Graphite Description Language

A formal grammar description

Version 1.0A formal grammar description

Sharon Correll
SIL Non-Roman Script Initiative (NRSI)

1 Introduction

The Graphite Description Language (GDL) is the programming language used by the Graphite package to describe the behavior of complex fonts. A program written in GDL can be compiled against a TrueType font to create a Graphite-enabled font, which can then be used by the Graphite engine to perform smart complex rendering.

2 Version

This file describes GDL version 2.003.

3 Notes on BNF Syntax

The follow describes elements of the BNF syntax and Extended BNF used in this document.

0 Identifiers are enclosed in angle brackets, i.e., <identifier>.

1 A vertical bar indicates logical alternatives.

2 Terminal symbols are surrounded with double or single quotes, i.e., "a", '"'.

3 Square brackets [ ] indicate an optional item or sequence.

4 Parentheses ( ) indicate grouping.

5 Braces { } indicate an optional repeated item or sequence.

6 A function syntax is used indicate primitives representing ASCII characters: ASCII(end-of-line), ASCII(32), ASCII(32..126).

7 Each rule terminates with a semicolon.

4 Grammar

4.1 Global and Environment Declarations

<gdlProgram> ::= { <globalDeclaration> | <topLevelDeclaration> } ;

<globalDeclaration> ::= <identifierChain> <equalOrPlusEqual> <expressionOrList>
<optSemiColon> ;

<topLevelDeclaration> ::= <topEnvironmentDecl> | <tableDeclaration> ;

<topEnvironmentDecl> ::= <kEnvironment> <directives> [<semiColon>]
{ topLevelDeclaration | globalDeclaration }
<kEndenvironment> <optsemiColon> ;

<directives> ::= <leftBrace>
[<directive> { <semiColon> <directive> } <optsemiColon>]
<rightBrace> <optsemiColon> ;

<expressionOrList> ::= (<leftParen> <expressionList> <rightParen>)
| <expression> ;

<directive> ::= <identifier> <equal> <expression>;

4.2 Table Declaration

<tableDeclaration> ::= <kTable>
(<nameTable> | <glyphTable> | <featureTable> | <subTable>
| <justTable> | <posTable> | <lineBreakTable> | <otherTable>)
<kEndtable> <optSemiColon> ;

4.3 Name Table

<nameTable> ::= <leftParen> <kName> <rightParen> [<directives>]
<optSemiColon>
{ <nameEnv> | <nameSpecList> | <tableDeclaration> } ;

<nameEnv> ::= <kEnvironment> [<directives>] <optSemiColon>
{ <nameSpecList> | <nameEnv> | <tableDeclaration> }
<kEndenvironment> <optSemiColon> ;

<nameSpecList> ::= (structuredNameSpec [<nameSpecList>])
| (flatNameSpec [<semiColon> <nameSpecList>]
<optSemiColon>) ;

<structuredNameSpec> ::= <identifier> <leftBrace> <nameSpecList> <rightBrace>
<optSemiColon> ;

<flatNameSpec> ::= (<identifier> <dot>
(<flatNameSpec> | <structuredNameSpec>))
| ((<identifier> | <languageID>) <equalOrPlusEqual>
<stringDefn>) ;

<stringDefn> ::= <literalString>
| <stringFunction>
| (<leftBrace> <stringDefn> { <comma> <stringDefn> }
<rightBrace>) ;

The third option above is permitted by the implementation but doesn't make much sense:
stringName.LG_USENG = { "string1", "string2", string("abc", 345) }

<stringFunction> ::= <kString> <leftParen> <literalString> [<comma> <codePage>]
<rightParen> ;

<languageID> ::= <literalInteger> ;

<codePage> ::= <literalInteger> ;

4.4 Glyph Table

<glyphTable> ::= <leftParen> <kGlyph> <rightParen> [<directives>]
<optSemiColon>
{ glyphEnv | glyphEntry | tableDeclaration } ;

<glyphEnv> ::= <kEnvironment> [<directives>] <optSemiColon>
{ glyphEntry | glyphEnv | tableDeclaration }
<kEndenvironment> <optSemiColon> ;

<glyphEntry> ::= (glyphContents | glyphAttributes) <optSemiColon> ;

<glyphContents> ::= <identifier> <equalOrPlusEqual> <glyphSpec> <attributes> ;

<glyphAttributes> ::= <identifer>
(<leftBrace> <attrItemList> <rightBrace>)
| (<dot> (flatAttrItem | structuredAttrItem)) ;

<glyphSpec> ::= <identifier>
| <codepointFunction> | <glyphidFunction>
| <postscriptFunction> | <unicodeFunction>
| <unicodeCodepoint> | <pseudoFunction>

| (<leftParen> <glyphSpec> [<optComma> <glyphSpec>]
<rightParen>) ;

4.4.1 Glyph Functions and Lists

<pseudoFunction> ::= <kPseudo> <leftParen>
(<codepointFunction> | <glyphidFunction>
| <postscriptFunction> | <unicodeFunction>
| <unicodeCodepoint>
)
<optComma> <intOrUniHex> <rightParen> ;

<codepointFunction> ::= <kCodepoint> <leftParen> <codepointList>
[<comma> <literalInteger>]
<rightParen> ;

<codepointList> ::= (<leftParen> <codepointItem>
{ <optComma> <codepointItem> } <rightParen>)
| <codepointItem> ;

<codepointItem> ::- <literalString> | <charOrIntOrRange> ;

<glyphidFunction> ::= <kGlyphid> <leftParen> <intOrRange>
{ <optComma> <intOrRange> } <rightParen> ;

<postscriptFunction> ::= <kPostscript> <leftParen> <literalString>
{ <optComma> <literalString> } <rightParen> ;

<unicodeFunction> ::= <kUnicode> <leftParen> <intOrRange>
{ <optComma> <intOrRange> } <rightParen> ;

<unicodeCodepoint> ::= (<literalUniHex> <dot> <literalUniHex>)
| <literalUniHex> ;

<intOrRange> ::= (<literalInteger> <dot> <literalInteger>)
| <literalInteger> ;

<charOrIntOrRange> ::= (<literalChar> | <literalInteger>) [<dot> (<literalChar> |
<literalInteger>)] ;

4.5 Feature Table

<featureTable> ::= <leftParen> <kFeature> <rightParen> [<directives>]
<optSemiColon>
{ featureEnv | featureEntry | tableDeclaration } ;

```
<featureEnv>          ::=   <kEnvironment> [ <directives> ] <optSemiColon>
                            { featureSpecList | featureEnv | tableDeclaration }
                            <kEndenvironment> <optSemiColon> ;

<featureSpecList>     ::=   ( structuredFeatureSpec [ <featureSpecList> ] )
                            | (  flatFeatureSpec [ <semiColon> <featureSpecList> ]
                               <optSemiColon> ) ;

<structuredFeatureSpec > ::=   ( <identifier> | <kName>  | <kValue> )
                               <leftBrace> [ <featureSpecList> ] <rightBrace>
                               <optSemiColon> ;

<flatFeatureSpec>     ::=   ( ( <identifier> | <kName> | <kValue> )
                               ( <dot> ( <flatFeatureSpec> | <structuredFeatureSpec> ) )
                               | ( <equal> <featureValue> )
                            )
                            | ( <languageID> <equal> <featureValue> ) ;

<featureValue>        ::=   <signedInt> | <stringDefn> | <identifier> ;
```

4.6 Substitution and Justification Tables

```
<subTable>            ::=   <leftParen> <kSubstitution> <rightParen> <directives>
                            <optSemiColon> { subEntry } ;

<justTable>           ::=   <leftParen> <kJustification> <rightParen> <directives>
                            <optSemiColon> { subEntry } ;

<subEntry>            ::=   <subIf> | <subRule> | <subPass> | <subEnv>
                            | <tableDeclaration> ;

<subEnv>              ::=   <kEnvironment> [ <directives> ] <optSemiColon>
                            { <subEntry> } <kEndenvironment> <optSemiColon> ;

<subPass>             ::=   <kPass> <leftParen> <literalInteger> <rightParen> <directives>
                            <optSemiColon> { <subEntry> } <kEndpass> <optSemiColon> ;

<subIf>               ::=   <kIf> <leftParen> <expression> <rightParen>
                            { <subEntry> }
                            { <subElseIf> }
                            [ <kElse> { <subEntry> } ]
                            <kEndif> <optSemiColon> ;

<subElseIf>           ::=   <kElseif> <leftParen> <expression> <rightParen> { <subEntry> } ;
```

```
<subRule>              ::=   [ <subLhs> <transformsInto> ]
                             <subRhs> [ <slash> <context> ] <semiColon> ;

<subLhs>               ::=   <subLhsRange> { <subLhsRange> } ;

<subLhsRange>          ::=   <subLhsOptionalList>
                             | ( subLhsItem [ <questionMark> ] ) ;

<subLhsOptionalList>   ::=   <leftBracket> <subLhs> { <subLhs> } <rightBracket>
                             <questionMark> ;

<subLhsItem>           ::=   ( <underscore> | <glyphSpec> ) <alias>

<subRhs>               ::=   <subRhsItem> { <subRhsItem> } ;

<subRhsItem>           ::=   ( <underscore>
                             | ( <atPlusSlotIndicator> [ <colon> <associations> ] )
                             | ( <glyphSpec> [ <associationsPlusSelector> ] )
                             )
                             <alias>
                             <attributes>

<assocationsPlusSelector> ::=  ( <dollar> <slotIndicator> [ <colon> <associations> ] )
                               | ( <colon> <associations> [ <dollar> <slotIndicator> ] ) ;

<associations>         ::=   <slotIndicator> | <associationsList> ;

<associationsList>     ::=   <leftParen> [ <slotIndicator> { <optComma> <slotIndicator> } ]
                             <rightParen> ;

<atPlusSlotIndicator>  ::=   <atPlusIdentifier> | ( <atSign> [ <literalInteger> ] ) ;

<slotIndicator>        ::=   <literalInteger> | <identifier> ;

<alias>                ::=   <equal> <identifier> ;
```

4.7 Positioning and Line Break Tables

```
<posTable>             ::=   <leftParen> <kPositioning> <rightParen> <directives>
                             <optSemiColon> { posEntry } ;

<lineBreakTable>       ::=   <leftParen> <kLinebreak> <rightParen> <directives>
                             <optSemiColon> { posEntry } ;
```

```
<posEntry>          ::=  <posIf> | <posRule> | <posPass> | <posEnv>
                         | <tableDeclaration> ;

<posEnv>            ::=  <kEnvironment> [ <directives> ] <optSemiColon>
                         { <posEntry> } <kEndenvironment> <optSemiColon> ;

<posPass>           ::=  <kPass> <leftParen> <literalInteger> <rightParen> <directives>
                         <optSemiColon> { <posEntry> } <kEndpass> <optSemiColon> ;

<posIf>             ::=  <kIf> <leftParen> <expression> <rightParen>
                         { <posEntry> }
                         { <posElseIf> }
                         [ <kElse> { <posEntry> } ]
                         <kEndif> <optSemiColon> ;

<posElseIf>         ::=  <kElseif> <leftParen> <expression> <rightParen> { <posEntry> } ;

<posRule>           ::=  <posRhs> [ <slash> <context> ] <semiColon> ;

<posRhs>            ::=  <posRhsRange> { <posRhsRange> } ;

<posRhsRange>       ::=  <posRhsOptionalList>
                         | ( posRhsItem [ <questionMark> ] ) ;

<posRhsOptionalList> ::= <leftBracket> <posRhs> { <posRhs> } <rightBracket>
                         <questionMark> ;

<posRhsItem>        ::=  <glyphSpec> <alias> <attributes>
```

4.8 Context

```
<context>           ::=  { <contextRange> } ;

<contextRange>      ::=  <contextList> | <caret> | ( <contextItem> [ <questionMark> ] ;

<contextList>       ::=  <leftBracket> <contextRange> { <contextRange> } <rightBracket>;

<contextItem>       ::=  ( <hash> | <underscore> | <glyphSpec> )
                         <alias> <constraint>

<constraint>        ::=  <leftBrace> <expression> <rightBrace> ;
```

4.9 Attributes

<attributes> ::= <leftBrace> [<attrItemList] <optSemiColon> <rightBrace> ;

<attrItemList> ::= <structuredAttrItem> | <flatAttrItem> ;

<structuredAttrItem> ::= (<identifier> | <literalInteger>) <leftBrace> [<attrItemList>]
<optSemiColon> <rightBrace> ;

<flatAttrItem> ::= (<dot> <attrItemList>)
| (<attrAssignmentOp (<function> | <expression>)) ;

<attrAssignmentOp> ::= <equal> | <plusEqual> | <minusEqual> | <divEqual>
| <multEqual> ;

4.10 Expressions

<expression> ::= <conditionalExpression> ;

<expressionList> ::= <expression> [<comma> <expression] ;

<conditionalExpression> ::= <logicalOrExpression>
[<questionMark> <expression> <colon> <expression>] ;

<logicalOrExpression> ::= <logicalAndExpression>
[<orOperator> <logicalAndExpression>] ;

<logicalAndExpression> ::= <comparativeExpression>
[<andOperator> <comparativeExpression>] ;

<comparativeExpression> ::= <additiveExpression>
{ <comparativeOperator> <additiveExpression> } ;

<comparativeOperator> ::= <equalEqual> | <notEqual> | <lessThan> | <lessThanOrEqual>
| <greaterThan> | <greaterThanOrEqual> ;

<additiveExpression> ::= <multiplicativeExpression>
{ <additiveOperator> <multiplicativeExpression> } ;

<additiveOperator> ::= <plus> | <minus> ;

<multiplicativeExpression> ::= <unaryExpression>
{ <multiplicativeOperator> <unaryExpression> };

<multiplicativeOperator> ::= <mult> | ;

<unaryExpression> ::= [<unaryOperator>] <singleExpression> ;

<unaryOperator> ::= <notOperator> | <minus> ;

<singleExpression> ::= (<leftParen> <expression> <rightParen>)
| <literalString> | <arithmeticFunction> | <lookupExpression>
| <signedInt> ;

<lookupExpression> ::= [<selectorExpression> <dot>]
<identifierChain> [<clusterExpression>] ;

<selectorExpression> ::= <atPlusIdentifier> | (<atSign> <literalInteger>) ;

<clusterExpression> ::= <dot> <literalInteger> ;

4.11 Functions

<arithmeticFunction> ::= (“max” | “min”) <leftParen> [<expressionList>] <rightParen> ;

<function> ::= <identifier> <leftParen> [<expressionList>] <rightParen> ;

4.12 Other

<intOrUniHex> ::= <literalInteger> | <literalUniHex> ;

<identifierChain> ::= (<identifier> | “position”) [<dot> <identifierChain>] ;

<optSemiColon> ::= [<semiColon>] ;

<optComma> ::= [<comma>] ;

<transformsInto> ::= <greaterThan> ;

<slash> ::= ;

5 Lexical Tokens

5.1 Whitespace

Whitespace may occur between any two lexical items. Whitespace may not be included in the middle of a token except where specified (e.g., <kElseif>).

5.2 Comments

Comments may occur anywhere whitespace may occur, and are ignored with respect to the generation of the lexical tokens.

<comment> ::= <embeddedableComment> | <endOfLineComment> ;

<embeddedableComment> ::= “/*” { <embeddedableComment> | <lowerAsciiChar> } “*/” ;

<endOfLineComment> ::= “//” { <lowerAsciiChar> } <endOfLine> ;

5.3 Keywords

<kCodepoint> ::= “codepoint” ;

<kElse> ::= “else” ;

<kElseif> ::= “elseif” | “else” <space> { <space> } if” ;

In other words, “else” followed by any number of space characters (but not an intervening new-line) followed by “if” is treated as equivalent to “elseif”.

<kEnvironment> ::= “environment” | “env” ;

<kEndenvironment> ::= “endenvironment” | “endenv” ;

<kEndif> ::= “endif”

<kEndpass> ::= “endpass” ;

<kEndtable> ::= “endtable” ;

<kFeature> ::= “feature” ;

<kGlyph> ::= “glyph” ;

<kGlyphid> ::= “glyphid” ;

<kIf> ::= “if” ;

<kJustification> ::= “justification” | “just” ;

<kLinebreak> ::= “linebreak” | “lb” ;

<kName> ::= “name” ;

<kPass> ::= “pass” ;

<kPosition> ::= “position” | “pos” ;

<kPositioning> ::= “positioning” | “position” | “pos” ;

<kPostscript> ::= “postscript” ;

<kPseudo> ::= “pseudo”

<kString> ::= “string” ;

<kSubstitution> ::= “substitution” | “subs” | “sub”;

<kTable> ::= “table” ;

<kUnicode> ::= “unicode” ;

<kValue> ::= “value” ;

5.4 Numbers and Identifiers

<signedInt> ::= “true” | “false” | [<plus> | <minus>] <literalInteger> ;

<identifier> ::= <alpha> { <underscore> | <alpha> | <digit> };

<atPlusIdentifier> ::= “@” <identifier> ;

<alpha> ::= (“a” | “b” | “c” | “d” | “e” | “f” | “g” | “h” | “i” | “j” | “k” | “l” | “m” | “n” | “o” | “p” | “q” | “r” |”s” | “t” | “u” | “v” | “w” | “x” | “y” | “z” | “A” | “B” | “C” | “D” | “E” | “F” | “G” | “H” | “I” | “J” | “K” | “L” | “M” | “N” | “O” | “P” | “Q” | “R” | “S” | “T” | “U” | “V” | “W” | “X” | “Y” | “Z”)

```
<digit>          ::=  ( “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9” );

<hexDigit>       ::=  ( “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”
                      | “a” | “b” | “c” | “d” | “e” | “f”
                      | “A” | “B” | “C” | “D” | “E” | “F” ) ;

<literalInteger> ::=  ( <digit> { <digit> } )
                      | ( “0x” <hexDigit> { <hexDigit> } )
                      [ “m” | “M” ] ;

<literalString>  ::=  <doubleQuote>
                      { <escapeSequence> | <lowerAsciiChar> }
                      <doubleQuote>

<literalChar>    ::=  <singleQuote>
                      ( <escapeSequence> | <lowerAsciiChar> )
                      <singleQuote>

<literalUniHex>  ::=  “U+” <hexDigit> { <hexDigit> } ;

<escapeSequence> ::=  ‘\’ ( ‘n’ | ‘r’ | ‘t’ | ‘b’ | ‘f’ | ‘”’ | “’” | ‘\’) ;

<symbolChar>     ::=  ‘!’ | ‘#’ | ‘$’ | ‘%’ | ‘&’ | “’” | ‘(’ | ‘)’ | ‘*’ | ‘+’ | ‘,’
                      | ‘-’ | ‘.’ | ‘/’ | ‘:’ | ‘;’ | ‘<’ | ‘=’ | ‘>’ | ‘?’ | ‘@’ | ‘[’
                      | ‘\’ | ‘]’ | ‘^’ | ‘_’ | ‘`’ | ‘{’ | ‘|’ | ‘}’ | ‘~’ ;
```

5.5 Assignment Operators

```
<equalOrPlusEqual> ::=  <equal> | <plusEqual> ;

<equal>            ::=  “=” ;

<plusEqual>        ::=  “+=” ;

<minusEqual>       ::=  “-=” ;

<multEqual>        ::=  “*=” ;

<divEqual>         ::=  “/=” ;
```

5.6 Comparison Operators

```
<equalEqual>       ::=  “==” ;
```

<notEqual> := “!=” ;

<lessThan> ::= “<” ;

<lessThanOrEqual> ::= “<=” ;

<greaterThan> ::= “>” ;

<greaterThanOrEqual> ::= “>=” ;

5.7 Logical and Arithmetic Operators

<orOperator> ::= “||” ;

<andOperator> ::= “&&” ;

<notOperator> ::= “!” ;

<plus> ::= “+” ;

<minus> ::= “-” ;

<mult> ::= “*” ;

 ::= “/” ;

5.8 Other Symbols

<dot> ::= “.” ;

<dotDot> ::= “..” ;

<semiColon> ::= “;” ;

<comma> ::= “,” ;

<colon> ::= “:” ;

<dollar> ::= “$” ;

<underscore> ::= “_” ;

<hash> ::= “#”;

<caret> ::= “^” ;

<atSign> ::= “@”;

<questionMark> ::= “?” ;

<leftParen> ::= “(” ;

<rightParen> ::= “)” ;

<leftBrace> ::= “{” ;

<rightBrace> ::= “}” ;

<leftBracket> ::= “[“ ;

<rightBracket> ::= “]” ;

5.9 ASCII Primitives

<space> ::= ASCII(32) ;

<endOfLine> ::= ASCII(end-of-line) ;

<lowerAsciiChar> ::= ASCII(32..126) ;

6 Revision History

1. 30 April 2004. File created by Sharon Correll.

7 File Name

GDL_BNF.rtf