WORM User Manual

Version 1.7.0

TABLE OF CONTENTS

1. Introduction

WORM (Write One, Run Many) is an easy to use, cross platform, embedded and extensible, functional programming language designed to facilitate the creation of input-decks for computer codes that use standard ASCII text files for input description. WORM makes it easy to create generic (yet, complex and powerful) reusable models. Additionally its nature allows for complex calculations and routines to be coded once and easily reused, further simplifying the creation of input decks.

1.1 Versions

Version	Date	Comments
1.6.7	2008-01-03	Developed by Tom Jones.
1.6.8	2020-08-24	Added keyword 'modelname'.
1.6.9	2020-10-19	Revised the loop logic for lists to ensure the terminal value is included (versus being omitted due to round-off).
1.7.0	2021-10-18	Completed the help and version options for the WORM command line. Added the library functions (the read command and the command line options –t, -T, -l, and –r) to WORM.

1.2 Platform

WORM requires Perl version 5.004 or later.

1.3 Notation

The notation used in this manual includes:

1. Command and model lines are shown in `Courier New` font.
2. Generally, uppercase text indicates user-specified information. For example,
   - `CHARACTER(S)`
   - `NUMBER`

   Underlining is sometimes used to indicate user-specified information.
3. Square brackets, "[ ]", indicate optional input. Square brackets may be nested; to use an inner option, the outer option must also be used.

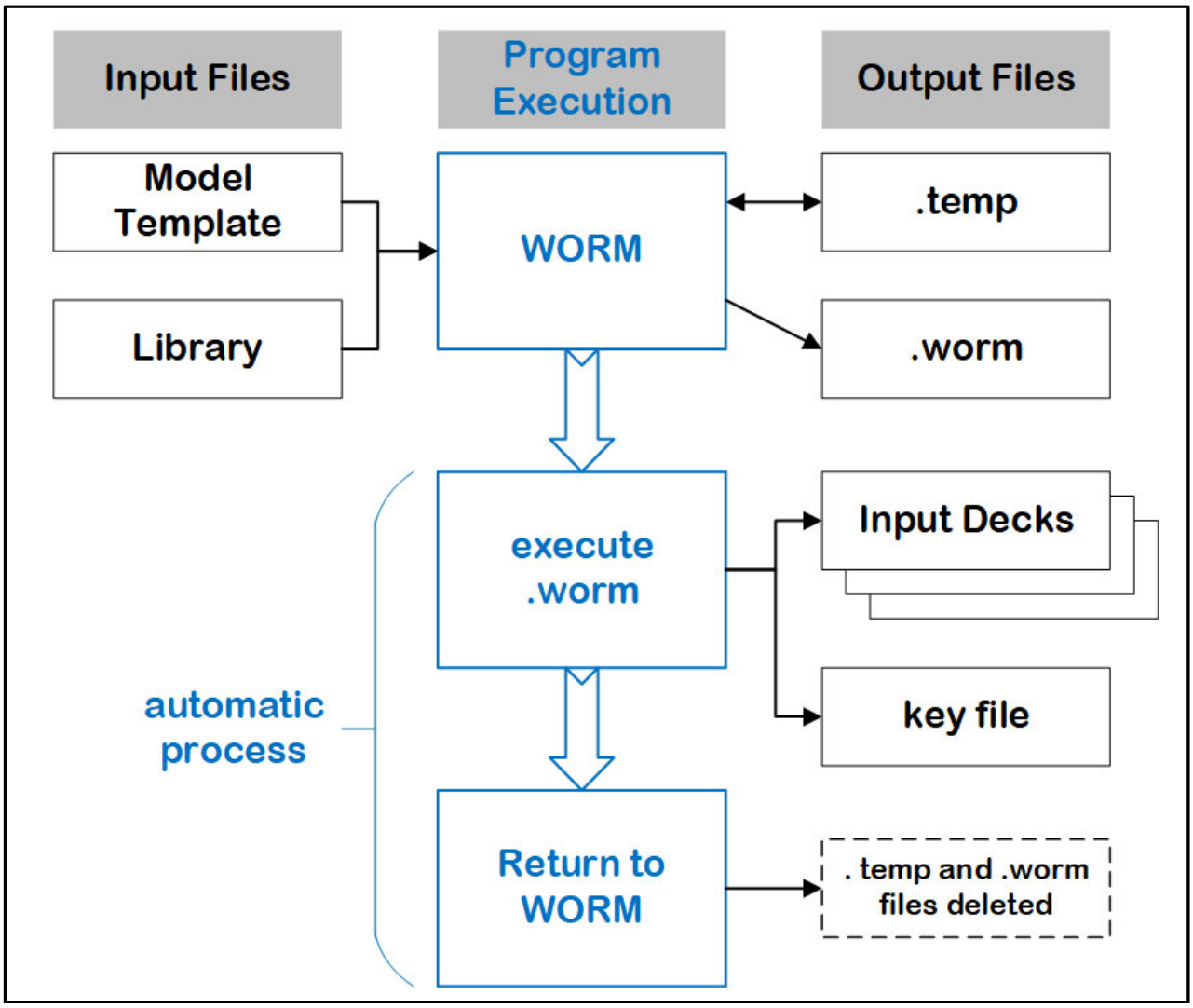
2. Program Flow

WORM is composed of two parts, one is permanent and the other is temporary. These two parts are:

1. "WORM" is a permanent perl program that does the majority of the work. It parses the model file and creates a ".worm" file which is a perl program. It then executes the perl program ".worm". By default, "worm" then deletes ".worm".

2. ".worm" is a temporary perl program that is created and automatically executed by worm. It produces the input-decks described by the model template.

This logic is also illustrated in the following figure.



As illustrated above, the input to WORM is a model template file. WORM will also get input from one or more files in a library directory when the model template contains one or more *read* commands. The output files are the input deck files and a key file. The .worm file is an output of WORM which is subsequently executed by WORM. The .temp file is a temporary file for processing the *read* commands before creating the .worm file. Typically, WORM deletes the .temp and .worm files but either deletion step can be suppressed with a command line option.

2.1.1 WORM Output Filenames

Note: To use the mcnp_out_to_xls post-processor, the "long" format (i.e., the default format) is necessary with the input deck and key filenames.

2.1.1.1 Long Format

By default, WORM uses the long format for the output filenames. WORM produces a 'key' file to document the list names in the order processed. WORM (with the default delimiter and suffix, see Section 3.2.2) will name the key file as follows:

```
ModelFileName_List1_List2_..._key
```

Where `List1` is the name of the first list, `List2` is the name of the second list, etc. The list names may not appear in the order in the model template file, see Section 2.1.1.2 below.

WORM (with the default delimiter and suffix, see Section 3.2.2) will name the output files as follows:

```
ModelFileName_Val1_Val2_..._in
```

Where `Val1` is the value from the first list, `Val2` is the value from the second list, etc., according to the order of the lists as given by the key filename. By default, the full values are printed in the output file name, i.e., calculated real values may have 15 digits. The `-p` option (see Section 3.2.2) will set the precision (number of significant figures) for values in the output filename *and unformatted numbers in the file*. Number formats in the WORM file do not format the values in the output filename.

2.1.1.2 Order of Lists

In the user-generated file, WORM does not require names[1] to be defined before being used. This is because WORM automatically sorts the model template statements, as necessary, to define a name before it is used. The order of the list names in the key filename is determined by the sorted model template statements. If the material pre-processor is used with the model template file, the material pre-processor names are listed first.

One method for the user to set the order of the list names in the key filename is to construct the model template file with the list names in the desired order and to define any other names before the name is used.

---

[1] Names in a model template are similar to variables in other programming languages in that a value can be assigned to a name. However, after a value is assigned to a model template name, a different value cannot be assigned to that name (unlike a program variable). Because of this difference, model templates names are not referred to as variables in this manual.

2.1.1.3 Short Format

By command line option, the user may specify the short format for the output filenames. With the short format (and the default delimiter and suffix, see Section 3.2.2), the 'key' file produced is named:

```
ModelFileName_key
```

WORM (with the default delimiter and suffix, see Section 3.2.2) will name the output files as follows:

```
ModelFileName#_in
```

Where # is a sequential number from 1 to the total number of input-deck files.

3. Command Line

3.1 Format

To execute worm, the command line is:

```
worm [OPTION …] MODEL …
```

where

`OPTION …` is one or more of the options listed below, and

`MODEL …` is one or more model filenames.

Note: Avoid using the delimiter character (see Sec. 3.2.2 below) in the filename of the model template. I.e., if the default delimiter is used, avoid using the underscore (_) in the filename. Using the delimiter character in the filename does not affect WORM but does affect the organization of the results table created by the post-processor mcnp_out_to_xls.

3.2 Options

3.2.1 Toggle Options

These options are used to either turn on or turn off certain behaviors of WORM. The capitalized options shown in bold are the default values.

Command line	Option	Comment
`-h`	Help	Display brief help information and terminate execution.
`-H`	No help (default)	Do not display the help information; this overrides the –h option.

Command line	Option	Comment
`-k`	Short keyfile name	Use the short key name format (see Sec. 2.1.1.3).
`-K`	Long keyfile name (default)	Use the long key name format (see Sec. 2.1.1.1).
`-l`	Library listing	Lists the contents of the library directory.
`-L`	No library listing	Do not list the contents of the library directory.
`-n`	Short input-deck name	Use the short input-deck name format (see Sec. 2.1.1.3). Also forces the short key name format.
`-N`	Long input-deck name (default)	Use the long input-deck name format (see Sec. 2.1.1.1).
`-t`	Keep ".temp"	Do not delete the temporary file ".temp".
`-T`	Delete ".temp" (default)	Delete the temporary file ".temp".
`-v`	Version	Display version and terminate execution.
`-V`		• Without model name, same as `-h`. • With model name, no effect.
`-w`	Keep ".worm"	Do not delete the temporary file ".worm". This can be useful for debugging your model.
`-W`	Delete ".worm" (default)	Delete the temporary file ".worm".
`-x`	No keyfile	Do not create a keyfile.
`-X`	Keyfile (default)	Create a keyfile.

To use the mcnp_out_to_xls post-processor, the "long" format, i.e., the default format, is necessary with the input-deck and key filenames.

3.2.2 Options with Arguments

Command line	Option	Comment	Default
`-d CHARACTER(S)`	Delimiter	Use the supplied character(s) as a delimiter in the input-deck filenames.	"_"
`-j [left, center, right]`	Justification	The justification to use when none is specified.	"left"

Command line	Option	Comment	Default
`-p NUMBER`	Precision	The precision (# of significant figures) to use for unformatted numbers[2].	15
`-r FILENAME`	Read	Read and display the library file.	N/A
`-s CHARACTER(S)`	Input-deck suffix	The character(s) to use as a suffix on the input-decks that the WORM creates.	"in"
`-S CHARACTER(S)`	Keyfile suffix	The character(s) to use as a suffix on the keyfile.	"key"

4. Models

A WORM model is essentially a standard input deck with some of its numerical values (or text) replaced with WORM code.

The model is a text file containing a template of the application (e.g., MCNP) input files with embedded WORM code. Except for quiet lines and *read* commands (see below), WORM writes each line of the model template to the WORM output files, i.e., input deck files. However, before the line is written, the WORM code embedded in the line is evaluated. In this manner, the template defines what your application input file(s) will look like.

With the list notation (see Section 4.5.2 below), multiple values can be sequentially assigned to a name. WORM creates an input deck for each value of the name. If multiple lists are used, WORM steps through each list individually, i.e., WORM creates input decks corresponding to each and every permutation of the list values.

If a pair or group of names should change concurrently, define the values in arrays (see Section 4.5.3 below) and use an index value defined by a list to increment through the values.

4.1 Quiet Lines

Lines that have a pound sign (#) in the first column will not be echoed in the input deck files created by WORM. However, any WORM code in the line is evaluated.

4.2 Continuation Lines

WORM does not have continuation lines. The entirety of each WORM statement must fit on a single line.

---

[2] Including numbers that are in the output filenames (See Section 2.1.1.1).

4.3 Read Command

The *read* command has the following format. Note, the command is case sensitive, i.e., lowercase only, and there cannot be any whitespace separating the '<' and 'read'.

```
<read FILENAME>
```

WORM will replace this command line with the contents of the file FILENAME in the library directory. Because blank lines has special significance to MCNP, any blank line in file FILENAME are not inserted. The *read* commands are processed first, therefore, the library files may contain WORM code, however, the library files cannot contain the *read* command.

The location of the library, i.e., the directory name, is specified by the environment variable wormlibr. One way to see the names of the available library files is to execute WORM with the -l option. One way to see the contents of a specific library file is to execute WORM with the -r option.

As WORM searches for and processes the *read* commands, the file .temp is created. File .temp is normally deleted by WORM but can be kept by the -t command line option.

4.4 WORM Code

The WORM code is enclosed within angle brackets and has the following form:

```
< [[NAME] =] CHANGE [| OPTIONS] >
```

or

```
< [[NAME] =] CHANGE [\ OPTIONS] >
```

where

NAME — WORM defines NAME as the evaluated CHANGE. Names may contain any alphanumeric character or the underscore character ("_"); however names must begin with a letter or the underscore. Names are case sensitive. A name can be defined only once by WORM.

= — required if a name is specified, otherwise optional.

CHANGE — Changes can be a numerical value, name, list definition, array definition, WORM constant, or expression. An expression is an equation of numbers, names, constants, operators, and functions. Numbers can be expressed in either standard or scientific notation (e.g., 0.025, 2.5e-2, 1000000, 1e6).

OPTIONS — is a list of options and format specification (see below for more information).

When WORM processes the model template, any code not in a quiet line (Sec. 4.1) is replaced with the evaluated CHANGE.

There can be multiple WORM code statements in a line of the model template. But each NAME can be assigned a value by only one statement, i.e., the value for a NAME cannot be changed. WORM ignores any whitespace (e.g., spaces and tabs) within the WORM code.

4.4.1 Options

The OPTIONS field is used to format the evaluated CHANGE.

	Format Options
N.M	a numeric field with N digits before the decimal point and M digits after
N	a numeric integer field with N digits
f	full precision (~16 significant figures)
Ns	a numeric field expressed in scientific notation with N significant figures
N#	an integer numeric field N characters long
Nl	a left justified text field N characters long
Nc	a centered text field N characters long
Nr	a right justified text field N characters long

	Miscellaneous Options
i	force integer (1.0001 = 1, 1.5 = 1)
n	include value in the input deck filename[3]

WORM does not prevent using a text format option with a numeric variable, but the results are not reliable. In this case, WORM/perl converts the number to text with the necessary number of digits to accurately represent the number and then prints the text according to the text format. If the text format option has a length shorter than the text representation of the numeric value, digits/characters will be truncated.

Example of numeric value printed with text format option (Value = -4.99999999999e-05)	
Format option	Output
18l	-4.99999999999e-05
14l	-4.99999999999

[3] The value of each shorthand list is included in the input deck filename by default.

4.5 Numbers

4.5.1 Scalar

A scalar is a CHANGE or NAME with a single value.

4.5.2 List Shorthand

With the list notation, multiple values can be sequentially assigned to a NAME. WORM creates an input deck for each value of the NAME. If multiple lists are used, WORM steps through each list individually, i.e., WORM creates input decks corresponding to each and every permutation of the list values.

Change expression	Result
x:y:z	x to y in increments of z. Example: `1:4:0.5 = 1, 1.5, 2, 2.5, 3, 3.5, 4`
x:y:linz	z linearly interpolated points between x and y, inclusive. I.e., values are $x + (i - 1) * dx$ for $i = 1$ to z where $dx = (y - x)/(z - 1)$. Example: `1:2:lin6 = 1, 1.2, 1.4, 1.6, 1.8, 2`
x:y:xlinz	z linearly interpolated points between x and y, exclusive. I.e., values are $x + i * dx$ for $i = 1$ to z where $dx = (y - x)/(z + 1)$. Example: `1:2:xlin4 = 1.2, 1.4, 1.6, 1.8`
x:y:logz	z logarithmically interpolated points between x and y, inclusive. I.e., values are $\exp(\log(x) + (i - 1) * \log(dx))$ for $i = 1$ to z where $\log(dx) = (\log(y) - \log(x))/(z - 1)$. Example: `1:1000:log4 = 1, 10, 100, 1000`
x:y:xlogz	z logarithmically interpolated points between x and y, exclusive. I.e., values are $\exp(\log(x) + i * \log(dx))$ for $i = 1$ to z where $\log(dx) = (\log(y) - \log(x))/(z + 1)$. Example: `1:1000:xlog2 = 10, 100`

Units cannot be directly included in a shorthand list. Units can be combined with a shorthand list by either defining another name that combines the list value with the unit, or including the unit with each reference of the name.

4.5.3 Arrays

Names preceded by "@" are array definitions. The elements of the array are set by comma separated list. The list also sets the dimension of the array. An element of the array is accessed

by the array name (without the "@") and the index number in square brackets. For example, the WORM code:

```
<@radius=2,5,7,1,0.5>
```

creates an array with the elements `radius[1] = 2` and `radius[5] = 0.5` and all the array elements in between.

4.5.4 Constants

The following names are defined by WORM and can be used in `CHANGE` expressions. The model template cannot redefine the WORM-specified constants, i.e., use the constant name as a `NAME` specification.

Mathematical Constants
pi = 3.14159265358979 (from atan(1)*4)
e = 2.71828182845905 (from exp(1))
an = 0.60221367
aN = 6.0221367e23
bit = 0.0001 (a small number, as in a little bit)

4.5.5 Units

The following unit names are defined by WORM and can be used in `CHANGE` expressions. The model template cannot redefine a unit name, i.e., use the unit name as a `NAME` specification. A unit name is applied with the multiplication (or division) operator. For example, a one-inch diameter could be defined with `<dia = 1*in>`.

Unit Conversion Factors	
cm = 1	g = 1
mm = 0.1	kg = 1000
m = 100	lb = 453.59237
in = 2.54	oz = 28.349523125
ft = 30.48	
yd = 91.44	rad = 1
mil = 0.00254	deg = pi/180
cc = 1	sec = 1
l = 1000	min = 60
ml = 1	hr = 3600
gal = 3785.411784	day = 86400
ozfl = 29.5735295625	yr = 31556925.9747

4.5.6 Predefined Names

The following names are defined by WORM and can be used in CHANGE expressions. The model template cannot redefine a predefined name, i.e., use the name as a NAME specification.

Predefined Names	
dateANDtime	= the current date and time
time	= the current time
date	= the current date
wday	= the current day of the week
month	= the current month
mday	= the current day of the month
hour	= the hour component of the current time
minute	= the minute component of the current time
second	= the second component of the current time
year	= the current year
modelname	= the name of the model (WORM input filename) (beginning with version 1.6.8)

4.6 Text

WORM can assign a text string to a NAME. The text string can be enclosed with single quotes (') or double quotes ("). However, WORM must have a format specification whenever the text string is written; the assignment and reference statements must include a text format

specification or be on a quiet line. If the format length is shorter than the string, the string is truncated.

WORM strips any whitespace (spaces, tabs, etc.) from the text string. Therefore, it is practical to use a `NAME` for only single words.

4.7 Operators

The following operators are available for use in WORM code.

Symbol	Operation
+	addition
−	subtraction
*	multiplication
/	division
^	exponentiation
%	modulo (the remainder after division, for example 7%5=2)
.	concatenation

4.8 Functions

The following functions are available for use in WORM code. The model template cannot use a function name as a `NAME` specification.

Function	Description
`(...)`	parentheses can be used to group things
`cos(), sin(), tan()`	trigonometric functions; argument in radians
`acos(), asin(), atan()`	Inverse trigonometric functions; answer in radians
`cosh(), sinh(), tanh()`	hyperbolic functions
`log(), ln()`	logarithm base 10 & base e
`abs()`	absolute value
`int()`	the integer portion of a number
`rand()`	random number between 0 and the supplied argument

4.9 WORM Variables with a MCNP Source in a Repeated Structure or Lattice Geometry

When the source is specified in a repeated structure part of the geometry, the CEL parameter on the SDEF card must have a value that is a path, enclosed in parentheses, from level *n* to level 0 (i.e., the highest level), where *n* is not necessarily the bottom level:

$$\texttt{CEL} = (c_n < c_{n-1} < \ldots < c_0)$$

If c_i is one specific element in a lattice, it is indicated as:

…<c_i [i j k]<…

WORM cannot differentiate between the left angle bracket as used in an MCNP source definition and in WORM code definition. For example, the following line results in a WORM error

```
si4 l (101 < 104[0:<n|2#> 0:<n|2#> 0:<n|2#>] < 105)
sp4 1 <n^3-1|4#>r
```

because the first left angle bracket is paired with the first right angle bracket. The enclosed text, `104[0:<n|2#`, is not a valid WORM construct, i.e., not valid code for WORM.

However, WORM looks for bracket pairs on each input line, i.e., brackets on different input lines are not paired by WORM. The WORM error is avoided by placing the brackets for MCNP purposes and the brackets for WORM purposes on different lines. For example, a functional version of the previous input line is:

```
si4    l (101 <
          104[0:<n|2#> 0:<n|2#> 0:<n|2#>]
          < 105)
sp4    1 <n^3-1|4#>r
```

4.10 Subroutines

You can define your own functions/operators by using standard perl subroutines.

Subroutine definitions must start with "<perl>" on a line by itself. Similarly, it must end with "</perl>" on a line by itself.

In the WORM code (change specification), subroutine names are prefaced by an ampersand ("&"). Subroutines that are called with multiple arguments need to have the arguments separated with commas or semicolons (";").

4.10.1 Minimum Function Example

The following code returns the minimum value found in the function arguments. The function call can contain any number of arguments. The standard function name `min` could not be used because WORM uses `min` as a unit conversion factor (minutes to seconds).

```
sub minfct
# return the minimum of the function arguments
# the function call can contain any number of arguments
{
  my($minimum) = shift @_;
  foreach (@_) {
    if ($_ < $minimum) {$minimum = $_}
  }
  return $minimum;
}
```

4.10.2 Maximum Function Example

The following code returns the maximum value found in the function arguments. The function call can contain any number of arguments.

```
sub maxfct
# return the maximum of the function arguments
# the function call can contain any number of arguments
{
  my($maximum) = shift @_;
  foreach (@_) {
    if ($_ > $maximum) {$maximum = $_}
  }
  return $maximum;
}
```

4.10.3 Lattice Parameters Function

The following code returns the parameters for one dimension of a lattice centered on the origin. Additionally, the whole lattice may be translated, just include the cell translation distance in addition to the distance the window is being translated.

```
sub lattice_parameter
# returns a parameter for a linear lattice centered on the origin
# input
#   $_[0] = number of cells
#   $_[1] = cell length
#   $_[2] = index to the output parameter
# output
#   $_[2] = 0: array length
#   $_[2] = 1: minimum index number
#   $_[2] = 2: maximum index number
#   $_[2] = 3: minimum cell coordinate
#   $_[2] = 4: maximum cell coordinate
#   $_[2] = 5: minimum array coordinate
#   $_[2] = 6: maximum array coordinate
#   $_[2] = 7: cell translation distance
{
  my($n,$l,@parm);
  $n = $_[0];
  $l = $_[1];
  $parm[0] = $n*$l;                          # array length
  $parm[2] = ($n-$n%2)/2;                    # maximum index number
  $parm[1] = $parm[2] - ($n-1);              # minimum index number
  $parm[3] = -0.5*$l;                        # minimum cell coordinate
  $parm[4] = -$parm[3];                      # maximum cell coordinate
  $parm[5] = -0.5*$parm[0];                  # minimum array coordinate
  $parm[6] = -$parm[5];                      # maximum array coordinate
  $parm[7] = ($parm[1]+$parm[2])*$parm[4];   # cell translation distance
  return $parm[$_[2]];
}
```

4.10.4 Table Lookup Example

The following example used subroutine `seed` to return a value from a 600 element array (although only the first and last lines of the array are listed).

```
#  Godiva problem for uncertainty analysis
#
#  <case=1:600:1>
#
#  <rs=&seed(case)>
#
Godiva case #<case | 3> seed=<rs | 18>
1 1 -18.74 -1     imp:n=1  $ uranium core
2 0         1     imp:n=0  $ external void

1 so 8.741

m1   92235.80c  -93.71
     92238.80c   -5.27
     92234.80c   -1.02
c
kcode 20000 1 50 450
ksrc 0 0 0
c
print
rand seed=<rs | 18>
#
<perl>
sub seed {
#
#  seeds selected from known 18, 17, & 16-digit prime factors
#
  my @num = qw(''
    100055128505716009 100707069199565201 101210328665281103
 …
      9846347753151361   9876324585966499   9989112328612357
  );
  return $num[$_[0]];
}
```

4.10.5 Extract Number from a String

The following subroutine will extract a number from a character string given the position of the number within the string and the character length of the number. The number can be either an integer or a real value.

```
<perl>
sub number_string
#
# subroutine arguments
#   1. input string
```

```
#   2. starting position of the number
#      (the first character is position 0)
#   3. number of characters in the number
{
  return 0+substr($_[0],$_[1],$_[2]);
}
</perl>
```

4.10.6 Extract a Number from the WORM filename

Beginning with version 1.6.8, the number_string subroutine above can be used to extract a number from the WORM filename. For example, for the following filename (after the character ruler):

```
          1         2         3         4
01234567890123456789012345678901234567890
example reflector thickness = 10.0 cm
```

The reflector thickness can be set within the WORM file by:

```
<refl_thick = &number_string[modelname,30,4]>
```

4.11 Errors

4.11.1 Understanding Error Messages

Errors can occur during the execution of the .worm file (Perl program). Error messages frequently refer to the line number in the .worm file. However, the default setting is for WORM to delete the .worm file; the line number points to a non-existent line. In this case, it is useful to repeat the WORM command with the addition of the `-w` option to keep the .worm file. Then the .worm file can be inspected to find the WORM code that results in an error.

4.11.2 Truncated Filenames

WORM may attempt to create files whose filenames are too long for the operating system. In this case, the operating system will truncate the filenames. There will not be an explicit error message when the operating system truncates the filename.

Solution Option 1

If the filename has many embedded values that are being displayed to full precision, the filename can be shorten by using the `-p #` (where # is the number of digits) option in the Worm command. The `-p` option sets the default precision for values in the filename *and inside the file*. Note, inside the file any variable references without a format specification will be formatted in the output files according to the Worm command option. However, the `-p` option does not affect any values with a format specification. (But the format specification inside a file does not affect the filename.)

An example is a file named `test_format` containing the range variable `<a = pi:4*pi:lin4>`. The Worm command with default options creates:

Filename	File Contents
test_format_3.14159265358979_in	3.14159265358979
test_format_6.28318530717958_in	6.28318530717958
test_format_9.42477796076937_in	9.42477796076937
test_format_12.5663706143592_in	12.5663706143592

With option `-p 4`, Worm creates:

Filename	File Contents
test_format_3.142_in	3.142
test_format_6.283_in	6.283
test_format_9.425_in	9.425
test_format_12.57_in	12.57

Solution Option 2

If Option 1 does not work, WORM needs to be executed with the `-n -k` options (see Section 3.2.1) to shorten the filenames created.

4.11.3 Error Messages Related to WORM Output Filenames

If WORM attempts to create a filename that is too long, perl will create multiple error messages similar to the following two lines.

```
print() on closed filehandle LEGEND at /…/worm line …, <MODEL> line ….

write() on closed filehandle INPUT_DECK at .worm line ….
```

In this case, WORM needs to be executed with the `-n -k` options (see Section 3.2.1) to shorten the filenames created.

5. MCNP_OUT_TO_XLS

Note: To use the mcnp_out_to_xls post-processor, the "long" format (i.e., the default format) is necessary with the input deck and key filenames.

MCNP_OUT_TO_XLS is a Linux shell script to list the MCNP output from the cases generated by WORM. Although the script name refers to the Excel file format, the file produced is a tab separated text file. The key file generated by WORM is used to identify the MCNP output files to read and parse. MCNP_OUT_TO_XLS generates an output table. The first columns of the table are the case parameters, i.e., the parameters that appear in the filenames. The subsequent columns are the MCNP results:

1. k_{eff}
2. k_{eff} uncertainty (sigma)
3. average neutron energy causing fission (ANECF)
4. energy of average neutron lethargy casusing fission (EANLCF)
5. fraction of neutrons causing fission that are:
    a. thermal (< 0.625 eV)
    b. intermediate (0.625 ev – 100 kev)
    c. fast (> 100 keV)
6. Shannon entropy test result

When executed, MCNP_OUT_TO_XLS uses another shell script file (gmks), the Perl interpreter, and the Perl script (file) worm_sort.