Transduction Logging in Graphite

Sharon Correll

Last modified: 15 October 2007

1 Introduction

The current debugging facility in Graphite is quite primitive and not terribly user-friendly. In place of the ideal stepping debugger, it consists simply of a text file that logs the process the engine goes through as it transduces a text string for rendering. This file is generated when requested by the application that is functioning as the client for the Graphite engine.

Each Graphite renderer consists of a series of a passes, each of which fires rules which make changes to the stream of output. The final output consists of a list of glyphs and their positions, and mappings between the surface and underlying text that are used for hit testing and drawing selections.

The log file shows the original underlying input, indicates which rules were fired at each pass, and shows the resulting output. Also included are the mappings between underlying characters and surface glyphs, which are used to handle hit testing and rendering selections.

2 Obtaining the log

The output of the transduction log is controlled by the application that is using Graphite. In WorldPad, logging can be turned on by going to the Tools menu, choosing the Options item, and checking the box labeled “Output Graphite debug logs.”

The output is written to a file called “gr_xductn.log.” This is placed in the applications default directory. The file should be viewed with a fixed-width font.

3 Description of log file contents

3.1 Segment Output

Each successive segment of a session is appended to the log file. If the application does not send a previous segment to Graphite, it is assumed to be the beginning of a session and the log is reinitialized.

Divisions between segments are indicated by the following divider line:

```
==============================================
```

3.2 Underlying Input

The first table of information shows the underlying input. This includes the text forms of any ANSI characters for the rendered string and the corresponding Unicode codepoint in hex. Keep in mind that all Unicode codepoints and glyph IDs are logged in hex, not in decimal.

The header of the underlying input table shows the offsets of the characters relative to the start of the string, followed by the offsets relative to the start of the segment. Each segment has an "official" start and end, but may "borrow" and render characters from adjacent segments. Even if characters from a previous segment are not actually rendered in the current segment, they may have an effect on the current segment, and if so must be reprocessed. Such reprocessed slots are included in this table and have negative segment offsets.

The "Runs" row of the table marks the beginning of runs of text that are characterized by a set of styles and/or features. For instance, in the following table,

```
Text:          a      b      c      d      e      f
Unicode:       0061   0062   0063   0064   0065   0066
Runs:          |1                   |2            |3
```

there are three runs, the first consisting of 'abc', the second consisting of 'de', and the third 'f'.

Underneath of the "Runs" row, the feature values for each run are listed.

Note: in addition to size, color, or boldness, changes in underlining may be a reason for having separate runs, but underlining is not indicated in the log file.

If the text in the input stream is UTF-16, any supplementary-plane Unicode characters will be represented using 16-bit surrogates. The log file displays the surrogates together in a stack as shown below in the third column.

```
string         55     56     57     59     60
segment        0      1      2      4      5

Text:          a      b             d      e
Unicode:       0061   0062   DE83   0064   0065
                             DD99
```

Note that the indices given above the characters are the indices of the 16-bit characters.

3.3 Input to Pass 1

This table shows the glyph IDs that were generated for each character in the underlying input. If there is no linebreak table, it will also show any line-break "glyphs" that have been inserted in to the stream corresponding to the start and end of the segment. Line-break glyphs are indicated by "#."

3.4 Rules Matched

For each pass, there is a list of rules that were matched by the glyphs in the stream. Each item indicates which the current position of the input stream at the point that the rule was matched, the number of the rule, and whether the rule failed or fired. Rules fail due to tests in the constraints (-if- statements or conditionals in the rule's context) returning false.

The numbers of the rules match those in the debugger files output by the compiler—dbg_ruleprec, dbg_enginecode.txt, and dbg_fsm.txt. Refer to these files to see a (possibly abbreviated) text representation of the rule. (The order of the rules does correspond to the order in the original GDL file, but the engine may have several rules corresponding to a single rule in the source code, due to the presence of optional items.)

Keep in mind that the slot indices listed are those in the *input* stream, that is, the glyph stream listed previously in the log, which was generally the output from the previous pass.

Also note that the pass numbers do not correspond to the numbers used in the GDL file; rather all the passes are numbered sequentially, starting with the passes from the linebreak table, followed by the substitution passes, and finally the positioning passes.

Since the bidi pass never fires rules, but only runs the bidi algorithm, no rules are listed for the bidi pass.

3.5 Output of Pass

The next table shows the output of the pass in which the rules were fired. In other words, it is showing the changes that the rules made to the glyph stream.

Insertions and Deletions

First, above the table header, there are indications of inserted and deleted glyphs. If INS appears above a slot number, it means that that slot was inserted during the previous pass. DEL between two slot numbers means that a single slot was deleted, while DEL-n indicates a group of deleted slots.

Glyph IDs

A "#" in the glyph ID list indicates an inserted line-break "glyph". Again, remember that all glyph IDs are in hex.

Attributes Modified

Underneath the glyph IDs are an indication of which slot attributes changed during the pass. If a slot's entry in the row is empty, it can be assumed that its value is unchanged from the previous pass. Similarly, if a slot attribute is not listed at all, it can be assumed that no slot had that attribute modified during the pass.

Note that these modifications are with respect to the same or corresponding glyph, which might not be at the same index in the previous stream. Specifically if insertions or deletions have occurred, an adjustment needs to be made in locating the corresponding glyph in the previous stream. Consider for instance:

```
OUTPUT OF PASS 1 (substitution)

               0      1      2      3      4      5

Glyph IDs:     0010   0011   0012   0013   0014   0015
[insert        true   false  false  true   false  false]

PASS 2

Rules matched:
 * 2. rule FIRED [to insert 0x0099]
 * 4. rule FIRED [to delete 0x0014]

OUTPUT OF PASS 2 (substitution)

                             INS              DEL
               0      1      2      3      4      5

Glyph IDs:     0010   0011   0099   0012   0013   0015
insert                              true
```

Because there was an insertion at slot 2, slot 3 in the output—0x0012—corresponds to slot 2 in the input (the output of pass 1). Its previous insert value was false, and is now true, therefore the value is listed in the table.

Any time a slot is modified, its associations are listed in the table. If there are no more than two associations per slot, the before- and after-associations are listed on a single line; otherwise several lines are used. (The “before” association is the slot in the input corresponding to the leading edge of the glyph; the “after” association corresponds to the trailing edge of the glyph. There may be other intermediate associated slots.)

(Occasionally associations may be listed when apparently no other slot attribute changed, nor were the associations themselves changed. This is simply due to an infelicity in the way slots are copied and is nothing to be concerned about.)

Skipped Slots

When there is cross-line-boundary contextualization occuring, it is necessary for the renderer to reprocess some of the input from the previous segment, and then skip slots here and there in order to resync and avoid duplicated rendering. The reprocessed slots are initially indicated by negative segment offsets in the underlying input table.

Then, at the bottom of the pass output table, the word SKIP will appear under slots that were skipped after that pass. For instance, in the following example, two slots have been skipped after pass 1. Notice the correspondences between slot 2 in the first pass and slot 0 in the second pass, slot 3 in the first pass and slot 1 in the second pass, etc.

```
OUTPUT OF PASS 1 (substitution)

               0      1      2      3     4      5      6     7

Glyph IDs:     0033   0022   0011   #     0005  0006   0007   0008

               SKIP   SKIP

PASS 2

Rules matched:
 * 2. rule 0 FIRED

OUTPUT OF PASS 2 (substitution)

               0      1      2      3     4      5

Glyph IDs:     0011   #      0005  0006   0007   0008
```

3.6 Mappings between underlying characters and surface glyphs

Underlying to Surface Mappings

The underlying-to-surface mappings are used to render a selection appropriately based on the position of the selection in the underlying text. The “before” association indicates the surface glyph of interest when the selection is before the given character, and the “after” association indicates the surface glyph of interest when the selection is just after the given character (ie, at position *n+1*). There may be additional intermediate associations that are used to draw range selections.

Sometimes a character that is officially located within a given segment may not be rendered within that segment at all, but in a previous or following segment. Or it may be rendered in both segments. In the following example,

```
UNDERLYING TO SURFACE MAPPINGS

string         10     11     12     13     14     15
segment        0      1      2      3      4      5

Text:          a      b      c      d      e      f
Unicode:       0061   0062   0063   0064   0065   0066
before         <--    1      2      3      4      -->
after          0      1      2      3      4      -->
```

the first character in the segment, “a,” is rendered both in this segment (by glyph 0) and in the previous segment (as indicated by “`<--`” arrow). The final “f” is rendered only in the following segment (indicated by the two “`-->`” arrows).

If any of the characters are ligature components, this will be indicated in the table. The “ligature” row indicates which surface where the corresponding ligature is to be found, and the “component” row indicates which component the given slot is functioning as.

The table may include characters that are not officially part of this segment, but are “borrowed” from the previous or following segment and rendered in this one. (Characters with negative segment offsets are borrowed from the previous segment.) Because they are rendered in this segment, the mappings to the surface glyphs must be included in order to render selections.

Surface to Underlying Mappings

The surface-to-underlying mappings are used for hit testing—converting screen coordinates to a position for a selection in the underlying text. The “before” association indicates the character associated with the leading edge of the glyph, and the “after” association indicates the character associated with the trailing edge of the glyph.

If the glyph is a ligature, the character locations of its components will also be listed.

Mapping Correspondences

Note that there should be a close correspondence between the numbers in the underlying-to-surface and surface-to-underlying tables. (Any lack of correspondence is probably due to a bug in the Graphite engine!) For example:

```
    UNDERLYING TO SURFACE MAPPINGS

    string         75     76     77     78     79     80     81     82
    segment        0      1      2      3      4      5      6      7

    Text:          a      b      x      y      z      c      d      e
    Unicode:       0061   0062   0078   0079   0080   0063   0064   0065
    before         <--    1      2      2      2      3      6      5
    after          <--    1      2      2      2      4      6      5
    ligature                     2      2      2
    component                    1      2      3

    SURFACE TO UNDERLYING MAPPINGS

                   0      1      2      3      4      5      6

    Glyph IDs:     #      0162   0222   0163   0263   0165   0166
    before                1      2      5      5      7      6
    after                 1      4      5      5      7      6
    component 1                  2
    component 2                  3
    component 3                  4
```

Notice several things. First of all, there is no surface slot that maps to underlying character 0, since it is only rendered in the previous segment. Underlying characters 2-4 all map to surface glyph 2, which in turn knows that its leading edge is character 2 and its trailing edge is character 4. Also both tables are aware that characters 2-4 form a ligature which is surface glyph 2.

Surface glyph 4 (0x0263) represents an inserted glyph, which along with glyph 3, both serve to render character 5. And that information is reflected in the associations for character 5. Finally, characters 6 and 7 have been reversed and are rendered as glyphs 6 and 5, respectively.

Supplementary-plane characters

As mentioned above, the underlying text is in UTF-16 form, so that surrogates are used to represent supplementary-plane characters. All of the surface-to-underlying mappings are in terms of the indices of the UTF-16 characters. For a surrogate pair, all the surface-to-underlying mappings are recorded in terms of the first of the pair, and there should be no mappings to the second of the pair. For example, the following shows a straightforward one-to-one mapping with an supplementary -plane character:

```
    UNDERLYING TO SURFACE MAPPINGS

    string         55     56     57     58     59     60
    segment        0      1      2      3      4      5

    Text:          a      b                    d      e
    Unicode:       0061   0062   DE83 + DD99   0064   0065
    before         1      2      3             4      5
    after          1      2      3             4      5

    SURFACE TO UNDERLYING MAPPINGS

                   0      1      2      3      4      5

    Glyph IDs:     #      0034   0043   0163   0061   0097
    before                0      1      2      4      5
    after                 0      1      2      4      5
```

4 Helpful hints

When you are trying to track down a problem in your renderer, it may be helpful to isolate the problem to as short a string as possible. This will eliminate spurious data in the log and make it easier to trace.

Remember to use a fixed-width font to display the gr_xductn.log file.

5 File Name

TransductionLog.rtf