RandomGen and Library Writing Manual

(RandomGen v3.5.1 or Later)

Introduction

1 General Information

- RandomGen is a utility for generating random text based on a set of tables. As such, it lends itself well to automating the process of table-hopping that occurs during pencil and paper role-playing games when there is a need to generate random treasure, monsters or adventure ideas etc...
- The random text is generated using directives in tables that make up what are known as libraries. The libraries are text files that serve as a way to group together tables with a common purpose. For example, one could create a library that contains tables for the purpose of generating random treasure and another for random adventure ideas. It is the directives in the library tables that control the flow of table-hopping. Each table line contains a set of directives. These direct RandomGen to other table lines based on randomly generated values (similar to die-rolls) until the procedure is complete. This procedure is analogous to what a person would do when rolling dice on tables in game reference books.

2 RandomGen Graphical User Interface

- The RandomGen main program (file named "RandomGenDist.jar" in the main program folder) is a graphical interface for generating random text using libraries. It requires the Java SE Runtime Environment (JRE) version 6 or later (available at http://www.java.com) to be installed on your system. Most modern computer systems already have this installed by default though it may need to be updated.
- The RandomGen main window has several buttons that are self-explanatory. The other, possibly less obvious fields are described in this paragraph. The drop-down menu in the top-left corner allows the user to choose which library to use. Directly below it is an input text-field where the user can type an initial value or a set of directives. What is typed in here is passed to the chosen library when "Generate" is clicked. Directly to the right of the text-field is a number field. The value in this field designates the number of times RandomGen is to generate random text when "Generate" is clicked. The large text area that takes up most of the RandomGen window is the output area. This is where results of a generation are displayed.
- The RandomGen main window contains tabs to more easily organize generated text. Each tab contains a set of navigation buttons to flip back and forth through the pages generated in the tab. This is similar to the interface of any modern web browser.
- Generated text can be saved and then loaded at a later date. Options to do so exist in the file menu of the main program window. The "Open" option allows one to load previously saved text into the RandomGen main window. The "Save Page" option saves only the single currently visible page to file. The "Save All" option saves all pages and all tabs to a single file.
- The "Variables" button in the program main window brings up the "User Variables" window. Some libraries will have user variables defined and this window allows their values to be modified. The changes are taken into account the next time "Generate" is clicked. If the library directives modify any of these variables during the generation process then these changes will be reflected in this window when generation is complete.

Libraries

1 General Information

- Libraries are text files that contain a set of tables and directives. RandomGen uses libraries as a means to generate random text.
- There are two types of runtimes for libraries, the load-time and the generation-time. At load-time, the library file is read into memory by RandomGen. All tables and variables defined in the library are stored internally in the RandomGen database. Also, any directives that are not confined to tables are executed. This is why one rarely sees #desc or #name directives inside tables. Generation-time refers to when "Generate" is clicked in the RandomGen main window. Table lines are visited and their directives executed. Any directives not confined to tables are ignored at generation-time.
- Each library must contain a table named "main". At generation-time, this table is the starting point for random text generation. When "Generate" is clicked in the RandomGen main window, it is the "main" table in the chosen library that is first visited. RandomGen determines which line of the "main" table to first visit based on the value given in the input text-field. If this field is left blank then RandomGen uses "dietype" in the main table header to determine which line to first enter (see Table Setup in the Tables section below for more information).
- A good way to learn how to create libraries is by having a look at the examples provided with RandomGen. They are located in the "libraries" folder of the program directory. The most basic library is "coins.txt". The use of variables is introduced in "gems.txt". Linking between tables is shown in "treasure.txt". Advanced features are used in "chargen.txt" and "cave.txt". Modify and experiment with these libraries. Use any text editor you like, such as Notepad, TextEdit or jEdit (more on using jEdit in a later chapter). To register changes made to the libraries in RandomGen, save any changes in the text editor and then click "reload libraries" in the "Actions" menu of RandomGen.

2 Libraries and RandomGen

- When the RandomGen main program is launched, it scans the "libraries" folder and loads all files with the ".txt" extension. These files are all expected to be library text files. If any errors are encountered during this load-time process then an error message will be displayed in the RandomGen main window. Clicking "reload libraries" in the "Actions" menu of the RandomGen program window reloads libraries in the same manner.
- Any library that has a name defined at load-time can be chosen from the drop-down menu in the RandomGen main window. A library name is set using the #name directive (see section on directives below). A nameless library is loaded and can be linked from an external library but will not appear in the drop-down menu.

Tables

1 General Information

- All tables have a header, body and footer. The header and footer must begin with a colon ":". They designate the start and end of a table. They each must begin on a new line of text in the library file. Each table line must also be started on a new line of text.
- A table line must appear in its entirety on a single line. But, for the sake of readability, any table line can be continued on the next line of text by starting the new line with the plus sign (+) character. Internally, any such lines will be concatenated with the previous line and be considered a single table line by RandomGen.
- The RandomGen table format was created with table-hopping in mind. So library tables look much like what the tables look like in game reference books. This means that all table lines have a probability range associated with them. This range is always given at the start of a table line.
- One can jump between tables using the link directive. The idea is to simulate what a person would do when rolling dice on tables in game reference books (see Link Directive in the Directives section below for more information).
- The main entry table of the library must be named "main". If a link is made from another library without a table name as an argument, table "main" will be accessed by default.

2 Table Setup

Header:

- A table header has the form

  :tablename [dietype] [directivelist]

  Where tablename is the name given to the table. Parameter dietype is the die that is to be rolled by default when linked to this table. If no dietype is provided the table rolls on the number of lines in the table. Parameter directivelist is a semi-colon delimited list of directives that are to be executed before the directives in the table line.
- The table name must be a single word containing no spaces or any of the following characters "$!@#&();". The dietype parameter must begin with an exclamation mark (!), may not contain any spaces or variables and must follow correct die syntax. See the section on rolling dice below for more information.

Body:

- Each table line is given a probability range. The syntax for a range is "int1-int2", where int1 and int2 are integers and int1 is less than or equal to int2 (no spaces are allowed and both must be greater than 0). A particular table line is entered if the value of the die-roll defined in the header lands in this range. If the same probability is to be assigned to each line then the range can be replaced with a hyphen (-). The probability ranges are defined at the start of each line. Examples of tables and their possible formats are given below:

Probability ranges (format 1)

```
:TableName   !d20
  1-10   line 1 directives etc...
  11-15 line 2 directives etc...
  16-20 line 3 directives etc...
:
```

Same probability range denoted by a "-", (format 2)

```
:TableName
  -line 1 directives etc...
  -line 2 directives etc...
  -line 3 directives etc...
:
```

- When linking to a table, the given dietype in the header is rolled. Based on this roll, the line with the corresponding range is entered. The only exception to this is when the "param" parameter is given in the link directive. If such is the case, the value "param" is treated as the roll value and the corresponding line is entered. Note that for the format 2 type of table, the range for any particular line is equal to the rank of the line. This is to say that the range for line 1 is "1-1", the range for line 2 is "2-2" and so on.

Footer:

- A table footer has the form

  :[directivelist]

  Where directivelist is a semi-colon delimited list of directives that will be executed after the table line has been entered.

Directives

1 General Information

- All directives begin with the pound (#) or ampersand (&) sign and end with a semi-colon (;). For a directive to be recognized at load-time, it must be the first token on line. For a directive to be recognized at generation-time, it must either be the first token on a table line (after the range) or follow a semi-colon (from the directive before it).
- There must be a space or tab between the directive and its first argument and between subsequent arguments unless stated otherwise.
- All directives that start with the pound symbol (#) are allowed anywhere in a library file whereas directives starting with an ampersand (&) are ignored unless they appear in tables.
- In the library text file, directives may be given in either all uppercase letters or all lowercase letters, not a combination of the two.

2 List of Directives

Items in [] are optional.

Common Directives:

- #d [expression];
  Append the reserved character "$" to output. Parameter expression designates the number of these characters to append and can be any mathematical expression as described in the #math directive. This parameter can be as simple as a whole number and cannot be negative. If expression is omitted then this character is only output once.
- #define varname [varval];
  Defines a variable that can be set/used by the library. If the variable was previously defined then this directive will function as the #set directive. If a user variable is redefined with this directive then it will be removed from the "User Variables" window in RandomGen. The varval parameter is the value given to the variable.
- #desc string;
  Sets the description for this library. Clicking "Get Description" in the RandomGen main program window will display what is set with this directive. Multiple usages of this directive append the given string to the description.
- #e [expression];
  Appends the reserved character "!" to output. Parameter expression designates the number of these characters to append and can be any mathematical expression as described in the #math directive. This parameter can be as simple as a whole number and cannot be negative. If expression is omitted then this character is only output once.
- #endl [expression];
  Appends a new-line character to output. Parameter expression designates the number of new-line characters to append and can be any mathematical expression as described in the #math directive. This parameter can be as simple as a whole number and cannot be negative. If expression is omitted then this character is only output once.

- #name namestr;
  Where namestr is the string to be displayed in the drop-down menu of the RandomGen main program window for this library. If the #name directive does not appear in the library then it will not appear in the drop-down menu but can still be linked from other libraries. Subsequent uses of this directive will overwrite the name set by any previous use. Changes to the library name only take effect at load-time and not at generation-time.

- #out [string]
  Appends parameter string to the output. The parameter can be entirely composed of blank spaces. These spaces will be appended to the output.

- #s [expression];
  Appends the reserved character ";" to output. Parameter expression designates the number of these characters to append and can be any mathematical expression as described in the #math directive. This parameter can be as simple as a whole number and cannot be negative. If expression is omitted then this character is only output once.

- #set varname [varval];
  Sets the value of variable varname to varval. If varval is omitted then the value of varname is set to an empty string.

- #tab [expression];
  Appends a tab character to output. Parameter expression designates the number tabs to append and can be any mathematical expression as described in the #math directive. This parameter can be as simple as a whole number and cannot be negative. If expression is omitted then this character is only output once.

Advanced Directives:

- #add varname num;
  Where varname is the name of a defined variable and num is an integer. This directive adds the value in num to the variable varname. For more involved mathematical operations, consider using the #math directive described below.

- #alias name;
  Sets the alias for the library. Where name is the string by which the library may be referred to in a link directive. The parameter "name" may not contain the following reserved characters: "$!()@&#" and must be a single word containing no spaces. Multiple libraries with the same alias can cause the wrong library to be referenced at generation-time.

- #div varname num;
  As #add above but the variable varname is divided by num. Integer division is performed and so any decimal value is dropped. For more involved mathematical operations, consider using the #math directive described below.

- #divc varname num;
  As #div above except that if performing the division does not produce a whole number then the next largest integer is stored in varname (note that the next smallest integer is stored in varname if the result of the division is negative). In other words, this directive stores the result of the ceiling function on varname divided by num (floor function if the division result is negative). For more involved mathematical operations, consider using the #math directive described below.

- #end;
  Prevents execution of any pending directives and prints any output to the RandomGen main program window.
- #math varname [expression];
  Evaluates the given mathematical expression and stores the result in varname. The expression may contain any of the following operators "+-%*/^\" and integer operands (note that "\" behaves as #divc and "%" as #mod). All operations are performed using integer arithmetic. Conventional operator precedence is upheld and the unary negation operator "-" has highest priority. Precedence may be overridden by using parentheses "()". If expression is omitted or if it contains only a series of empty parentheses, an empty string is stored in varname.
- #matho [expression];
  As #math above but the result of the mathematical expression is appended to the output and is not stored in any variable.
- #mod varname num;
  As #add above but the variable varname will hold the remainder of the integer division of varname by num. For more involved mathematical operations, consider using the #math directive described above.
- #mul varname num;
  As #add above but the variable varname is multiplied by num. For more involved mathematical operations, consider using the #math directive described above.
- #sorta [var1 var2 var3 ... varn];
  Sorts a list of variable values in ascending order. The parameters var1, var2, var3 etc. are variable names. For example, if var1=2, var2=3, var3=1 and "#sorta var1 var2 var3;" were called, then variable values would change to the following: var1=1, var2=2, var3=3.
- #sortd [var1 var2 var3 ... varn];
  Same as #sorta except that the values are sorted in descending order.
- #sub varname num;
  As #add above except num is subtracted from variable varname. For more involved mathematical operations, consider using the #math directive described above.
- #uservar varname [varval] [|val1|val2|...|valn|];
  Defines a variable that can be set by the user in the "User Variables" window of RandomGen. The variable's initial value is set to parameter varval. The values delimited by "|" will be selectable from a drop-down menu in the "User Variables" window. If the "|"-delimited list is omitted then the drop-down menu is replaced with a text-field and the user can set the variable to any value.

3 Link Directive:

- The link directive begins with an ampersand (&), ends with a semi-colon (;) and is recognized only when placed in tables.
- General form (items in square brackets "[]" are optional):

  &tablename [@libraryname] [param] [(repeatnum)] [*]; or
  &@libraryname [param] [(repeatnum)] [*];

  Where tablename is the name of the table to link to. Parameter libraryname is the alias of the

library that holds the desired table if linking to an external library. This parameter is not required if linking to a table within the same library. Parameter param is a value passed to the table; it is the line to access in the desired table. Parameter repeatnum is the number of times to consecutively link to the desired table. Note that repeatnum must be in brackets. If the library name is given without a table name then the table named "main" in the library is accessed. If repeatnum is provided and is greater than 1 then adding an asterisk "*" to the directive will cause only distinct table lines to be entered. Of course, if repeatnum is greater than the number of lines available in the table, there will be duplications (but only after each table line has been chosen once). Also note that if the parameters param and repeatnum are both given then the asterisk will not cause distinct table lines to be visited because param designates the single same line to be visited "repeatnum" times.

- The value of param above can be numeric, a string or be omitted altogether. If param is numeric, the table will be entered at the line whose range includes the value of param. Though if param is "0" or omitted then the default roll (see the section on tables above) will determine the line to access. If param is a string then the table will not be accessed. This is useful if one wishes to only access a table if a variable is empty. If a variable is set as a parameter of the link directive then it will only access the table if the value contained in the variable is a number or an empty string (the equivalent of omitting the param argument). Also note that a string passed to the main table will generate an error message.

Using Variables

- The value of a variable can be inserted in any line by using the dollar sign ($) and variable name combination, e.g. $varname. This signals the directive handler to replace the string "$varname" by the variable's value before running the directive. For example, if the value of the variable "x" were previously set to 3 then the line "#out the value in x is $x;" would be interpreted as "#out the value in x is 3;".
- Sometimes, due to the variable's location in a line it must be differentiated from the line using brackets. For example, using x from above in the following way "#out x*10 = $x0;" would be incorrect because the variable name would be registered as "x0". It becomes necessary to use brackets in the following way: "#out x*10 = $(x)0;". If "x" held the value 3 then the line would be interpreted as "#out x*10 = 30;".
- Variables have precedence over die-rolls and so are resolved before them. If more than one variable is placed on the same line, they are resolved from left to right. Precedence can be overridden using parentheses "$()", that is, any die-rolls or variables inside the brackets will be resolved first, before anything outside the brackets. Variables may be "nested" using parentheses in this fashion. An example is provided below.
- As an advanced feature, it is possible to build variable names from other variable values. This may be useful if one wishes to cycle through a set of variables simulating an array. This is done using brackets in a way similar to that described above. For example, the line "$(var$x)" would first have $x replaced with its value, say 3. Internally the line would then look like $(var3). Before processing any directives, the value of var3 (say 10) would replace $(var3) and the line would finally be interpreted as just the number 10. Another example using actual directives:

  ```
  #define a the;
  #define b var;
  #define thevar hello world;
  #out $($a$b);
  ```

  Initially, the variables "a" and "b" are created using the #define directive and are set to the values "the" and "var", respectively. A variable named "thevar" is then created, also with the #define directive, and is set to hold the value "hello world". The last line uses the #out directive to output what comes after it to the RandomGen main window. When this directive is reached, $a and $b are internally replaced by their values, "the" and "var" respectively, to give the following: "$(thevar)". Then "$(thevar)" is replaced with its value "hello world". The directive is then evaluated and this prints "hello world" in the RandomGen main window.
- Note that using brackets with variables when the rest of the line contains bracket characters, not related to the variables, can cause errors or unintended output. To be safe, try to delimit the line with semi-colons, separating the brackets for variables from the in-text brackets. For example:

  ```
  Do:           (some text) ;$(var$stuff); more(text));
  Instead of:   (some text) $(var$stuff) more(text));
  ```

- The variable named "*" is an internal variable that holds the value of the die-roll for a particular instance of a link to a table. It is used as follows in a table line: "-this is line $*". Brackets can be used as usual: "$(*)" is a valid in-line statement. This variable is useful if it becomes necessary to know exactly which value was generated when a line with a large probability range is entered.
- Variable names may not contain the following reserved characters: "$!();" and may not be named "*". Though the name may contain the "*" character.

Rolling Dice

- A die-roll can be inserted anywhere with the exclamation mark (!). This signals the directive handler to replace what comes after the ! by a numeric value that is the result of the die-roll before running the directive. For example, the directive "#set x !d20;" will place the result of a 20-sided die-roll into variable x.
- The general format of a die-roll is the following: ![b]dc[+v|xv], spaces are not allowed. The result is the sum of b die-rolls on c-sided dice. The parameter v may be added to (using "+"), or multiply (using "x") the result of the die-roll. Parameters b, c and v must be integers and in particular, c must be positive or zero. if either b or c is given as "0" then the roll result will be returned as "0" (unless v is added to this result). The following are examples of valid die-roll syntax: !d10x2, !3d6, !4d8+3, !3d10+-3, !-2d4, !d0+3.
- As is the case for variables, brackets may be used to delimit the die-roll from the rest of the line as follows: "value!(d12)isNotMisinterpreted".
- Die-rolls have lesser precedence than variables and so are resolved after them. If more than one die-roll is placed on the same line, they are resolved from left to right. Precedence can be overridden using parentheses "!()", that is, any die-rolls or variables inside the brackets will be resolved first, before anything outside the brackets. Die-rolls may be "nested" using parentheses in this fashion. For example: !(!(d3)d6+2) will cause "!(d3)" to be evaluated first, then that result will be applied to the die-roll contained by the outer parentheses. If "!(d3)" had been determined to be "2" then the resulting die-roll for the example would be: "!(2d6+2)", which would then be evaluated to, say, 10.
- Note that using brackets with die-rolls when the rest of the line contains bracket characters, not related to the die-rolls, can cause errors or unintended output. To be safe, try to delimit the line with semi-colons as explained in the section on variables.

Miscellaneous Information

- The input text-field of the RandomGen main window can accept more complex input than just a single value. It can accept a variable name or a set of directives. A range of possible values can also be specified. In the case of inputting a variable name, include the "$" symbol and be sure that the variable is defined in the chosen library. The value held in the variable will be passed to the main table of the chosen library. If directives are given then these directives are executed in the context of the chosen library. A range of values can be specified with a hyphen (-) or individual values with a comma (,). For example, if one wants to pass a random value between 1 and 5 and possibly 10 and 11 then the following should be typed in the input text-field: 1-5,10,11.
- The #out directive does not always have to be used to append to the output. Any string that does not start with "#" or "&" is directly appended to the output. For example, the table line "here at line!d10;" will be appended to the output as "here at line3" (if 3 were rolled). Such table entries must end with a semi-colon (;) just like any other directive.
- Comments can be inserted anywhere in a library by starting a line with two slashes (//). The two slashes are only recognized as comments when at the start of a line of text.
- When saved files are deleted using the RandomGen open/save windows they are not truly deleted from your system. The files are just given a ".bak" extension and become invisible to RandomGen. These files exist in the "save" folder of the program directory and can be recovered by changing the file name back to having a ".txt" extension.
- Reserved characters are strictly "$", "!" and ";". They should not be used other than for their intended purpose. If needed, directives #d, #e, and #s can be used to output these symbols to the RandomGen console. Certain characters that may cause problems are "()@#&|*". The latter can be included in some names and most lines but must be used with caution since the RandomGen parser may confuse them to mean something else. When using the #out directive, these latter symbols are safe to use without ambiguity.
- If you have any questions about RandomGen or if you would like to share any libraries you have created send an email to randomgen.mail@gmail.com.
- Program website: http://randomgen.site40.net/

Editing Libraries with jEdit

- A text editor known as jEdit is recommended for library editing because it supports useful features such as syntax highlighting and code folding. If you opt to use jEdit then the RandomGen Edit Mode must also be installed to enable these features. Follow these instructions to do so (note that these instruction were tested with jEdit 4.4.2 and may be slightly different for other versions):

    1. Download and install jEdit from http://www.jedit.org. Make note of the installation directory on your system.
    2. Download the Edit Mode package from the RandomGen website (http://randomgen.site40.net/).
    3. Extract the "randomgen.xml" file from the Edit Mode package into the "modes" folder of the jEdit installation directory. Choose to overwrite files if prompted.

       (If the following two steps seem daunting, try, instead, to extract the "catalog" file of the Edit Mode package into the jEdit "modes" folder and choose to overwrite if prompted. Note that this may not work with newer versions of jEdit.)
    4. Start jEdit and open the file named "catalog" located in the "modes" folder of the jEdit installation directory.
    5. Find the "<MODES>" tag located near the top of the file. On the next line, copy and paste the following text or type it in exactly as shown.

```
<MODE NAME="randomgen"
        FILE="randomgen.xml"
         FILE_NAME_GLOB="*.txt"
        FIRST_LINE_GLOB="//RandomGen Library"/>
```

    6. Save the changes and restart jEdit. The RandomGen Edit Mode should now be available.

- For jEdit to automatically recognize RandomGen libraries, the file name must have the ".txt" extension and the first line of the library file must be exactly "//RandomGen Library" (without the quotation marks).
- If jEdit does not perform any syntax highlighting for RandomGen libraries after following these steps, try refreshing the Edit Modes by clicking "Utilities > Troubleshooting > Reload Edit Modes". The Edit Mode can also be selected manually by clicking "Utilities > Buffer Options" and setting the Edit Mode option to "randomgen" in the window that is displayed.