Getting Started with the Nudge Support Toolset

Michael A. Gohde

December 13, 2016

1 Overview

This document exists to provide a series of tutorials on how to format, test, and install Nudge storylines.

The tutorials presented here assume that the reader has an existing knowledge of a *NIX command line and a more or less complete storyline.

2 Formatting the story

All of the tools used in this tutorial are designed to work on a special story definition format. Due to the time consuming nature of rewriting each story by hand for the toolset, a special translation tool was developed. This tool can convert a somewhat strictly formatted plain text document into the intermediate format used by all of the other tools in the set.

2.1 Example formatted story

The example below presents a story in the correct format for use with the conversion tool. Please note that, due to some limitations in its design, the tool will automatically discard line breaks and indentation.

```
Title: Example story

D1_0:
    You are confronted with a serious question:
```

```
    To cheese it or not to cheese it?

    Responses:
        Cheese it! -> 50% to D2_0, 50% to D2_1
        Don't cheese it! -> 100% to D2_2

D2_0:
    Note: This is a comment.
    Note: All comment text is discarded.

    You were able to cheese it!

    Responses:
        Proceed -> 100% to D3_0

D2_1:
    Comment: This is also a comment.
    You were unsuccessful at cheesing it!

    Responses:
        Proceed -> 100% to D3_0

D2_2:
    You proceed not to cheese it.

    Responses:
        Proceed -> 100% to D3_0

D3_0:
    Regardless of whether you cheesed it,
    something happened.

    Responses:
        End -> 100% to END
```

From the above, a few simple formatting rules can be inferred:

1. Story nodes[1] start with an unindented node name (this name is used for all response destinations) followed by indented[2] node contents.

2. Notes can be added to each story node by starting a line with "Comment:" or "Note:". These notes will be discarded by the translation tool during translation.

3. Story text can be more or less free form, as long as each story text line is on the same level of indentation as all of the other lines or blocks in the node.

4. All possible user responses are found within a block starting with the text "Responses:"

5. A user response contains both a prompt (the first bit of text to the left of the "->") and a comma-separated set of destinations and their weights.

2.2 Translating the story

Once the story has been formatted correctly, it is necessary to run the translation tool. From a terminal, run the following command in the story tools directory with appropriate substitutions:

```
./story2xml.py yourstoryname.txt >outputfile.xml
```

You can optionally omit the ">outputfile.xml" to print the translated story out on the terminal.

2.3 What to do if something goes wrong

The story translation tool is capable of detecting and reporting some rudimentary problems with a story. If an error message is displayed, it will come with information about where, exactly, the problem was found. For reference on each particular error message, please refer to "story2xml.py" in "The Nudge Support Toolset"

[1] A "story node" refers to a story decision point consisting of a block of descriptive text and a set of destinations with prompts.

[2] Each indented segment is referred to as a "block" in other documentation. Each successive level of indentation creates a new block under some heading

3 Testing the story

Once a story has been translated, it is possible to perform in-depth testing and analysis. The tests detailed in this section[3] should be able to determine if a story is complete and logically correct.

3.1 Running the story test tool

Running a full set of tests on each story is fairly straightforward. Simply run the following with the correct filename:

```
./sanitytest.py outputfile.xml
```

In the event that an error message is generated, please see "Running sanitytest" in "The Nudge Support Toolset". Once the error has been corrected, repeat the process of translating the story and running the story test tool.

4 Installing the story

After translating and verifying a story, it is necessary to install it in Nudge's database. The toolset contains a tool that can translate the intermediate format into a series of SQL statements.

4.1 Generating SQL statements

To generate a series of SQL statements from an intermediate file, run the following:

```
./dbtool outputfile.xml >sqloutput.sql
```

Once this process is complete, see your SQL server documentation for instructions on how to install the resulting set of SQL statements.

[3]More specifically, the process detailed here will: 1. check if the story can be completed through every possible set of user choices and random events, 2. determine if all weights on user responses sum to 100%, and 3. determine if it is possible for a user to get caught in an infinitely repeating set of story choices.

5 Other useful tools

The Nudge Support Toolset features several additional tools that should help with the story writing and debugging process. This section will document the function and usage of each tool.

5.1 Generating story graphs

One of the most useful aids in developing a storyline can be visualization. The Nudge Support Toolset contains a tool that can assist with story graphing by translating a story into s series of commands usable by an external package named *graphviz*.

If graphviz is installed, it is possible to generate a story graph with the following command:

```
./storygraph.py outputfile.xml — dot -Tpng >outputfile.png
```

The above command translates a story into a graph command listing, then sends the translated output to a program that can generate an image file from that listing.

If graph command output is to be retained:

```
./storygraph.py outputfile.xml — tee output.dot — dot -Tpng
>outputfile.png
```

5.2 Generating a full story listing

Sometimes, it may be useful to generate a complete set of every possible storyline for every possible user action. This is especially helpful when determining if each possible user action results in a coherent story.

To generate a listing, run the following:

```
./storylist.py outputfile.xml
```

It may also be helpful to save the resulting story for later consultation. In order to do this, run the following:

```
./storylist.py outputfile.xml >listing.txt
```