. The list and the associated code can be expanded as needed to accommodate new types or special cases. One of those special cases is *pr_rc_test_ion*. This selects all reactions with a charged test reagent (e.g., H_2^+) for separate treatment. In this case, because, charged test particles are not tracked (they effectively have a velocity of 0), a track-length estimator cannot be used. In fact, *tallysetup* will check that a collision estimator has been chosen.

The estimators for reaction type tallies are specified one a single line with spaces in between. In addition to T and C, P is also valid. This last option refers to “post processed tallies” and is available for most cases. The order of the characters corresponds to the order of the reaction types specified in the macro list below. For convenience, a comment and numbered separator line in the example tally input files makes this association more explicit. Other restrictions include:

- reactions of type *ionize_suppress* can not be scored with a collision estimator (because there are no collisions),
- recombination reactions and reactions involving test ions cannot be used with a track-length estimator (there is no tracking),
- a post-processing estimator is valid for spectrum tallies only for recombination (which uses an average ion velocity). For other reactions, the instantaneous kinetic velocity is needed to compute the spectrum properly.

A fourth estimator, N, corresponding to “none”, is provided for debugging purposes and for specific cases. When specified, the contributions made to a tally of type reaction by a particular reaction or source are ignored. This functionality is not needed for the other tally types since the effect would be a null tally.

The default estimator for reaction type tallies is collision.

Each tally is terminated by a line containing “-” in the first column. To clearly delimit the tallies, the example file actually uses full lines,

```
-----
or
----1-----2-----3-----4-----5-----6-----7-----
```

with the numbers being used to count the reaction estimator specifications. Since this line tells the code that the specification of the current tally is complete, it must be included with each tally, including the last one.

Although there are many pitfalls within *tallysetup* for the inexperienced user, a few simple variations that demonstrate the flexibility of the code can be made safely. To avoid changing a tally that some piece of post-processing code needs, do any testing with tallies having new names. For example:

1. Replace a dependent variable of `zone` with `zone_ind_1` or `zone_ind_2` to get a 1-D slice through the geometry in runs based upon UEDGE or old DEGAS input files (which define the *zn_index* array).
2. Delete a dependent variable of `zone` or `diagnostic` to get a globally integrated tally.
3. Remove a mass scaling conversion from one of the sector current tallies to get a mass flux. Additionally removing the species dependence will provide a total mass flux.

One additional feature of *tallysetup* minimizes the effort required to deal with the various isotopes of hydrogen. In the examples, one of the dependent variables is `emission_rate_[H]_alpha`; the corresponding tallies also contain the string `[H]`. The code assumes that the string inside the brackets is the symbolic name of a generic species. It will loop over the equivalent species present in the problem (e.g., H, D, and T) and replace `[H]` with the symbol for each. Recall that this “equivalence” is defined in the species input file.

```
"tallysetup.f" 1 ≡
  @m FILE 'tallysetup.web'
```

These tags will make it easier to pick the estimators for each reaction. Largely, these designations are the same as *rc_reaction_type*, but are defined locally so that they can be arbitrarily assigned for greater flexibility. There’s really nothing analogous needed for non-reaction tallies.

```
"tallysetup.f" 1.1 ≡
  @m pr_rc_ionize 1 // Includes excitation, deexcitation
  @m pr_rc_ionize_suppress 2
  @m pr_rc_chargex 3
  @m pr_rc_elastic 4
  @m pr_rc_dissociation 5
  @m pr_rc_test_ion 6
  @m pr_rc_plate 7
  @m pr_rc_puff 8
  @m pr_rc_recombination 9
  @m pr_rc_unknown 10
  @m pr_rc_max 10
```

To simplify the specification of the conversions to be used to change tallies from their internal values to the external quantities of interest to the user, we’ve defined several “conversion packages” which automate the specification of the actual tally class variables. Here’s a brief description of each along with the required macro values:

```
"tallysetup.f" 1.2 ≡
  @m cv_package_unknown 0 // Use to specify a null conversion
  @m cv_out_scale_test_mass 1 // On output, scale tally by mass of test species
  @m cv_out_scale_problem_sp_mass 2
    // Same, but scale by mass of background or test species (for exchanges)
  @m cv_out_scale_volume 3 // On output, scale tally by the zone volume
  @m cv_out_Pa_to_mTorr 4 // On output, convert Pa to mTorr (for pressure)
  @m cv_out_three_halves 5 // Again, for the pressure calculation
  @m cv_track_v_to_external 6 // Convert velocities from internal to external
    // coordinates; must be done during tracking.
```

The last three are needed largely just to help define the particle number and velocity used during the computation of post-processing tallies. However, the *divide_number* conversion has been found to not work as intended (for multiple source group runs) and should not be used. Namely, the “number” used in the denominator corresponds to only the current source group, and not to the sum over all groups. Providing the desired functionality may be difficult, leading to the development of alternative means of accomplishing the same task (e.g., via explicit code).

```
"tallysetup.f" 1.3 ≡
  @m cv_post_scale_test_mass 7 // During post-processing, scale by the test mass
  @m cv_post_divide_number 8 // During post-processing, divide tally by the
    // number of test particles.
  @m cv_post_v_to_internal 9 // During post-processing, convert velocity
    // from external back to internal coordinates.
  @m cv_max_package 9
```

Second index of the *conversion_packages* array. These correspond in the obvious way to the tally class conversion variables.

```
"tallysetup.f" 1.4 ≡
  @m cv_action 1
  @m cv_type 2
  @m cv_scaler_1 3
  @m cv_scaler_2 4
  @m cv_scaler_3 5
  @m cv_partner_1 6
  @m cv_partner_2 7

  @m cv_max_parameter 7

  @m sec_undefined 0 // Labels for different subsections of tally input file.
  @m sec_name 1
  @m sec_dep 2
  @m sec_indep 3
  @m sec_conv 4
  @m sec_est 5
  @m sec_skip 6
  @m sec_geom 7
```

The unnamed module.

```
"tallysetup.f" 1.5 ≡
  program tallysetup
    implicit_none_f77
    implicit_none_f90

    call readfilenames
    call read_geometry

    call nc_read_elements
    call nc_read_species
    call nc_read_reactions
    call nc_read_materials
    call nc_read_pmi
    call nc_read_problem

    call init_tally
    call nc_write_tally
    call clear_tally

    stop
  end
```

⟨Functions and Subroutines 1.6⟩

Set hardwired tally definitions.

⟨Functions and Subroutines 1.6⟩ ≡

```

subroutine init-tally
  define_varp(pr_rc_type, INT, tally_reac_ind)
  define_varp(var_num, INT, tally_index_ind)

  implicit_none_f77
  de_common // Common
  zn_common
  rc_common
  pr_common
  tl_common
  sp_common
  sc_common
  rf_common
  gi_common
  implicit_none_f90

  integer pointer, i, icv, test, pr_reac, /* Local */
    num_conversions, type, geometry, sub_section, b, e, p, rank, length, generic, num_equiv
  integer indep_var tl_rank_max, tab_index tl_rank_max, conversions tl_cv_max_conversions,
    estimator pr_rc_max, conversion_packages cv_max_package, cv_max_parameter,
    equivalents pr_max_equiv
  character*1 est_name tl_est_unknown:tl_est_max
  character*tl_tag_length cv_name cv_max_package
  character*FILELEN filename, name, name_equiv, dep_var, dep_var_equiv
  character*LINELEN line

  declare_varp(pr_rc_type)
  declare_varp(var_num)

  st_decls
  tl_decls
  ⟨Memory allocation interface 0⟩

  var_alloc(tally_type_num)
  var_alloc(tally_type_base)
  var_alloc(pr_rc_type)
  var_alloc(var_num)

  tl_num = 0
  do i = 1, tl_type_max
    tally_type_numi = 0
    tally_type_basei = int_unused
  end do
  tally_size = 0
  assert(max_bins ≥ max(sc_diag_max_bins, de_max_bins))
  assert(scoring_data_max > pr_var0_num) // Dimension of scoring_data.

  do pr_reac = 1, pr_reaction_num + so_type_num
    pr_rc_typepr_reac = pr_rc_unknown
  end do /* Associate "reaction groups" with rc_reaction_type types. */
  if (pr_reaction_num > 0) then
    do pr_reac = 1, pr_reaction_num
      if (rc_reaction_type(pr_reaction(pr_reac)) ≡ 'ionize') then

```

```

    pr_rc_type_pr_reac = pr_rc_ionize
else if (rc_reaction_type(pr_reaction(pr_reac)) ≡ 'ionize_suppress') then
    pr_rc_type_pr_reac = pr_rc_ionize_suppress
else if (rc_reaction_type(pr_reaction(pr_reac)) ≡ 'chargex') then
    pr_rc_type_pr_reac = pr_rc_chargex
else if (rc_reaction_type(pr_reaction(pr_reac)) ≡ 'elastic') then
    pr_rc_type_pr_reac = pr_rc_elastic
else if (rc_reaction_type(pr_reaction(pr_reac))SP(1:6) ≡ 'dissoc') then
    pr_rc_type_pr_reac = pr_rc_dissociation
else if (rc_reaction_type(pr_reaction(pr_reac)) ≡ 'excitation') then
    pr_rc_type_pr_reac = pr_rc_ionize
else if (rc_reaction_type(pr_reaction(pr_reac)) ≡ 'deexcitation') then
    pr_rc_type_pr_reac = pr_rc_ionize
else if (rc_reaction_type(pr_reaction(pr_reac)) ≡ 'recombination') then
    pr_rc_type_pr_reac = pr_rc_recombination // is a duplicate
else
    pr_rc_type_pr_reac = pr_rc_unknown
end if
end do /* This will overwrite some of the above (e.g., dissociation). */
do test = 1, pr_test_num
    if (sp_z(pr_test(test)) ≠ 0) then // Same check as in particle.hweb
        do i = 1, pr_rc_num(test)
            pr_rc_type_pr_ts_rc(test, i) = pr_rc_test_ion
        end do
    end if
end do
end if
pr_rc_type_pr_reaction_num+so_plate = pr_rc_plate
pr_rc_type_pr_reaction_num+so_puff = pr_rc_puff
pr_rc_type_pr_reaction_num+so_recomb = pr_rc_recombination

nconversions = 0
do icv = 1, tl_cv_max_conversions
    conversions_icv = cv_package_unknown
end do
call set_conversion_packages(conversion_packages, cv_name)
call set_var_list(var_num)

est_name_tl_est_unknown = 'N'
est_name_tl_est_track = 'T'
est_name_tl_est_collision = 'C'
est_name_tl_est_post_process = 'P'

tally_version = '$Id: tallysetup.web,v 1.14 2006/01/19 20:05:23 dstotler Exp $'

filename = filenames_array_tally_infile
assert(filename ≠ char_undef)
open(unit = disk_in, file = filename, form = 'formatted', status = 'old')

type = tl_type_undefined
sub_section = sec_name

loop1: continue

if (¬read_string(disk_in, line, length))
    go to eof

```

```

assert(length ≤ len(line))
length = parse_string(line(: length))
p = 0
if (line ≡ 'SECTOR_TALLIES') then
  type = tl_type_sector
else if (line ≡ 'TEST_TALLIES') then
  type = tl_type_test
else if (line ≡ 'REACTION_TALLIES') then
  type = tl_type_reaction
else if (line ≡ 'DEPENDENT_VARIABLE') then
  if (sub_section ≠ sec_skip)
    sub_section = sec_dep
else if (line ≡ 'GEOMETRY_OPERATOR') then
  if (sub_section ≠ sec_skip)
    sub_section = sec_geom
else if (line ≡ 'INDEPENDENT_VARIABLES') then
  if (sub_section ≠ sec_skip)
    sub_section = sec_indep
else if (line ≡ 'CONVERSIONS') then
  if (sub_section ≠ sec_skip)
    sub_section = sec_conv
else if (line ≡ 'ESTIMATORS') then
  if (sub_section ≠ sec_skip)
    sub_section = sec_est
else if (line(1:1) ≡ '-') then
@#if 0
  /* Having added the sources, there will always be reaction type tallies even if pr_reaction_num
  = 0. */
  if (sub_section ≠ sec_skip ∧ ¬(type ≡ tl_type_reaction ∧ pr_reaction_num ≡ 0)) then
@#else
  if (sub_section ≠ sec_skip) then
@#endif
    < Call Define Tally 1.10 >
  end if
  sub_section = sec_name

else
  if (sub_section ≡ sec_name) then
    name = read_text(line(: length))
    dep_var = char_uninit
    rank = 0
    num_conversions = 0
    pointer = 0
    geometry = tl_geom_volume // Make this the default
    if (type ≡ tl_type_sector) then
      estimator1 = tl_est_collision
    else if (type ≡ tl_type_test) then
      estimator1 = tl_est_track
    else if (type ≡ tl_type_reaction) then
      do i = 1, pr_rc_max
        estimatori = tl_est_collision
      end do
    end if
  end if

```

```

else if (sub_section ≡ sec_dep) then
  dep_var = read_text(line(: length)) // Checked in define_tally
else if (sub_section ≡ sec_geom) then
  if (type ≡ tl_type_sector) then
    geometry = tl_geom_surface
    pointer = diag_lookup(line(: length))
    if (pointer ≤ 0 ∨ pointer > sc_diagnostic_grps) then
      sub_section = sec_skip
      write(stdout, *) 'Tally', trim(name), 'skipped since diagnostic', trim(line),
        'does not appear in this problem.'

      go to loop1
    end if
  else if (type ≡ tl_type_reaction) then
    geometry = tl_geom_detector
    pointer = de_lookup(line(: length))
    if (pointer ≤ 0 ∨ pointer > de_grps) then
      sub_section = sec_skip
      write(stdout, *) 'Tally', trim(name), 'skipped since detector', trim(line),
        'does not appear in this problem.'
      go to loop1
    end if
  else
    assert('Only volume geometry available for test tallies' ≡ '')
  end if
else if (sub_section ≡ sec_indep) then
  ⟨ Indep Vars 1.7 ⟩
else if (sub_section ≡ sec_conv) then
  ⟨ Conversions 1.8 ⟩
else if (sub_section ≡ sec_est) then
  ⟨ Estimators 1.9 ⟩
else if (sub_section ≡ sec_skip) then
  // Do nothing but read the next line...Make this the first option?
end if // sub section

end if // type sections

go to loop1

eof: continue

var_reallocb(tally_name)
var_reallocb(tally_type)
var_reallocb(tally_geometry)
var_reallocb(tally_geometry_ptr)
var_reallocb(tally_rank)
var_reallocb(tally_dep_var_dim)
var_reallocb(tally_indep_var)
var_reallocb(tally_dep_var)
var_reallocb(tally_base)
var_reallocb(tally_tab_index)
var_reallocb(tally_est_test)
var_reallocb(tally_est_reaction)
var_reallocb(tally_num_conversions)
var_reallocb(tally_cv_ptr)

```

```

    var_reallocb(tally_cv_action)
    var_reallocb(tally_cv_type)
    var_reallocb(tally_cv_num_partners)
    var_reallocb(tally_cv_scalers)
    var_reallocb(tally_cv_partners)

    var_free(pr_rc_type)

    return
end

```

See also sections 1.11, 1.12, 1.13, 1.14, and 1.15.

This code is used in section 1.5.

Read in independent variables.

```

⟨ Indep Vars 1.7 ⟩ ≡
    rank++
    assert(next_token(line(:length), b, e, p))

    indep_var_rank = string_lookup(read_text(line(b:e)), tally_var_list_1, tl_index_max)
    if (indep_var_rank ≤ 0 ∨ indep_var_rank > tl_index_max) then
        write(stdout, *) line(b:e)
        assert('Unknown independent variable' ≡ ' ')
    end if

```

This code is used in section 1.6.

Conversions section.

```

⟨ Conversions 1.8 ⟩ ≡
    assert(next_token(line(:length), b, e, p))
    num_conversions++
    conversions_num_conversions = string_lookup(read_text(line(b:e)), cv_name, cv_max_package)
    if (conversions_num_conversions ≤ 0 ∨ conversions_num_conversions > cv_max_package) then
        write(stdout, *) line(b:e)
        assert('Unknown conversion' ≡ ' ')
    end if
    if (next_token(line(:length), b, e, p)) then
        assert(line(b:e) ≡ ':')
        assert(next_token(line(:length), b, e, p)) /* Currently have syntax only for 1 partner */
        conversion_packages_conversions_num_conversions.cv_partner_1 = string_lookup(line(b:), tally_name,
            tl_num)
        assert(conversion_packages_conversions_num_conversions.cv_partner_1 > 0)
    end if

```

This code is used in section 1.6.

Estimators section.

```

⟨Estimators 1.9⟩ ≡
  if (type ≡ tl_type_sector) then
    assert('Should not be a sector ESTIMATOR section' ≡ ' ')
  else if (type ≡ tl_type_test) then
    assert(next_token(line(: length), b, e, p))
    estimator1 = string_lookup(read_text(line(b : e)), est_name1, tl_est_max)
    if (estimator1 ≤ 0 ∨ estimator1 > tl_est_max) then
      write(stdout, *) line(b : e)
      assert('Unknown estimator' ≡ ' ')
    end if
  else if (type ≡ tl_type_reaction) then
    do i = 1, pr_rc_max
      if (next_token(line(: length), b, e, p)) then
        estimatori = string_lookup(read_text(line(b : e)), est_name1, tl_est_max)
        if (estimatori < 0 ∨ estimatori > tl_est_max) then
          write(stdout, *) line(b : e)
          assert('Unknown estimator' ≡ ' ')
        else if (estimatori ≡ 0) then
          assert(line(b : e) ≡ 'N')
        end if
      else
        estimatori = tl_est_collision // default
      end if
    end do /* Enforce restrictions on estimators. More of these must exist (e.g., on post-processing);
             add as needed. */
    if (estimator pr_rc_ionize_suppress ≡ tl_est_collision) then
      assert('Collision estimator not possible for ionize suppress' ≡ ' ')
    else if (estimator pr_rc_recombination ≡ tl_est_track ∨ estimator pr_rc_plate ≡
             tl_est_track ∨ estimator pr_rc_puff ≡ tl_est_track) then
      assert('Track length estimator not possible for sources' ≡ ' ')
    else if (estimator pr_rc_test_ion ≠ tl_est_collision) then
      assert('Test ions must use collision estimator' ≡ ' ')
    else if (int_lookup(tl_index_wavelength_bin, indep_var, rank) > 0) then
      do i = 1, pr_rc_max
        if (estimatori ≡ tl_est_post_process ∧ i ≠ pr_rc_recombination) then
          assert('Spectrum cannot done with post processing estimator' ≡ ' ')
        end if
      end do
    end if // Checks
  end if // tally section

```

This code is used in section 1.6.

Define tally section, including loop over generic species.

```

⟨ Call Define Tally 1.10 ⟩ ≡
  /* First, assign and check tab_index. Need to do this here in case the independent variables section
  of the input file came before the geometry operator specification. */
  do i = 1, rank
    if (indep_vari ≡ tl_index_detector) then
      assert(pointer > 0 ∧ pointer ≤ de_grps)
      assert(type ≡ tl_type_reaction)
      tab_indexi = detector_num_viewspointer
    else if (indep_vari ≡ tl_index_diagnostic) then
      assert(pointer > 0 ∧ pointer ≤ sc_diagnostic_grps)
      assert(type ≡ tl_type_sector)
      tab_indexi = diagnostic_num_sectorspointer
    else if (indep_vari ≡ tl_index_energy_bin ∨ indep_vari ≡ tl_index_angle_bin) then
      assert(pointer > 0 ∧ pointer ≤ sc_diagnostic_grps)
      assert(type ≡ tl_type_sector)
      tab_indexi = diagnostic_tab_indexpointer
    else if (indep_vari ≡ tl_index_wavelength_bin) then
      assert(pointer > 0 ∧ pointer ≤ de_grps)
      assert(type ≡ tl_type_reaction)
      tab_indexi = detector_tab_indexpointer
    else
      tab_indexi = var_numindep_vari
    end if

    if (tab_indexi ≤ 0) then
      write(stdout, *) line(: length)
      assert('Do not have size of this independent variable' ≡ ' ')
    end if
  end do

  if (index(name, '[') ≡ 0) then
    call define_tally(name, type, geometry, pointer, dep_var, rank, indep_var, tab_index,
      num_conversions, conversions, conversion_packages, estimator, pr_rc_type)
  else
    b = index(name, '[') + 1
    e = index(name, ']') - 1
    generic = sp_lookup(name(b:e))
    num_equiv = 0
    do i = 1, pr_test_num
      if (sp_generic(pr_test(i)) ≡ generic) then
        num_equiv ++
        equivalentsnum_equiv = i
      end if
    end do
    assert(num_equiv > 0)
    do i = 1, num_equiv
      if (b ≡ 2) then
        name_equiv = trim(sp_sy(pr_test(equivalentsi))) || name(e + 2 :)
      else
        name_equiv = name(: b - 2) || trim(sp_sy(pr_test(equivalentsi))) || name(e + 2 :)
      end if
      /* Only name and dep_var are still strings at this point. We may eventually have
      isotope-specific detectors and will have to generalize the setting of pointer. */
    end do
  end if

```

```

if (index(dep_var, '[') > 0) then
  b = index(dep_var, '[') + 1
  e = index(dep_var, ']') - 1
  if (sp_lookup(dep_var(b:e)) ≠ generic) then
    assert('Dep.var.generic.species.must.match.one.in.name' ≡ '[')
  end if
  if (b ≡ 2) then
    dep_var_equiv = trim(sp_sy(pr_test(equivalentsi))) || dep_var(e + 2 :)
  else
    dep_var_equiv = dep_var(: b - 2) || trim(sp_sy(pr_test(equivalentsi))) || dep_var(e + 2 :)
  end if
else
  dep_var_equiv = dep_var
end if
call define_tally(name_equiv, type, geometry, pointer, dep_var_equiv, rank, indep_var,
  tab_index, num_conversions, conversions, conversion_packages, estimator, pr_rc_type)
end do
end if

```

This code is used in section 1.6.

Carry out individual variable assignments for tally definition.

⟨Functions and Subroutines 1.6⟩ +=

```
subroutine define_tally(name, type, geometry, pointer, dep_var, rank, indep_var, tab_index,
    num_conversions, conversions, conversion_packages, estimator, arg_type)
```

```
    implicit_none_f77
```

```
    pr_common
```

```
    tl_common    // Common
```

```
    implicit_none_f90
```

```
    integer type, geometry, pointer, rank, num_conversions    // Input
```

```
    integer indep_var tl_rank_max, tab_index tl_rank_max, conversions tl_cv_max_conversions,
        conversion_packages cv_max_package, cv_max_parameter, estimator pr_rc_max, arg_type*
```

```
    character*(*) name, dep_var
```

```
    integer i, i_var0, est, pr_reac    // Local
```

```
    tl_decls
```

```
    st_decls
```

⟨Memory allocation interface 0⟩

```
    i_var0 = string_lookup(dep_var, pr_var0_list, pr_var0_num)
```

```
    if (i_var0 ≤ 1) then    // "1" is 'unknown'
```

```
        write (stdout, *) '␣Tally␣', trim(name), '␣skipped␣since␣dep.␣var.␣', trim(dep_var),
            '␣does␣not␣appear␣in␣this␣problem.'
```

```
        return
```

```
    end if
```

```
    tl_num++
```

```
    var_realloca(tally_name)
```

```
    var_realloca(tally_type)
```

```
    var_realloca(tally_geometry)
```

```
    var_realloca(tally_geometry_ptr)
```

```
    var_realloca(tally_rank)
```

```
    var_realloca(tally_dep_var_dim)
```

```
    var_realloca(tally_indep_var)
```

```
    var_realloca(tally_dep_var)
```

```
    var_realloca(tally_base)
```

```
    var_realloca(tally_tab_index)
```

```
    var_realloca(tally_est_test)
```

```
    var_realloca(tally_est_reaction)
```

```
    var_realloca(tally_num_conversions)
```

```
    var_realloca(tally_cv_ptr)
```

```
    if (tally_type_numtype ≡ 0) then
```

```
        tally_type_basetype = tl_num
```

```
    else    /* Check that tallies of each type are all grouped together */
```

```
        assert(tally_type tl_num-1 ≡ type)
```

```
    end if
```

```
    tally_name tl_num = name
```

```
    if (index(name, 'vector') > 0) then
```

```
        assert(index(dep_var, 'vector') > 0)
```

```
        tally_dep_var_dim tl_num = tl_dep_var_vector
```

```
    else
```

```

    tally_dep_var_dimtl_num = tl_dep_var_scalar
end if
tally_typetl_num = type
tally_type_numtype ++
tally_geometrytl_num = geometry
tally_geometry_ptrtl_num = pointer

tally_dep_vartl_num = i_var0

tally_ranktl_num = rank
do i = 1, rank
    tally_indep_vartl_num,i = indep_vari
    tally_tab_indextl_num,i = tab_indexi
end do
if (rank < tl_rank_max) then
    do i = rank + 1, tl_rank_max
        tally_indep_vartl_num,i = tl_index_unknown
        tally_tab_indextl_num,i = 1
    end do
end if

tl_set_base_size(tally, tally_tab_index, tl_num)

tally_num_conversionstl_num = num_conversions
do i = 1, tl_cv_max_conversions
    tally_cv_ptrtl_num,i = tl_cv_unknown
end do

if (num_conversions > 0) then
    assert(num_conversions ≤ tl_cv_max_conversions)
    do i = 1, num_conversions
        nconversions ++
        var_realloca(tally_cv_action)
        var_realloca(tally_cv_type)
        var_realloca(tally_cv_num_partners)
        var_realloca(tally_cv_scalers)
        var_realloca(tally_cv_partners)

        tally_cv_ptrtl_num,i = nconversions
        tally_cv_actionnconversions = conversion_packagesconversionsi,cv_action
        tally_cv_typenconversions = conversion_packagesconversionsi,cv_type
        assert(tl_cv_max_scalers ≡ 3)
        tally_cv_scalersnconversions,1 = conversion_packagesconversionsi,cv_scaler_1
        tally_cv_scalersnconversions,2 = conversion_packagesconversionsi,cv_scaler_2
        tally_cv_scalersnconversions,3 = conversion_packagesconversionsi,cv_scaler_3
        assert(tl_cv_max_partners ≡ 3)
        tally_cv_partnersnconversions,1 = tl_num
        tally_cv_partnersnconversions,2 = conversion_packagesconversionsi,cv_partner_1
        tally_cv_partnersnconversions,3 = conversion_packagesconversionsi,cv_partner_2
        tally_cv_num_partnersnconversions = 1
        if (conversion_packagesconversionsi,cv_partner_1 ≠ tl_cv_partner_unknown) then
            tally_cv_num_partnersnconversions = 2
            if (conversion_packagesconversionsi,cv_partner_2 ≠ tl_cv_partner_unknown) then
                tally_cv_num_partnersnconversions = 3
            end if
        end if
    end do
end if

```

```

        end if
    end if

    end do
end if

if (tally_typetl_num ≡ tl_type_reaction) then
    do est = 1, tl_est_max
        do pr_reac = 1, pr_reaction_num + so_type_num
            tally_est_reactiontl_num,est,pr_reac = zero // Default
        end do
        tally_est_testtl_num,est = real_unused
    end do
    do pr_reac = 1, pr_reaction_num + so_type_num
        if (estimatorarg_type pr_reac ≠ tl_est_unknown) then
            tally_est_reactiontl_num,estimator arg_type pr_reac,pr_reac = one
        end if
    end do

else if (tally_typetl_num ≡ tl_type_test) then
    do est = 1, tl_est_max
        tally_est_testtl_num,est = zero // Default
        do pr_reac = 1, pr_reaction_num + so_type_num
            tally_est_reactiontl_num,est,pr_reac = real_unused
        end do
    end do /* There's only one "type" of test particle in this context: */
    tally_est_testtl_num,estimator_1 = one
else if (tally_typetl_num ≡ tl_type_sector) then
    do est = 1, tl_est_max
        tally_est_testtl_num,est = real_unused
        do pr_reac = 1, pr_reaction_num + so_type_num
            tally_est_reactiontl_num,est,pr_reac = real_unused
        end do
    end do
end if

return
end

```

Write out data into netcdf file `tally.nc` Note: might eventually merge these data into the problemfile, written by `nc_write_problem`.

⟨Functions and Subroutines 1.6⟩ +≡

```

subroutine nc_write_tally
  implicit none_f77
  rf_common
  pr_common
  tl_common // Common
  implicit none_f90
  tl_decls
  nc_decls
  st_decls
  integer fileid // Local

  tl_ncdecl
  character*LINELEN description, program_version
  character*FILELEN tempfile

  program_version = '$Id: tallysetup.web,v1.14_2006/01/19_20:05:23_dstotler_Exp$'

  tempfile = filenames_array_tallyfile
  assert(tempfile ≠ char_undef)
  fileid = nccreate(tempfile, NC_CLOBBER, nc_stat)

  description = 'Tally_description_for_problem_in_degas2'

  call ncattputc(fileid, NC_GLOBAL, 'description', NC_CHAR, string_length(description),
    description, nc_stat)

  call ncattputc(fileid, NC_GLOBAL, 'program_version', NC_CHAR,
    string_length(program_version), program_version, nc_stat)

  call ncattputc(fileid, NC_GLOBAL, 'tally_version', NC_CHAR, string_length(tally_version),
    tally_version, nc_stat)

  tl_ncdef(fileid)
  call ncendef(fileid, nc_stat)
  tl_ncwrite(fileid)

  call ncclose(fileid, nc_stat)

  return
end

```

Clear tally arrays.

⟨Functions and Subroutines 1.6⟩ +≡

```
subroutine clear_tally
  implicit_none_f77
  pr_common
  tl_common
  implicit_none_f90

  tl_decls

  ⟨Memory allocation interface 0⟩

  var_free(tally_type_num)
  var_free(tally_type_base)
  var_free(tally_name)
  var_free(tally_type)
  var_free(tally_geometry)
  var_free(tally_geometry_ptr)
  var_free(tally_rank)
  var_free(tally_indep_var)
  var_free(tally_dep_var)
  var_free(tally_base)
  var_free(tally_tab_index)
  var_free(tally_est_test)
  var_free(tally_est_reaction)
  var_free(tally_num_conversions)
  var_free(tally_cv_ptr)
  var_free(tally_cv_action)
  var_free(tally_cv_type)
  var_free(tally_cv_num_partners)
  var_free(tally_cv_scalers)
  var_free(tally_cv_partners)

  return
end
```

Define the contents of the conversion packages. Define these verbally in the main code and stick the grunge here. These are divided up into the separate *tl_cv* arrays in *define_tally*. Note: these definitions could possibly be used to *replace* the separate arrays in the tally class.

(Functions and Subroutines 1.6) +≡

```

subroutine set_conversion_packages(conversion_packages, cv_name)
  implicit_none_f77
  implicit_none_f90

  integer conversion_packages cv_max_package, cv_max_parameter // Output
  character*tl_tag_length cv_name*

  integer p // Local

  assert(tl_cv_max_scalers ≡ 3)
  assert(tl_cv_max_partners ≡ 3)

  do p = 1, cv_max_package
    conversion_packages p, cv_partner_1 = tl_cv_partner_unknown
    conversion_packages p, cv_partner_2 = tl_cv_partner_unknown
  end do

  p = cv_out_scale_test_mass
  cv_name_p = 'scale_test_mass'

  conversion_packages p, cv_action = tl_cv_scale
  conversion_packages p, cv_type = tl_cv_output
  conversion_packages p, cv_scaler_1 = tl_cv_test_mass
  conversion_packages p, cv_scaler_2 = tl_cv_scaler_unknown
  conversion_packages p, cv_scaler_3 = tl_cv_scaler_unknown

  p = cv_out_scale_problem_sp_mass
  cv_name_p = 'scale_problem_sp_mass'

  conversion_packages p, cv_action = tl_cv_scale
  conversion_packages p, cv_type = tl_cv_output
  conversion_packages p, cv_scaler_1 = tl_cv_problem_sp_mass
  conversion_packages p, cv_scaler_2 = tl_cv_scaler_unknown
  conversion_packages p, cv_scaler_3 = tl_cv_scaler_unknown

  p = cv_out_scale_volume
  cv_name_p = 'scale_volume'

  conversion_packages p, cv_action = tl_cv_scale
  conversion_packages p, cv_type = tl_cv_output
  conversion_packages p, cv_scaler_1 = tl_cv_volume
  conversion_packages p, cv_scaler_2 = tl_cv_scaler_unknown
  conversion_packages p, cv_scaler_3 = tl_cv_scaler_unknown

  p = cv_out_Pa_to_mTorr
  cv_name_p = 'Pa_to_mTorr'

  conversion_packages p, cv_action = tl_cv_scale
  conversion_packages p, cv_type = tl_cv_output
  conversion_packages p, cv_scaler_1 = tl_cv_Pa_per_mTorr
  conversion_packages p, cv_scaler_2 = tl_cv_scaler_unknown
  conversion_packages p, cv_scaler_3 = tl_cv_scaler_unknown

  p = cv_out_three_halves
  cv_name_p = 'three_halves'

```

```

conversion_packages_p,cv_action = tl_cv_scale
conversion_packages_p,cv_type = tl_cv_output
conversion_packages_p,cv_scaler_1 = tl_cv_three_halves
conversion_packages_p,cv_scaler_2 = tl_cv_scaler_unknown
conversion_packages_p,cv_scaler_3 = tl_cv_scaler_unknown

p = cv_post_scale_test_mass
cv_name_p = 'post_scale_test_mass'

conversion_packages_p,cv_action = tl_cv_scale
conversion_packages_p,cv_type = tl_cv_post
conversion_packages_p,cv_scaler_1 = tl_cv_test_mass
conversion_packages_p,cv_scaler_2 = tl_cv_scaler_unknown
conversion_packages_p,cv_scaler_3 = tl_cv_scaler_unknown

p = cv_track_v_to_external
cv_name_p = 'track_v_to_external'

conversion_packages_p,cv_action = tl_cv_to_external_coords
conversion_packages_p,cv_type = tl_cv_track
conversion_packages_p,cv_scaler_1 = tl_cv_pos_1
conversion_packages_p,cv_scaler_2 = tl_cv_pos_2
conversion_packages_p,cv_scaler_3 = tl_cv_pos_3

p = cv_post_divide_number
cv_name_p = 'divide_number'

conversion_packages_p,cv_action = tl_cv_divide_number
conversion_packages_p,cv_type = tl_cv_post
conversion_packages_p,cv_scaler_1 = tl_cv_scaler_unknown
conversion_packages_p,cv_scaler_2 = tl_cv_scaler_unknown
conversion_packages_p,cv_scaler_3 = tl_cv_scaler_unknown

p = cv_post_v_to_internal
cv_name_p = 'post_v_to_internal'

conversion_packages_p,cv_action = tl_cv_to_internal_coords
conversion_packages_p,cv_type = tl_cv_post
conversion_packages_p,cv_scaler_1 = tl_cv_zone_pos_1
conversion_packages_p,cv_scaler_2 = tl_cv_zone_pos_2
conversion_packages_p,cv_scaler_3 = tl_cv_zone_pos_3

return
end

```

Set string equivalents to independent variable macros.

(Functions and Subroutines 1.6) +=

```

subroutine set_var_list(var_num)
  implicit none f77
  pr_common    // Common
  so_common
  tl_common
  zn_common
  implicit none f90

  integer var_num0:* // Output
  integer i // Local

  tally_var_list_tl_index_unknown = 'unknown'
  tally_var_list_tl_index_zone = 'zone'
  tally_var_list_tl_index_plasma_zone = 'plasma_zone'
  tally_var_list_tl_index_test = 'test'
  tally_var_list_tl_index_problem_sp = 'problem_sp'
  tally_var_list_tl_index_detector = 'detector'
  tally_var_list_tl_index_test_author = 'test_author'
  tally_var_list_tl_index_reaction = 'reaction'
  tally_var_list_tl_index_pmi = 'pmi'
  tally_var_list_tl_index_material = 'material'
  tally_var_list_tl_index_source_group = 'source_group'
  tally_var_list_tl_index_sector = 'sector'
  tally_var_list_tl_index_strata = 'strata'
  tally_var_list_tl_index_strata_segment = 'strata_segment'
  tally_var_list_tl_index_energy_bin = 'energy_bin'
  tally_var_list_tl_index_angle_bin = 'angle_bin'
  tally_var_list_tl_index_wavelength_bin = 'wavelength_bin'
  tally_var_list_tl_index_diagnostic = 'diagnostic'
  tally_var_list_tl_index_zone_ind_1 = 'zone_ind_1'
  tally_var_list_tl_index_zone_ind_2 = 'zone_ind_2'

  /* Do loop starts with last of the above. */
  do i = tl_index_zone_ind_2 + 1, tl_index_max
    tally_var_list_i = char_unused
  end do

  do i = 1, tl_index_max
    var_num_i = 0
  end do

  var_num_tl_index_zone = zn_num
  var_num_tl_index_test = pr_test_num
  var_num_tl_index_problem_sp = pr_background_num + pr_test_num
  var_num_tl_index_test_author = so_type_num + pr_reaction_num + pr_pmi_num
  var_num_tl_index_reaction = pr_reaction_num + so_type_num

  /* Found it convenient to not read in background file for this code. So, so_grps is not available.
  Does not look like we need this as an independent variable anyway. */

  @#if 0
    var_num_tl_index_source_group = so_grps
  @#else
    var_num_tl_index_source_group = int_unused
  @#endif

```

```

var_numtl_index_zone_ind_1 = zone_index_max1
var_numtl_index_zone_ind_2 = zone_index_max2

```

```

return
end

```

arg-type: [1.11](#).

assert: [1.6](#), [1.7](#), [1.8](#), [1.9](#), [1.10](#), [1.11](#), [1.12](#), [1.14](#).

b: [1.6](#).

char_undef: [1.6](#), [1.12](#).

char_uninit: [1.6](#).

char_unused: [1.15](#).

clear_tally: [1.5](#), [1.13](#).

conversion_packages: [1.4](#), [1.6](#), [1.8](#), [1.10](#), [1.11](#), [1.14](#).

conversions: [1.6](#), [1.8](#), [1.10](#), [1.11](#).

cv_action: [1.4](#), [1.11](#), [1.14](#).

cv_max_package: [1.3](#), [1.6](#), [1.8](#), [1.11](#), [1.14](#).

cv_max_parameter: [1.4](#), [1.6](#), [1.11](#), [1.14](#).

cv_name: [1](#), [1.6](#), [1.8](#), [1.14](#).

cv_out_Pa_to_mTorr: [1.2](#), [1.14](#).

cv_out_scale_problem_sp_mass: [1.2](#), [1.14](#).

cv_out_scale_test_mass: [1.2](#), [1.14](#).

cv_out_scale_volume: [1.2](#), [1.14](#).

cv_out_three_halves: [1.2](#), [1.14](#).

cv_package_unknown: [1.2](#), [1.6](#).

cv_partner_1: [1.4](#), [1.8](#), [1.11](#), [1.14](#).

cv_partner_2: [1.4](#), [1.11](#), [1.14](#).

cv_post_divide_number: [1.3](#), [1.14](#).

cv_post_scale_test_mass: [1.3](#), [1.14](#).

cv_post_v_to_internal: [1.3](#), [1.14](#).

cv_scaler_1: [1.4](#), [1.11](#), [1.14](#).

cv_scaler_2: [1.4](#), [1.11](#), [1.14](#).

cv_scaler_3: [1.4](#), [1.11](#), [1.14](#).

cv_track_v_to_external: [1.2](#), [1.14](#).

cv_type: [1.4](#), [1.11](#), [1.14](#).

de_common: [1.6](#).

de_grps: [1.6](#), [1.10](#).

de_lookup: [1.6](#).

de_max_bins: [1.6](#).

declare_varp: [1.6](#).

define_tally: [1.6](#), [1.10](#), [1.11](#), [1.14](#).

define_varp: [1.6](#).

degas2: [1](#).

dep_var: [1.6](#), [1.10](#), [1.11](#).

dep_var_equiv: [1.6](#), [1.10](#).

description: [1.12](#).

detector_num_views: [1.10](#).

detector_tab_index: [1.10](#).

diag_lookup: [1.6](#).

diagnostic_num_sectors: [1.10](#).

diagnostic_tab_index: [1.10](#).

diskin: 1.6.
divide_number: 1.3.

e: 1.6.
eof: 1.6.
equivalents: 1.6, 1.10.
est: 1.11.
est_name: 1.6, 1.9.
estimator: 1.6, 1.9, 1.10, 1.11.

file: 1.6.
FILE: 1.
fileid: 1.12.
FILELEN: 1.6, 1.12.
filename: 1.6.
filenames_array: 1.6, 1.12.
form: 1.6.

generic: 1.6, 1.10.
geometry: 1.6, 1.10, 1.11.
gi_common: 1.6.

hweb: 1, 1.6.

i: 1.6, 1.11, 1.15.
i_var0: 1.11.
icv: 1.6.
implicit_none_f77: 1.5, 1.6, 1.11, 1.12, 1.13, 1.14, 1.15.
implicit_none_f90: 1.5, 1.6, 1.11, 1.12, 1.13, 1.14, 1.15.
in: 1.
indep_var: 1.6, 1.7, 1.9, 1.10, 1.11.
index: 1.10, 1.11.
init_tally: 1, 1.5, 1.6.
init_var0_list: 1.
INT: 1.6.
int_lookup: 1.9.
int_unused: 1.6, 1.15.
ionize_suppress: 1.

len: 1.6.
length: 1.6, 1.7, 1.8, 1.9, 1.10.
line: 1.6, 1.7, 1.8, 1.9, 1.10.
LINELEN: 1.6, 1.12.
loop1: 1.6.

mass_out: 1.
max: 1.6.
max_bins: 1.6.
momentum_vector: 1.

name: 1.6, 1.10, 1.11.
name_equiv: 1.6, 1.10.
NC_CHAR: 1.12.
NC_CLOBBER: 1.12.
nc_decls: 1.12.
NC_GLOBAL: 1.12.

nc_read_elements: 1.5.
nc_read_materials: 1.5.
nc_read_pmi: 1.5.
nc_read_problem: 1.5.
nc_read_reactions: 1.5.
nc_read_species: 1.5.
nc_stat: 1.12.
nc_write_problem: 1.12.
nc_write_tally: 1.5, 1.12.
ncattputc: 1.12.
ncclose: 1.12.
nccreate: 1.12.
ncendef: 1.12.
nconversions: 1.6, 1.11.
next_token: 1.7, 1.8, 1.9.
num_conversions: 1.6, 1.8, 1.10, 1.11.
num_equiv: 1.6, 1.10.

one: 1.11.

p: 1.6, 1.14.
parse_string: 1.6.
particle: 1.6.
pointer: 1.6, 1.10, 1.11.
pr_background_num: 1.15.
pr_common: 1.6, 1.11, 1.12, 1.13, 1.15.
pr_max_equiv: 1.6.
pr_pmi_num: 1.15.
pr_rc_chargex: 1.1, 1.6.
pr_rc_dissociation: 1.1, 1.6.
pr_rc_elastic: 1.1, 1.6.
pr_rc_ionize: 1, 1.1, 1.6.
pr_rc_ionize_suppress: 1.1, 1.6, 1.9.
pr_rc_max: 1.1, 1.6, 1.9, 1.11.
pr_rc_num: 1.6.
pr_rc_plate: 1.1, 1.6, 1.9.
pr_rc_puff: 1.1, 1.6, 1.9.
pr_rc_recombination: 1.1, 1.6, 1.9.
pr_rc_test_ion: 1, 1.1, 1.6, 1.9.
pr_rc_type: 1.6, 1.10.
pr_rc_unknown: 1.1, 1.6.
pr_reac: 1.6, 1.11.
pr_reaction: 1.6.
pr_reaction_num: 1.6, 1.11, 1.15.
pr_test: 1.6, 1.10.
pr_test_num: 1.6, 1.10, 1.15.
pr_ts_rc: 1.6.
pr_var0_list: 1.11.
pr_var0_num: 1.6, 1.11.
program_version: 1.12.

rank: 1.6, 1.7, 1.9, 1.10, 1.11.
rc_common: 1.6.
rc_reaction_type: 1, 1.1, 1.6.

read_geometry: 1.5.
read_string: 1.6.
read_text: 1.6, 1.7, 1.8, 1.9.
readfilenames: 1.5.
real_unused: 1.11.
rf_common: 1.6, 1.12.

sc_common: 1.6.
sc_diag_max_bins: 1.6.
sc_diagnostic_grps: 1.6, 1.10.
scoring_data: 1.6.
scoring_data_max: 1.6.
sec_conv: 1.4, 1.6.
sec_dep: 1.4, 1.6.
sec_est: 1.4, 1.6.
sec_geom: 1.4, 1.6.
sec_indep: 1.4, 1.6.
sec_name: 1.4, 1.6.
sec_skip: 1.4, 1.6.
sec_undefined: 1.4.
set_conversion_packages: 1, 1.6, 1.14.
set_var_list: 1, 1.6, 1.15.
so_common: 1.15.
so_grps: 1.15.
so_plate: 1.6.
so_puff: 1.6.
so_recomb: 1.6.
so_type_num: 1.6, 1.11, 1.15.
SP: 1.6.
sp_common: 1.6.
sp_generic: 1.10.
sp_lookup: 1.10.
sp_sy: 1.10.
sp_z: 1.6.
st_decls: 1.6, 1.11, 1.12.
status: 1.6.
stdout: 1.6, 1.7, 1.8, 1.9, 1.10, 1.11.
string_length: 1.12.
string_lookup: 1.7, 1.8, 1.9, 1.11.
sub_section: 1.6.

tab_index: 1.6, 1.10, 1.11.
tally: 1, 1.11.
tally_base: 1.6, 1.11, 1.13.
tally_cv_action: 1.6, 1.11, 1.13.
tally_cv_num_partners: 1.6, 1.11, 1.13.
tally_cv_partners: 1.6, 1.11, 1.13.
tally_cv_ptr: 1.6, 1.11, 1.13.
tally_cv_scalers: 1.6, 1.11, 1.13.
tally_cv_type: 1.6, 1.11, 1.13.
tally_dep_var: 1.6, 1.11, 1.13.
tally_dep_var_dim: 1.6, 1.11.
tally_est_reaction: 1.6, 1.11, 1.13.

tally_est_test: 1.6, 1.11, 1.13.
tally_geometry: 1.6, 1.11, 1.13.
tally_geometry_ptr: 1.6, 1.11, 1.13.
tally_indep_var: 1.6, 1.11, 1.13.
tally_index_ind: 1.6.
tally_infile: 1, 1.6.
tally_name: 1.6, 1.8, 1.11, 1.13.
tally_num_conversions: 1.6, 1.11, 1.13.
tally_rank: 1.6, 1.11, 1.13.
tally_reac_ind: 1.6.
tally_size: 1.6.
tally_tab_index: 1.6, 1.11, 1.13.
tally_type: 1.6, 1.11, 1.13.
tally_type_base: 1.6, 1.11, 1.13.
tally_type_num: 1.6, 1.11, 1.13.
tally_var_list: 1.7, 1.15.
tally_version: 1.6, 1.12.
tallyfile: 1, 1.12.
tallysetup: 1, 1.5.
tempfile: 1.12.
test: 1.6.
tl_common: 1.6, 1.11, 1.12, 1.13, 1.15.
tl_cv: 1.14.
tl_cv_divide_number: 1.14.
tl_cv_max_conversions: 1.6, 1.11.
tl_cv_max_partners: 1.11, 1.14.
tl_cv_max_scalers: 1.11, 1.14.
tl_cv_output: 1.14.
tl_cv_Pa_per_mTorr: 1.14.
tl_cv_partner_unknown: 1.11, 1.14.
tl_cv_pos_1: 1.14.
tl_cv_pos_2: 1.14.
tl_cv_pos_3: 1.14.
tl_cv_post: 1.14.
tl_cv_problem_sp_mass: 1.14.
tl_cv_scale: 1.14.
tl_cv_scaler_unknown: 1.14.
tl_cv_test_mass: 1.14.
tl_cv_three_halves: 1.14.
tl_cv_to_external_coords: 1.14.
tl_cv_to_internal_coords: 1.14.
tl_cv_track: 1.14.
tl_cv_unknown: 1.11.
tl_cv_volume: 1.14.
tl_cv_zone_pos_1: 1.14.
tl_cv_zone_pos_2: 1.14.
tl_cv_zone_pos_3: 1.14.
tl_decls: 1.6, 1.11, 1.12, 1.13.
tl_dep_var_scalar: 1.11.
tl_dep_var_vector: 1.11.
tl_est_collision: 1.6, 1.9.
tl_est_max: 1.6, 1.9, 1.11.

tl_est_post_process: 1.6, 1.9.
tl_est_track: 1.6, 1.9.
tl_est_unknown: 1.6, 1.11.
tl_geom_detector: 1.6.
tl_geom_surface: 1.6.
tl_geom_volume: 1.6.
tl_index_angle_bin: 1.10, 1.15.
tl_index_detector: 1.10, 1.15.
tl_index_diagnostic: 1.10, 1.15.
tl_index_energy_bin: 1.10, 1.15.
tl_index_material: 1.15.
tl_index_max: 1.7, 1.15.
tl_index_plasma_zone: 1.15.
tl_index_pmi: 1.15.
tl_index_problem_sp: 1.15.
tl_index_reaction: 1.15.
tl_index_sector: 1.15.
tl_index_source_group: 1.15.
tl_index_strata: 1.15.
tl_index_strata_segment: 1.15.
tl_index_test: 1.15.
tl_index_test_author: 1.15.
tl_index_unknown: 1.11, 1.15.
tl_index_wavelength_bin: 1.9, 1.10, 1.15.
tl_index_zone: 1.15.
tl_index_zone_ind_1: 1.15.
tl_index_zone_ind_2: 1.15.
tl_ncdecl: 1.12.
tl_ncdef: 1.12.
tl_ncwrite: 1.12.
tl_num: 1.6, 1.8, 1.11.
tl_rank_max: 1.6, 1.11.
tl_set_base_size: 1.11.
tl_tag_length: 1.6, 1.14.
tl_type_max: 1.6.
tl_type_reaction: 1, 1.6, 1.9, 1.10, 1.11.
tl_type_sector: 1, 1.6, 1.9, 1.10, 1.11.
tl_type_test: 1, 1.6, 1.9, 1.11.
tl_type_undefined: 1.6.
trim: 1.6, 1.10, 1.11.
type: 1.6, 1.9, 1.10, 1.11.
unit: 1.6.
var_alloc: 1.6.
var_free: 1.6, 1.13.
var_num: 1.6, 1.10, 1.15.
var_realloca: 1.11.
var_reallocb: 1.6.
zero: 1.11.
zn_common: 1.6, 1.15.
zn_index: 1.
zn_num: 1.15.

zone_index_max: 1.15.

⟨ Call Define Tally 1.10 ⟩ Used in section 1.6.
⟨ Conversions 1.8 ⟩ Used in section 1.6.
⟨ Estimators 1.9 ⟩ Used in section 1.6.
⟨ Functions and Subroutines 1.6, 1.11, 1.12, 1.13, 1.14, 1.15 ⟩ Used in section 1.5.
⟨ Indep Vars 1.7 ⟩ Used in section 1.6.
⟨ Memory allocation interface 0 ⟩ Used in sections 1.13, 1.11, and 1.6.

COMMAND LINE: "fweave -f -i! -W[-ykw700 -ytw40000 -j -n/
/u/dstotler/degas2/src/tallysetup.web".

WEB FILE: "/u/dstotler/degas2/src/tallysetup.web".

CHANGE FILE: (none).

GLOBAL LANGUAGE: FORTRAN.