# The Nudge Support Toolset

Michael A. Gohde

December 8, 2016

## 1 Overview

The task of writing stories for Nudge is inherently difficult. The Nudge Support Toolset exists to alleviate some of this difficulty by providing a large set of utilities designed to compile, validate, and install stories.

This document exists to detail the function and operation of each utility in the Nudge Support Toolset.

## 2 story2xml.py

story2xml exists to translate well structured, human-readable stories into an intermediate XML format for use by all of the other tools in the set.

### 2.1 Input file format

```
1 Title: <Insert story title here>
2
3 First story node title:
4    <Insert story node text here>
5
6    Responses:
7        Response 1 text -> prb1% to dst1, prbN% to dstN
8        Response N text -> prb3% to dst3
```

Each story2xml input file must follow a common set of conventions:

1. The story's title must be written on its own line and prefixed with "Title:"

2. Each story "block" or "decision" (hereafter referred to as a "node") must start with a title followed by a colon.

3. The text content in each node (ie. the story text) must not start with a word followed by a colon.

4. All possible user actions are defined in a block prefixed by the keyword "Responses:". This block must be indented more than all of the other text in its node.

5. Every story must be terminated with a reference to a node named "END". This node should not be defined, but exists as a special destination.

## 2.2 A complete input file example

```
1  Title: Example story
2
3  D1_0:
4      You are confronted with a serious question:
5      To cheese it or not to cheese it?
6
7      Responses:
8          Cheese it! -> 50% to D2_0, 50% to D2_1
9          Don't cheese it! -> 100% to D2_2
10
11 D2_0:
12     Note: This is a comment.
13     Note: All comment text is discarded.
14
15     You were able to cheese it!
16
17     Responses:
18         Proceed -> 100% to D3_0
19
20 D2_1:
```

```
21      Comment: This is also a comment.
22      You were unsuccessful at cheesing it!
23
24      Responses:
25          Proceed -> 100% to D3_0
26
27  D2_2:
28      You proceed not to cheese it.
29
30      Responses:
31          Proceed -> 100% to D3_0
32
33  D3_0:
34      Regardless of whether you cheesed it,
35      something happened.
36
37      Responses:
38          End -> 100% to END
```

## 2.3 Running story2xml

story2xml accepts one argument on the command line: the name of a text file containing a properly formatted story. Once run, story2xml will print an XML-formatted story to stdout, so it may be useful to redirect its output to a different file.

Example usage:

story2xml.py mystory.txt >mystory.xml

## 2.4 Error messages

When compiling a story, story2xml attempts to perform a few tests to ensure that the provided source file is logically sound. If an error is encountered, story2xml will print a message and information that can be used to locate and correct the error.

List of error messages:

**Error: for response (response text) on line (line number), destination probabilities exceed 100%** This error indicates that the total of all of the weights for some response exceeds 100%. To correct this error, go to the line indicated and re-evaluate all weights.

**Error: for response (response text) on line (line number), destination probabilities sum to a value below 100%** This error indicates that the total of all o fthe weights for some response is below 100%. To correct this error, go to the line indicated and re-evaluate all weights.

**Warning: for line (line number), unknown token (string)** This message indiciates that the compiler has found a command (a word followed by a colon) that it has no rule to handle. This can safely be ignored in some cases.

# 3 sanitytext.py

sanitytest exists to check whether a story is logically complete by a number of metrics. These include whether a story can be finished through every possible combination of user actions, whether every set of weights sums to 100%, and whether there are any circular references within a story.

## 3.1 Running sanitytest

Like story2xml, sanitytest accepts one argument on the command line: the nane of a text file containing an XML-formatted story file. When run, sanitytest will either produce an error output or a message indicating that the story passed all checks.

Example usage:

sanitytest.py mystory.xml

## 3.2 Example output

```
1 user@computer-$ ./sanitytest.py ../examples/err.story.
    xml
```

```
2 Circular reference check failed for node D1_0 referred
    to by D2_1
```

## 3.3 Error messages

When performing checks on a story, sanitytest my produce one of several error messages. Each error message specific directions as to the nature of the error and where it is in the story.

**Error: No node should be able to have itself as a destination** When encountered, this error indicates that a node has itself listed as a possible destination. This is problematic for two reasons:

1. Because it recursively descends through all story nodes to check for completion, sanitytest would crash if a node referenced itself.
2. Users would likely tire of the theoretically infinite repetition of the same story text and destinations.

**Error: All destinations should have a probability greater than 0** This error should never be encountered if a story is first compiled by story2xml. It indicates that some destination probability is less than or equal to 0. Since this is likely unintended (and can break Nudge's story display code) it is an error that will result in termination of the program.

**Error: Probability for all destinations does not sum to 100%** This error, like the above, should never be encountered if a story is first compiled by story2xml. It indicates that all destination weights in a node sum to a value not equal to 100%.

**Story failed sanity check on node (node name) with destination (destination node name)** This error indicates that the story cannot be completed through every possible set of choices.

**Circular reference check failed for node (node name) referred to by (node name)** A circular reference occurs when a story node is referred to by one of its children. This is considered problematic because a user could get stuck in an endless series of repetitive story elements.

# 4 storylist.py

storylist provides story developers with the ability to see every possible decision set in their story evaluated and presented in a human-readable format.

## 4.1 Running storylist

When run, storylist:

1. Iterates through every possible set of decisions that a user can make in a story
2. Determines how likely the given story is based on all node weights
3. Prints out every possible story.

**Warning** When run, storylist assumes that a story has been sanity checked.

Example usage:

storylist.py mystory.xml

## 4.2 Example output

```
1  user@computer./storylist.py example.story.xml
2  Storyline weight probability: 50.0%
3          Node ID: D1_0
4          Text: You are confronted with a serious
               question: To cheese it or not to cheese it?
5          Answer chosen: Cheese it!
6          Destination: D2_0
7          Destination weight: 50.0%
8
9          Node ID: D2_0
10         Text: You were able to cheese it!
11         Answer chosen: Proceed
12         Destination: D3_0
13         Destination weight: 100.0%
14
15         Node ID: D3_0
```

```
16          Text: Regardless of whether you cheesed it,
                something happened.
17          Destination: END
18
19  Storyline weight probability: 50.0%
20          Node ID: D1_0
21          Text: You are confronted with a serious
                question: To cheese it or not to cheese it?
22          Answer chosen: <unknown>
23          Destination: D2_1
24          Destination weight: 50.0%
25
26          Node ID: D2_1
27          Text: You were unsuccessful at cheesing it!
28          Answer chosen: Proceed
29          Destination: D3_0
30          Destination weight: 100.0%
31
32          Node ID: D3_0
33          Text: Regardless of whether you cheesed it,
                something happened.
34          Destination: END
35
36  Storyline weight probability: 100.0%
37          Node ID: D1_0
38          Text: You are confronted with a serious
                question: To cheese it or not to cheese it?
39          Answer chosen: Don't cheese it!
40          Destination: D2_2
41          Destination weight: 100.0%
42
43          Node ID: D2_2
44          Text: You proceed not to cheese it. This is an
                additional line to test the parser. Yet
                another line!
45          Answer chosen: Proceed
46          Destination: D3_0
47          Destination weight: 100.0%
```

```
48
49          Node ID: D3_0
50          Text: Regardless of whether you cheesed it,
                something happened.
51          Destination: END
```

# 5 storygraph.py

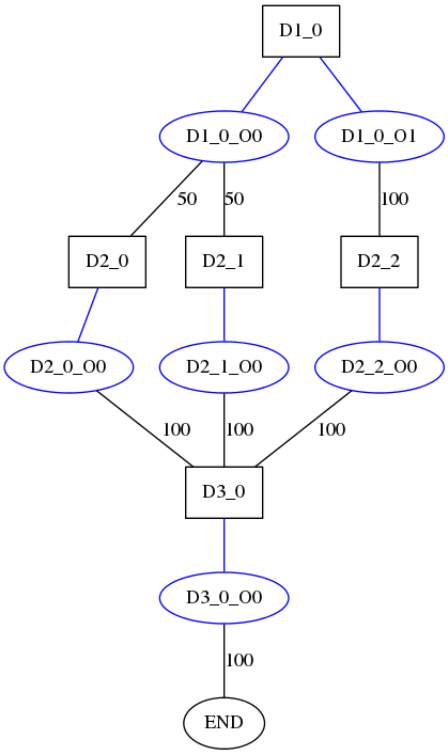
When writing a story, it is often helpful to plot out how all nodes and decisions connect to each other. storygraph attempts to automate this process by translating a story in XML format to a specialized format used by a plot generation tool package named *graphviz*.

## 5.1 Running storygraph

Since storygraph produces data in an intermediate format, it is often useful to simply pipe its output into graphviz to obtain a complete image file. The following example will demonstrate this.

storygraph.py mystory.xml — dot -Tpng >mystory.png

### 5.2 Example output




## 6 dbtool.py

Once a story has been completed and sanity tested, it is necessary to convert its contents into a series of SQL statements for insertion into the Nudge story database. dbtool is used to perform this conversion.

### 6.1 Running dbtool

dbtool produces a series of SQL statements. As such, it is likely useful to redirect this output to a file that can be sourced later on.

dbtool.py mystory.xml >mystory.sql

### 6.2 Example output

```
1  -- Generated statements for node: D1_0
2  INSERT INTO storytable VALUES (1,'Example story','D1_0
       ','You are confronted with a serious question: To
       cheese it or not to cheese it?',0);
3  INSERT INTO answers VALUES ('Example story','D1_0','A
       ','Cheese it!');
4  INSERT INTO results VALUES (1,'Example story','D1_0','A
       ',0,50,'D2_0');
5  INSERT INTO results VALUES (2,'Example story','D1_0','A
       ',50,100,'D2_1');
6  INSERT INTO answers VALUES ('Example story','D1_0','B
       ','Don\'t cheese it!');
7  INSERT INTO results VALUES (3,'Example story','D1_0','B
       ',0,100,'D2_2');
8  -- Generated statements for node: D2_0
9  INSERT INTO storytable VALUES (2,'Example story','D2_0
       ','You were able to cheese it!',1);
10 INSERT INTO answers VALUES ('Example story','D2_0','A
       ','Proceed');
11 INSERT INTO results VALUES (4,'Example story','D2_0','A
       ',0,100,'D3_0');
12 -- Generated statements for node: D2_1
13 INSERT INTO storytable VALUES (3,'Example story','D2_1
       ','You were unsuccessful at cheesing it!',2);
14 INSERT INTO answers VALUES ('Example story','D2_1','A
       ','Proceed');
15 INSERT INTO results VALUES (5,'Example story','D2_1','A
       ',0,100,'D3_0');
16 -- Generated statements for node: D2_2
17 INSERT INTO storytable VALUES (4,'Example story','D2_2
       ','You proceed not to cheese it. This is an
       additional line to test the parser. Yet another line
       !',3);
18 INSERT INTO answers VALUES ('Example story','D2_2','A
       ','Proceed');
19 INSERT INTO results VALUES (6,'Example story','D2_2','A
       ',0,100,'D3_0');
```

```
20 -- Generated statements for node: D3_0
21 INSERT INTO storytable VALUES (5,'Example story','D3_0
       ','Regardless of whether you cheesed it, something
       happened.',4);
22 INSERT INTO answers VALUES ('Example story','D3_0','A
       ','End');
23 INSERT INTO results VALUES (7,'Example story','D3_0','A
       ',0,100,'END');
```