The Randomizer Generators

Table of contents

Generators

The Randomizer Generators

Welcome to The Randomizer Generators help system. Please select an topic below to learn more.

Overview

What are Generators?

Generators are the language of The Randomizer. They determine what content is generated and how. The different applications store the generators in different ways. The web application keeps a database of them while the windows application loads them from xml files with the extension **rgen**. There are a number of different types of generators which are listed below.

- **Assignment Generator** is the most common and feature rich of the generators and should serve most purposes included those performed by other generator types.
- **Dice Generator** is designed to make generators that are for making dice rolls simplified.
- **List Generator** is the simplest of the generators, picks a single item from a list of items.
- **LUA Generator** is a generator powered by LUA scripts. This type of generator is not supported by the web application.
- **Phonotactics Generator** is a generator that uses the concept of Phonotactics to generate words.
- **Table Generator** is designed to mimic the tables found in many role playing source books.

Generator Header

All generator types share some common fields that define them. Below are the fields common to most or all of the generator types.

Property	Description	Values	Required	Notes
name	The name of the generator		✓	
author	The author of the generator			
url	A url included int he generator description			
description	A longer description of the generator's purpose			

tags	A list of tags used to categorize the generator			
supportsMaxLength	Does the generator support limiting the max length of the output			Only applicable to Assignment and Phonotactic generators
parameters	A list of parameters for the user to populate			Not applicable to List and Dice generators
css	CSS to apply to the content generated			

Tags

Tags are used to categorize the generator. In the windows application, these can be altered during runtime in order to categorize generators as you wish. On the web app, they can only be changed by the author of the generator.

Example:

```
<tags>
  <tag>Environment</tag>
  <tag>D&amp;D</tag>
  <tag>Fantasy</tag>
</tags>
```

Parameters

Parameters let a user select different options for a particular generator such as race for a fantasy NPC generator. There are several different types of parameters available as defined below.

Property	Description	Values	Required
name	The name of the parameter		
value	The default value of the parameter		
display	The display name of the parameter		
type	The parameter type	Text, Integer, Decimal, List, Date	
{options}	The options available to select for a List type parameter		

Options

For List type parameters, options provide the items that will populate the combo box the user will be presented with.

Property	Description
display	The text to show in the combo box. If this is not populated, value will be displayed
{value}	The value that will be sent to the generator.

Example :

```
<parameters>
    <parameter name="Gender" display="Gender" type="List" valueType="Label">
      <option>Any</option>
      <option>Male</option>
      <option>Female</option>
    </parameter>
</parameters>
```


Content

The most common and feature rich of the generators and should serve most purposes included those performed by other generator types. It is based upon the concept of Markov Chains.

Items:

Assignment Generators are populated using "Item" definitions that determine what to generate and where to move next. The properties that define an Item are:

Property	Description	Values
name	The label of the item	
weight	The relative probability of selecting this item	
next	The label of the item that will come after this one	Name of the next item to evaluate
repeat	How many times to repeat this item	A calculation expression
{expression}	The value of this item	See the expressions section below

This is an example of the Xml format for the Items section:

```
<items>
	<item name="Start">[Gender][SurName]</item>
	<item name="Any" weight="50">[MaleName]</item>
	<item name="Any" weight="50">[FemaleName]</item>
	<item name="Male">[MaleName]</item>
	<item name="Female">[FemaleName]</item>
	<item name="MaleName" weight="300" next="[Space]">James</item>
	<item name="MaleName" weight="299" next="[Space]">John</item>
	<item name="MaleName" weight="298" next="[Space]">Robert</item>
	<item name="MaleName" weight="297" next="[Space]">Michael</item>
	<item name="SurName" weight="1">Williamson</item>
</items>
```

Expressions:

Expressions are the key to the items. They can be as simple as a string value or as complex as a series of nested labels, strings, calculation, and formatting commands.

Strings

The simplest form of expression is a string. These values are placed on the output as is.

Example:

```
<item name="Start">[FirstName] [LastName]</item>
<item name="FirstName">Lance</item> <!-- This is a string expression -->
<item name="LastName">Boudreaux</item> <!-- This is a string expression -->
```

Example Result:

```
Lance Boudreaux
```

Labels

A label are contained within square brackets (ex [label]) and tell the generator to replace that text with the text from another item.

Example:

```
<item name="FullName">[FirstName] [LastName]</item> <!-- These are label
expressions -->
<item name="FirstName">Lance</item>
<item name="FirstName">John</item>
<item name="LastName">Boudreaux</item>
<item name="LastName">Doe</item>
```

Example Result:

```
Lance Doe
```

Label names can be nested in order to generate a new label name like so:

```
<item name="FullName">[FirstName[Race]]</item>
<item name="Race">Human</item>
<item name="Race">Orc</item>
```

```
<item name="FirstNameHuman">Lance</item>
<item name="FirstNameOrc">Orgol</item>
```

In this case, Race is selected first since it is nested and will be either Human or Orc. Then the full label in the FullName item will be either FirstNameHuman or FirstNameOrc.

Calculations

Calculations look similar to Labels, however; they are denoted using an equal sign after the first square bracket (ex [=2+2]. Calculations can contain basic math, special functions, variables, and parameters. They are fairly complex and are detailed in the Calculations section of this help system. In addition to the available functions, an Item name can be included in the calculation as a variable name.

Example:

```
<item name="Strength">Strength: [=Roll(3,6)]</item>
<item name="NumberOfOrcs">You encounter [=ToText(Roll(2,4)+2)] orcs!</item>
```

Example Result:

```
Strength: 12
You encounter six orcs!
```

Formatting

Output of generators is in HTML format. This allows HTML tags to be included in the item values enabling custom formatting of the output. Note that any HTML contained in an item must either be escaped or contained inside of a CDATA tag.

Example:

```
<item name="Orcs"><![CDATA[<span style="font-
weight:bold;color:red">ORCS!</span>]]></item>
```

Example Result:

```
ORCS!
```

Imports

Another feature of Assignment Generators is the ability to create libraries that can be imported by a generator. These libraries looks like a standard Assignment Generator but have a flag called isLibrary that is True. These items are added to the list of items in the generator as if they were in that file. These imports are contained in their own section of the xml.

Example:

```
<imports>
    <import>USNames</import>
</imports>
```

Example

```
<?xml version="1.0" encoding="utf-8"?>
<generator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xsi:type="Assignment"
version="2">
  <name>Assignment Sample</name>
  <author>Lance Boudreaux</author>
  <description>A sample Assignment Generator</description>
  <tags>
    <tag>Table</tag>
  </tags>
  <supportsMaxLength>false</supportsMaxLength>
  <parameters>
    <parameter name="Color" display="Color" type="List">
      <option display="Any">AnyColor</option>
      <option display="Blue">Blue</option>
      <option display="Red">Red</option>
      <option display="Green">Green</option>
    </parameter>
  </parameters>
  <css />
  <items>
    <item name="Start">[=Roll(1,6)] [Color] [Table] table(s)</item>
    <item name="AnyColor" weight="50">Blue</item>
    <item name="AnyColor" weight="25">Red</item>
    <item name="AnyColor" weight="10">Green</item>
    <item name="Blue">Blue</item>
    <item name="Red">Red</item>
    <item name="Green">Green</item>
    <item name="Table">Poker</item>
    <item name="Table">Dining</item>
    <item name="Table">Folding</item>
    <item name="Table">Picnic</item>
    <item name="Table">End</item>
  </items>
  <imports />
</generator>
```


Content

Dice generators are designed to roll dice. In particular, this type of generator is handy for more complicated dice rolling such as rolling stats for a character.

Function

Each function acts as a possible set of dice rolls. If multiple functions are defined, they must each have a unique name because they are shown to the user as a parameter. A function consists of a series of expressions.

Property	Description	Values
name	The name of the function	

displayName	The name to show the user	If this is not included, name will show instead
{Expression}	The expressions to run for this function	

Expressions

Expressions are the driving force behind Dice generators. The functionality of these expressions is almost entirely defined in the Calculations section of this help system. The one important difference is Variable Assignment.

Variable Assignment

To assign the result of a roll to a variable, as opposed to having it appear in the results, use the following structure:

```
Variable := Expression
```

This will create a variable named "Variable" that stores the results of the Expression and can be used in subsequent expressions.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<generator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xsi:type="Dice" version="2">
  <name>D&amp;D Attributes</name>
  <author>Lance Boudreaux</author>
  <description>Rolls the character attribute stats for D&amp;D.</description>
  <outputFormat>Html</outputFormat>
  <tags>
    <tag>Dice Roll</tag>
    <tag>D&amp;D</tag>
    <tag>Fantasy</tag>
  </tags>
  <tagList>Dice Roll, D&amp;D, Fantasy</tagList>
  <supportsMaxLength>false</supportsMaxLength>
  <function name="standard" displayName="3d6 in Order Old School">'STR ' +
Roll(3,6)
 'INT ' + Roll(3,6)
 'WIS ' + Roll(3,6)
 'DEX ' + Roll(3,6)
 'CON ' + Roll(3,6)
 'CHA ' + Roll(3,6)</function>
  <function name="standard" displayName="3d6 in Order Modern">'STR ' +
Roll(3,6)
 'DEX ' + Roll(3,6)
 'CON ' + Roll(3,6)
 'INT ' + Roll(3,6)
 'WIS ' + Roll(3,6)
 'CHA ' + Roll(3,6)</function>
  <function name="3d6" displayName="3d6">Roll(3,6)
 Roll(3,6)
 Roll(3,6)
 Roll(3,6)
```

```
 Roll(3,6)
 Roll(3,6)</function>
  <function name="4d6">Roll(4,6,'DL')
 Roll(4,6,'DL')
 Roll(4,6,'DL')
 Roll(4,6,'DL')
 Roll(4,6,'DL')
 Roll(4,6,'DL')</function>
  <function name="5d6">Roll(5,6,'DL',2)
 Roll(5,6,'DL',2)
 Roll(5,6,'DL',2)
 Roll(5,6,'DL',2)
 Roll(5,6,'DL',2)
 Roll(5,6,'DL',2)</function>
</generator>
```


List Generators

Content

The list generator is by far the simplest of all generators. It simply picks a single item from a list with all items having the same weight.

There are two elements to List Generators:

KeepWhiteSpace

When this is true, all whitespace at the beginning and end of the selected item is kept, otherwise it is trimmed.

Items

This is a list of items to choose from. Each item must be placed on a separate line. To include multiple lines in a single item, add the HTML break tag (
) where you want the line break to be.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<generator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xsi:type="List" version="2">
  <name>The Sorting Hat</name>
  <author>Lance Boudreaux</author>
  <description>Randomly chooses a Harry Potter House</description>
  <tags>
    <tag>Fantasy</tag>
  </tags>
  <keepWhiteSpace>False</keepWhiteSpace>
  <items>
  Gryffindor
  Hufflepuff
  Ravenclaw
```

```
  Slytherin
  </items>
</generator>
```


LUA Generators

Content

Lua generators use the powerful Lua Programming Language to drive their content creation. There are several custom methods defined in and usable by the generator engine via Lua calls.

Notes:
Imports are not supported by the Lua generator.
This generator type is not available in the Web Application.

Script

The Lua generator must contain one and only one Script element. Contained in this element is the actual Lua Script to execute. Any output from the script will be written to the results of the generator execution.

Parameters

The values of parameters are included in the Lua engine as global level variables and can be accessed as such.

Custom Methods

print(value)

Prints the contents of the parameter to the results.

printFormat(format, values[])

Prints the values provided using the format string. Internally, this method uses the .net String.Format() method.

printIf(condition, value)

Prints the value supplied provided that the condition is true.

printLine([value])

Prints the contents of the parameter to the results and inserts a new line. If value is absent, it simply prints a blank line.

printLineFormat(format, values[])

Prints the values provided using the format string and inserts a new line. Internally, this method uses the .net String.Format() method.

printLineIf(condition, value)

Prints the value supplied and inserts a new line provided that the condition is true.

createTable()

Creates an empty Lua table. Equivalent to "return {}".

calc(expression)

This runs the expression through the calculation engined defined in the Calculations section of this help

system.

selectFromTable(table)

Selects a single value from the table. The indexes of the table must be numeric and the space between indexes determines weight. For example, a table like this: {[1] = "Hello", [3] = "World"} will have a 25% chance of selecting "Hello".

Example

```
<?xml version="1.0" encoding="utf-8"?>
<generator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xsi:type="Lua" version="2">
  <name>Lua Sample Generator</name>
  <author>Lance Boudreaux</author>
  <description>Demonstrates the Lua generator type</description>
  <tags>
    <tag>Test</tag>
  </tags>
  <parameters>
 <parameter name="name" display="Name" type="Text" />
  </parameters>
  <script><![CDATA[

  local greeting =
selectFromTable({"Hello","Bonjour","Aloha","Hola","S'up","Aang","Ciao"})
  printFormat("{0} {1}", greeting, name)

  ]]></script>
</generator>
```


Phonotactic Generators

Content

The Phonotactics generator is based around the concept of phonotactics which defines the available combinations of phonemes a language has.

TextCase

This value determines how the casing of the content results will be handled. There are five options listed below. For the examples, the input string is "lance Edward boudreaux".

1. **None:** leave the case of the results as is. (ex: lance Edward boudreaux)
2. **Lower**: convert all characters to lower case. (ex. lance edward boudreaux)
3. **Upper:** convert all characters to upper case. (ex. LANCE EDWARD BOUDREAUX)
4. **Title:** convert the results to title case. (ex. Lance Edward Boudreaux)
5. **Proper:** convert the results to proper case. (ex. Lance edward boudreaux)

Definitions

Definitions are used to define what values a specific key stands for. The generator will choose a random item from the definition for each key it finds in a pattern. Currently only single character keys are supported.

Property	Description	Values
key	The key to denote this definition in a pattern	
delimiter	What character is used to separate each value	Defaults to comma (",")
{value}	A list of values separated by the delimiter	

Example that defines English consonants and vowels:

```
<definitions>
    <item key="C">B,C,D,F,G,H,J,K,L,M,N,P,Q,R,S,T,V,W,X,Y,Z</item>
    <item key="V">A,E,I,O,U</item>
</definitions>
```

Patterns

Patterns define the possible combinations of the definitions that are available. Any characters inside of the pattern not defined in the definitions is considered a literal. Placing a key inside of parentheses denotes that it is an optional key.

Property	Description	Values
weight	The relative probability of selecting this pattern	
{value}	The pattern	

Example that would generate words like "Hello" and "Cello":

```
<patterns>
    <item>CVCCV</item>
</patterns>
```


Example

```
<?xml version="1.0" encoding="utf-8"?>
<generator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xsi:type="Phonotactics"
version="2">
  <name>Phonotactics Sample</name>
  <author>Lance Boudreaux</author>
  <description>A sample Phonotactics Generator</description>
  <tags>
    <tag>Test</tag>
  </tags>
```

```
  <supportsMaxLength>false</supportsMaxLength>
  <definitions>
 <item key="C">B,C,D,F,G,H,J,K,L,M,N,P,Q,R,S,T,V,W,X,Y,Z</item>
 <item key="V">A,E,I,O,U</item>
  </definitions>
  <patterns>
 <item>CVC</item>
 <item>CVCCV</item>
  </patterns>
</generator>
```


Content

Table generators were designed in an effort to mimic random tables often found in role-playing game source books. They consist of a series of tables that are populated with a number of options that determine how content is derived from them.

Table

A table is a single content generation definition. The body of the Table is a table defined using delimited values. The default delimiter is a vertical bar. There are three types of tables detailed below.

Random Table

Selects a single row from the table by rolling a die and comparing it to the values in a defined column.

Property	Description	Values	Default Value	Required
name	The name of the table. This value is used when referencing the results from the table.			✓
repeat	The number of times to use this table.			
repeatJoin	A string used to separate values generated when this table is repeated			
delimiter	The character used to delimit columns in the table		|	
skipTable	A calculation that will cause the table to be skipped if it is true			
column	The column that contains the die roll values			✓
modifier	A modifier to the die roll. This can be a constant but would usually be a calculation			

Example:

```
<randomTable name="FortuneCookie" column="Roll"><![CDATA[
     Roll | Fortune
     02   | The man or woman you desire feels the same about you.
     03   | Meeting adversity well is the source of your strength.
```

```
      04   | A dream you have will come true.
      05   | Our deeds determine us, as much as we determine our deeds.
      06   | Never give up. You're not a failure if you don't give up.
      07   | You will become great if you believe in yourself.
      08   | There is no greater pleasure than seeing your loved ones prosper.
      09   | You will marry your lover.
      10   | A very attractive person has a message for you.
]]></randomTable>
```

Sample Results:

```
FortuneCookie = A dream you have will come true.
```

Loop Table

Loops through a table, generating content for each row that is accessible via a unique key value for that row.

Property	Description	Values	Default Value	Required
name	The name of the table. This value is used when referencing the results from the table.			✓
repeat	The number of times to use this table.			
repeatJoin	A string used to separate values generated when this table is repeated			
delimiter	The character used to delimit columns in the table		|	
skipTable	A calculation that will cause the table to be skipped if it is true			
column	The column used as the key for each row			✓

Example:

```
<loopTable column="Class" name="Highest-level locals"><![CDATA[
 Class      |Level
 Barbarian  |=Roll(1,4
 Bard       |=Roll(1,6)
 Cleric     |=Roll(1,6)
 Druid      |=Roll(1,6)
 Fighter    |=Roll(1,8)
]]></loopTable>
```

Sample Results:

```
Class.Barbarian = 2
Class.Bard = 5
Class.Cleric = 6
Class.Druid = 1
Class.Fighter = 7
```

Select Table

Selects a row from the table based on a value in one of the columns.

Property	Description	Values	Default Value	Required
name	The name of the table. This value is used when referencing the results from the table.			✓
repeat	The number of times to use this table.			
repeatJoin	A string used to separate values generated when this table is repeated			
delimiter	The character used to delimit columns in the table		|	
skipTable	A calculation that will cause the table to be skipped if it is true			
column	The column used as the key to search for			✓
selectValue	The value to look for in the column			

Example:

```
<selectTable column="Count" name="Barbarians"
selectValue="[=Class.Barbarian]"><![CDATA[
      Count | Value
      0     | This place is civilized
      1     | That guy is crazy
      2     | Are they from around here?
      3     | This place is rough
]]></selectTable>
```

Sample Results:

```
Barbarians.Value = "Are they from around here?"
```

Output

The output section of a table generator determines how the results of the tables are presented.

Example:

```
<output>
  <![CDATA[
 <h3 style="border-bottom:1px solid black">Town Info</h3>
 <b>Town Size:</b>             [Town Size.Town Size] <br />
 <b>Population:</b>            [Town Size.Population] <br />
 <b>GP Limit: </b>             [Town Size.GP Limit] <br />
 <b>Power Center: </b>         [Power Center.Power Center]<br />
 <b>Power Center Alignment:</b>[Power Center Alignment.Alignment]<br />
```

```
<b>Community Authorities:</b> [Community Authorities.Rank]<br />
<h3 style="border-bottom:1px solid black">Highest-level locals:</h3>
<b>Adept:</b>                 [Highest-level locals.Adept.Level]<br />
<b>Aristocrat:</b>            [Highest-level locals.Aristocrat.Level]<br />
<b>Barbarian:</b>             [Highest-level locals.Barbarian.Level]<br />
]]></output>
```

Sample Output:

Town Info

Town Size: Village
Population: 543
GP Limit: 200 gp
Power Center: **Conventional**
Power Center Alignment: Lawful Good
Community Authorities: Second highest level fighter

Highest-level locals:

Adept: 3
Aristocrat: 2
Barbarian: 1

Example

```
<?xml version="1.0" encoding="utf-8"?>
<generator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xsi:type="Table" version="2">
  <name>Town Generator (D&amp;D 3e)</name>
  <author>Lance Boudreaux</author>
  <description>A town generator based on the D&amp;D 3e DMG.</description>
  <tags>
    <tag>Environment</tag>
    <tag>D&amp;D</tag>
    <tag>Fantasy</tag>
  </tags>
  <tables>
    <randomTable column="Roll" name="Town Size" delimiter="|">
  Roll |Town Size  |Population          |GP Limit |PCA|CM
  10   |Thorp      |=Rnd(20,80)         |40 gp    |-1 |-3
  30   |Hamlet     |=Rnd(81,400)        |100 gp   |0  |-2
  50   |Village    |=Rnd(401,900)       |200 gp   |+1 |-1
  70   |Small Town |=Rnd(901,2000)      |800 gp   |+2 |0
  85   |Large Town |=Rnd(2001,5000)     |3000 gp  |+3 |+3
  95   |Small City |=Rnd(5001,12000)    |15000 gp |+4 |+6
  99   |Large City |=Rnd(12001,25000)   |40000 gp |+5 |+9
  100  |Metropolis |=Rnd(25001,100000) |100000 gp|+6 |+12
  </randomTable>
    <randomTable column="Roll" name="Power Center" delimiter="|"
randomModifier="=[Town Size.PCA]">
  Roll |Power Center
  13   |Conventional
  18   |Nonstandard
  20   |Magical
  </randomTable>
    <randomTable column="Roll" name="Power Center Alignment" delimiter="|">
  Roll |Alignment
  35   |Lawful Good
```

```
  39   |Neutral Good
  41   |Chaotic Good
  61   |Lawful Neutral
  63   |True Neutral
  64   |Chaotic Neutral
  90   |Lawful Evil
  98   |Neutral Evil
  100  |Chaotic Evil
  </randomTable>
    <randomTable column="Roll" name="Community Authorities" delimiter="|">
  Roll |Rank
  60   |Highest level warrior
  80   |Second highest level fighter
  100  |Highest level fighter
  </randomTable>
    <loopTable column="Class" name="Highest-level locals" delimiter="|">
  Class      |Level
  Barbarian  |=ToOrdinal(Max(Roll(1,4)+[Town Size.CM],0))
  Bard       |=ToOrdinal(Max(Roll(1,6)+[Town Size.CM],0))
  Cleric     |=ToOrdinal(Max(Roll(1,6)+[Town Size.CM],0))
  Druid      |=ToOrdinal(Max(Roll(1,6)+[Town Size.CM],0))
  Fighter    |=ToOrdinal(Max(Roll(1,8)+[Town Size.CM],0))
  Monk       |=ToOrdinal(Max(Roll(1,4)+[Town Size.CM],0))
  Paladin    |=ToOrdinal(Max(Roll(1,3)+[Town Size.CM],0))
  Ranger     |=ToOrdinal(Max(Roll(1,3)+[Town Size.CM],0))
  Rogue      |=ToOrdinal(Max(Roll(1,8)+[Town Size.CM],0))
  Sorceror   |=ToOrdinal(Max(Roll(1,4)+[Town Size.CM],0))
  Wizard     |=ToOrdinal(Max(Roll(1,4)+[Town Size.CM],0))
  Adept      |=ToOrdinal(Max(Roll(1,6)+[Town Size.CM],0))
  Aristocrat |=ToOrdinal(Max(Roll(1,4)+[Town Size.CM],0))
  Commoner   |=ToOrdinal(Max(Roll(4,4)+[Town Size.CM],0))
  Expert     |=ToOrdinal(Max(Roll(3,4)+[Town Size.CM],0))
  Warrior    |=ToOrdinal(Max(Roll(2,4)+[Town Size.CM],0))
  </loopTable>
  </tables>
  <output>
 &lt;h3 style="border-bottom:1px solid black"&gt;Town Info&lt;/h3&gt;
 &lt;b&gt;Town Size:&lt;/b&gt;             [Town Size.Town Size] &lt;br /&gt;
 &lt;b&gt;Population:&lt;/b&gt;            [Town Size.Population] &lt;br /&gt;
 &lt;b&gt;GP Limit: &lt;/b&gt;             [Town Size.GP Limit] &lt;br /&gt;
 &lt;b&gt;Power Center: &lt;/b&gt;         [Power Center.Power Center]
&lt;br /&gt;
 &lt;b&gt;Power Center Alignment:&lt;/b&gt;[Power Center Alignment.Alignment]
&lt;br /&gt;
 &lt;b&gt;Community Authorities:&lt;/b&gt; [Community Authorities.Rank]
&lt;br /&gt;
 &lt;h3 style="border-bottom:1px solid black"&gt;Highest-level
locals:&lt;/h3&gt;
 &lt;b&gt;Adept:&lt;/b&gt;                 [Highest-level locals.Adept.Level]
&lt;br /&gt;
 &lt;b&gt;Aristocrat:&lt;/b&gt;            [Highest-level
locals.Aristocrat.Level]&lt;br /&gt;
 &lt;b&gt;Barbarian:&lt;/b&gt;             [Highest-level
locals.Barbarian.Level]&lt;br /&gt;
 &lt;b&gt;Bard:&lt;/b&gt;                 [Highest-level locals.Bard.Level]
&lt;br /&gt;
 &lt;b&gt;Cleric:&lt;/b&gt;                [Highest-level locals.Cleric.Level]
&lt;br /&gt;
 &lt;b&gt;Commoner:&lt;/b&gt;              [Highest-level
locals.Commoner.Level]&lt;br /&gt;
```

```
 &lt;b&gt;Druid:&lt;/b&gt;                [Highest-level locals.Druid.Level]
&lt;br /&gt;
 &lt;b&gt;Expert:&lt;/b&gt;               [Highest-level locals.Expert.Level]
&lt;br /&gt;
 &lt;b&gt;Fighter:&lt;/b&gt;              [Highest-level
locals.Fighter.Level]&lt;br /&gt;
 &lt;b&gt;Monk:&lt;/b&gt;                 [Highest-level locals.Monk.Level]
&lt;br /&gt;
 &lt;b&gt;Paladin:&lt;/b&gt;              [Highest-level
locals.Paladin.Level]&lt;br /&gt;
 &lt;b&gt;Ranger:&lt;/b&gt;              [Highest-level locals.Ranger.Level]
&lt;br /&gt;
 &lt;b&gt;Rogue:&lt;/b&gt;               [Highest-level locals.Rogue.Level]
&lt;br /&gt;
 &lt;b&gt;Sorceror:&lt;/b&gt;            [Highest-level
locals.Sorceror.Level]&lt;br /&gt;
 &lt;b&gt;Warrior:&lt;/b&gt;             [Highest-level
locals.Warrior.Level]&lt;br /&gt;
 &lt;b&gt;Wizard:&lt;/b&gt;              [Highest-level locals.Wizard.Level]
&lt;br /&gt;
 </output>
</generator>
```


Calculations

One of the most important features of the generators is the ability to run calculations. This functionality is provided by the NCalc calculation engine which is extended in the randomizer generators with the following:

Parameters

Parameters defined for the generator are available in the calculations and can be referenced by their parameter name.

Functions

In addition to the standard functionality offered by NCalc, the following functions are available

Function	Description	Example
CBool (value)	Converts the provided *value* to a boolean.	`CBool(1) Converts 1 to True`
CDbl (value)	Converts the provided *value* to a double.	`CDbl("0.1") Converts "0.1" to 0.1`
CInt (value)	Converts the provided *value* to a 32-bit integer.	`CInt("22") Converts "22" to 22`
CLng (value)	Converts the provided *value* to a 64-bit integer.	`CLng("1218189") Converts "1218189" to 1218189`
Rnd ()	Generates a random non-negative integer.	`Rnd() Generates a non-negative random integer`
Rnd (max)	Generates a random non-negative integer that is	`Rnd(42) Generates a number`

	less than or equal to *max*.	`between 0 and 42 inclusive`
Rnd (min, max)	Generates a random integer greater than *min* and less than or equal to *max*.	`Rnd(1,6) Generates a number between 1 and 6 inclusive`
Roll (sides)	Rolls one die with the number of *sides* provided.	`Roll(20) Rolls 1d20`
Roll (count, sides)	Rolls *count* dice with the number of *sides* provided.	`Roll(3,6) Rolls 3d6`
Roll (count, sides, modifier)	Rolls *count* dice with the number of *sides* provided and alters the result by the amount of the modifier.	`Roll(3,6,-1) Rolls 3d6-1`
Roll (count, sides, options[])	Rolls *count* dice with the number of *sides* provided using the *options* provided. Options are detailed below.	`Roll(4,6,"DL") Rolls 4d6 and drops the lowest`
Roll (count, sides, modifier, options[])	Rolls *count* dice with the number of *sides* provided using the *options* provided and alters the result by the amount of the modifier.	`Roll(2,20,+3,"DL") Rolls 2d20, drops the lowest, and adds 3`
LastRoll ()	Returns the result of the last roll.	`LastRoll() Returns the result of the last roll`
LastRoll (detail)	Returns the *detail* of the last roll. Detail is a string that can be: Result, ResultList, Successes, Failures, or Botches.	`LastRoll("ResultList") Returns all die rolls from the last roll separated by commas`
Pick (values[])	Selects one item from *values* at random.	`Pick("Red","Green","Blue") Picks either "Red","Green", or "Blue"`
UCase (value)	Converts *value* to uppercase	`UCase("dog") Returns "DOG"`
LCase (value)	Converts *value* to lowercase	`LCase("DOG") Returns "dog"`
TCase (value)	Converts *value* to titlecase	`TCase("big RED dog") Returns "Big Red Dog"`
ToOrdinal (number)	Returns the ordinal value of the provided *number.*	`ToOrdinal(34) Returns "34th"`
ToText (number)	Returns the word for the provided *number.*	`ToText(123) Returns "One Hundred Twenty Three"`
Format (string, values[])	Inserts the *values* into the *string* based on their index.	`Format("The {0} is {1}", "Rose", "Red") Returns "The Rose is Red"`

Generate (string)	Calls a generator with the name provided by *string*.	`Generate("ElfName") Calls a generator called "ElfName"`
Generate(string, <string, value>[])	Calls a generator with the name provided by *string* and provides the list of parameters in the array provided. MaxLength can be included as a parameter in this list.	`Generate("ElfName","MaxLength",20,"Gender","Male") Calls the generator called "ElfName" and passes in a Max Length of 20 and a parameter called "Gender" with the value "Male"`