PDAM Text of ISO/IEC 9541-3, Font information interchange - Part 3: Glyph shape representation - Amendment 2: Additional Shape Representation Technology for Open Font Format

Foreword

In Open Font Format (ISO/IEC 14496-22) specification, there are 2 basic glyph shape representations in OpenType: **sfnt TrueType** (used in sfnt OpenType) and **embedded Adobe CFF font** (used in CFF OpenType). There is no existing ISO standard to refer these glyph shape representation methods. As a result, for the harmonization of ISO/IEC 9541 to Open Font Format, ISO/IEC 9541-3 should include the standards of these glyph shape representations.

Introduction

This amendment appends an additional glyph shape representation technology for the harmonization of ISO/IEC 9541-3 to Open Font Format (ISO/IEC 14496-22). The glyph shape representation is an extended version of ISO/IEC 9541-3 Type 1 glyph shape representation.

The additional interchange format is described by SGML (Standard Generalized Markup Language) conforming to ISO 8879:1986 and its Technical Corrigendum 2 (Annex K: Web SGML Adaptations).

Font information interchange -- Part 3: Glyph shape representation Amendment 2: Additional Shape Representation Technology for Open Font Format

Page 3

Clause 1.7.2

Replace the declaration of 1.7.2 with:

```
<!-- (c) International Organization for Standardization 2004 Permission
        to copy in any form is granted for use with comforming WebSGML
        systems and appilications as defined in ISO 8879:1986(WWW),
        provided this notice is included in all copies. -->
<!-- Public document type definition. Typical invocation:
     <!DOCTYPE gshapes PUBLIC "ISO 9541-3:1994 AM1:2004 AM2:2007//DTD Glyph Shapes//EN">
-->


<!-- GLYPHSHAPES -->
     <!ELEMENT gshapes (t1shapes | t2shapes | ot3shapes | niprop)+ )>


<!-- Type 1 shape information. Typical invocation:
     <!DOCTYPE t1shapes PUBLIC "ISO 9541-3:1994//DTD Type 1 Glyph Shapes//EN">
-->
```

```
<!-- Type 2 shape information. Typical invocation:
     <!DOCTYPE t2shapes PUBLIC "ISO 9541-3:1994 AM1:2004//DTD Type 2 Glyph Shapes//EN">
-->

<!-- Open Type 3 shape information. Typical invocation:
     <!DOCTYPE ot3shapes PUBLIC "ISO 9541-3:1994 AM2:2007//DTD Open Type 3 Glyph Shapes//EN">
-->
```

Page 45

Section 4

Add the following Section:

Section 4: Open Type 3 glyph shape representation

4.1 Scope

This section specifies the architecture and interchange format of one standard Glyph Shape Representation: ISO/IEC 9541 Standard OPEN TYPE 3. The Open Type 3 glyph shape representation is designed for the harmonization to the glyph shape representation used in “CFF” table in Open Font Format (ISO/IEC 14496-22). It is an extended version of ISO/IEC 9541-3 Type 1 glyph shape representation. The original ISO/IEC 9541-3 Type 1 glyph shape representation used graphical drawing operators that take single set of operands. For example, relative lineto operator (rlineto, described in 2.7.3.2.5) takes 2 number object operands of dx and dy that specifies relative displacements. To compress the glyph procedures, the Open Type 3 glyph shape representation enhances Type 1 glyph procedure operators to take multiple set of operands. The relative lineto operator in the Open Type 3 glyph shape representation recognizes the operand list as an array of pairs of dx and dy. By this enhancement, the glyph procedure repeating rlineto operator can be compressed to multiple sets of relative displacements and single rlineto operator. By the operator to be interpreted and the length of the operand list, the glyph procedure interpreter dynamically determines how many sets of operands are taken from the tail of operand list.

4.1.1 Compatibilities with original Type 1 glyph shape representation

The bi-directional round-trip conversion is possible between original Type 1 glyph shape representation and Open Type 3 glyph shape representation. The interpreter of the Open Type 3 glyph procedure can interpret original Type 1 glyph shape representation, except of the deprecated operators described in 4.5.

4.2 Definitions

This section uses the terms defined by 2.2. Extra terms in the following definitions are specific to this section.

4.3 Extended virtual machine

The interpretation of Open Type 3 glyph procedure is modeled by the virtual machine that is described in 2.7.1. To illustrate the interpretation of Open Type 3 glyph shape representation, "a transient array" to store any objects is introduced in state variables (described in 2.7.1.4). The Open Type 3 glyph procedure has no operators to allocate, free, initialize the transient array explicitly, it must be allocated dynamically or pre-allocated before the interpretation. The size of the transient array must at least 32 elements, although individual implementations may have longer array. As other state variables, the entries of the transient array are persistent only during the interpretation of each glyph procedures. The transient array has no default values. Therefore, it is possible that individual implementation resets all entries to number 0, or to random number, or keeps the objects stored in previous interpretation.

4.4 Open Type 3 glyph representation

By the comparison with original Type 1 glyph shape representation in the section 2, the Open Type 3 glyph procedure is classified into 4 groups.

- Original Type 1 glyph procedure operators that are not modified from the definition in section 2
- Enhanced Type 1 glyph procedure operators that the syntaxes are enhanced
- Additional operators that are not defined in original Type 1 glyph representation
- Obsolete operators that described in section 2 but should not be used in Open Type 3

Some operators in the Open Type 3 glyph procedure take the multiple sets of the operands, and the number how many sets are taken is calculated dynamically from the length of the operand list when the operator is interpreted. For the description of such operators' syntax, following notation is used for enhanced operators. Other notations are described in Table 1 and 2 in 2.7.3.

Table 3 Symbols preceding glyph procedure notation for extended Type 1 glyph representation

operand-list notation	Meaning
{...}	indicates grouping for set of operands
?	takes zero or one operand or set of operands if available in the operand list.
+	takes the array of operand or set of operands as many as available from the operand list.

The example syntax using the set of operand is

* {dxa dya}+ rlineto (00/05) *

The original syntax of relative lineto operator described in 2.7.3.2.5 takes 2 operands to draw a direct line from current point. The enhanced relative lineto operator takes multiple sets of 2 operands. Each set of 2 operands are used to append a direct line to the current point and update the current point in the direction from the head to the tail of the operand list. If the length of operand list is not the multiple of 2, the extra operands in the head of operand list are just removed (as indicated by final "*"), they have no effect on the result.

4.4.1 Original Type 1 glyph procedure operators

The following operators are same with original Type 1 glyph procedures.

4.4.1.1 Arithmetic operators

For the following arithmetic operator, giving multiple sets of operators is impossible because the result is stacked per each set.

4.4.1.1.1 `div`

This operator is same with that described in 2.7.3.4.1.

4.4.1.2 Graphical operators

For the following graphical operators, the expected result by giving multiple sets of operands is identical to the result by the final set of operands only. The detailed behaviours of the operators are described in Section 2.7.3.

4.4.1.2.1 `vmoveto`

This operator is same with that described in 2.7.3.2.10.

4.4.1.2.2 `closepath`

This operator is same with that described in 2.7.3.2.1.

4.4.1.2.3 `rmoveto`

This operator is same with that described in 2.7.3.2.6.

4.4.1.2.4 `hmoveto`

This operator is same with that described in 2.7.3.2.3.

4.4.1.2.5 `setcurrentpoint`

This operator is same with that described in 2.7.3.2.11.

4.4.1.3 Control operators

The following control operators switch the sequence of glyph procedure tokens.to be interpreted. Therefore it is impossible to take multiple sets of operands.

4.4.1.3.1 `callsubr`

This operator is same with that described in 2.7.3.5.1.

4.4.1.3.2 `return`

This operator is same with that described in 2.7.3.5.2.

4.4.1.3.3 `callutilsubr`

This operator is same with that described in 2.7.3.5.3.

4.4.1.3.4 `endglyph`

This operator is same with that described in 2.7.3.1.3.

4.4.1.4 Glyph reference and composition operators

The following operators set the reference point and escapement or compose a composite glyph of 2 component glyphs. Because the glyph procedure interpreter cannot memorize the glyph index after its rendering, the following operators cannot take multiple sets of operators.

4.4.1.4.1 `xrpe`

This operator is same with that described in 2.7.3.1.2.

4.4.1.4.2 `rpe`

This operator is same with that described in 2.7.3.1.1.

4.4.1.4.3 `siag`

This operator is same with that described in 2.7.3.1.4.

4.4.2 Enhanced Type 1 glyph procedure operators

The following operators are enhanced to take multiple set of operands.

4.4.2.1 Graphical operators

The following operators are designed to draw a line or curve between the current point memorized by interpreter and the parameters given by the set of operands. The interpretation update the current point in the interpreter. In the description of glyph outline, they are most frequently used. Collecting all control point and omitting similar operator can reduce the size of the procedure. In most enhancements, the operators are enhanced to draw the zig-zag kinked line.

4.4.2.1.1 Horizontal lineto `hlineto` (original syntax is given in 2.7.3.2.3)

This operator is interpreted by most appropriate syntax in following syntaxes:

```
* dx1 dy2 ... dxN hlineto (00/06) *
* dx1 dy2 ... dxN dy(N+1) hlineto (00/06) *
```

and appends the alternating horizontal and vertical line from the current point. The first line is horizontal and the second line is vertical (specified by only dy1).

4.4.2.1.2 Vertical lineto `vlineto` (original syntax is given in 2.7.3.2.9)

This operator is interpreted by most appropriate syntax in following syntaxes:

```
* dy1 dx2 ... dyN vlineto (00/07) *
* dy1 dx2 ... dyN dx(N+1) vlineto (00/07) *
```

and appends the alternating vertical and horizontal line from the current point. The first line is vertical and the second line is horizontal (specified by only dx1).

4.4.2.1.3 Horizontal-vertical curveto `hvcurveto` (original syntax is given in 2.7.3.2.4)

This operator is interpreted by most appropriate syntax in following syntaxes:

```
* dx1 dx2 dy2 dy3 dx3? hvcurveto (01/15) *
* dx1 dx2 dy2 dy3 {dya dxb dyb dxc dxd dxe dye dyf}+ dxf? hvcurveto (01/15) *
* {dxa dxb dxb dyc dyd dxe dye dxf}+ dyf? hvcurveto (01/15) *
```

and appends one or more Bezier curves to the current point. The specification of a Bezier curve from currentpoint requires 6 operands. They are x1, y1, x2, y2, x3, y3 in figure 6, x0 and y0 are given by currentpoint. In the first and second syntax of this operator, the tangent of beginning of first Bezier curve must be horizontal (y1=y0), and that of ending of first Bezier curve must be vertical (x3=x2). By this restriction, the number of operands for the first Bezier curve is reduced to 4 (dx1 dx2 dy2 dy3). The restriction of beginning and ending tangents are alternating. The second Bezier curve must start with vertical tangent and finish with horizontal tangent specified by 4 operands (dya, dxb, dyb, dxc), because the first Bezier curve finishes with vertical tangent. The ending tangent of the final Bezier curve is not restricted. In basic syntax, although the final Bezier curve is specified by 4 operands (dxd dxe dye dyf in the first and second syntax, or dyd dxe dye dxf in the third syntax), extra operand (dxf in the first and second syntax, or dyf in the third syntax) makes sloping end of final Bezier curve. The standard order of 2 operands is x and y to specify a point, but the order of extra operand is non-standard y and x order in the first and second syntax.

4.4.2.1.4 Vertical-horizontal curveto `vhcurveto` (original syntax is described in 2.7.3.2.8)

This operator is interpreted by most appropriate syntax in following syntaxes:

```
* dy1 dx2 dy2 dx3 dyf? vhcurveto (01/14) *
* dy1 dx2 dy2 dx3 {dxa dxb dyb dyc dyd dxe dye dxf}+ dyf? vhcurveto (01/14) *
* {dya dxb dyb dxc dxd dxe dye dyf}+ dxf? vhcurveto (01/14)
```

and appends one or more Bezier curves to the current point. The syntax is same with enhanced hvcurveto except x, y coordinates are exchanged for initial and final curves.

4.4.2.1.5 Relative lineto `rlineto` (original syntax is described in 2.7.3.2.5)

```
* {dxa dya}+ rlineto (00/05) *
```

This operator appends one or more lines to the current point. Each set of 2 operands is interpreted by original rlineto operator syntax described in 2.7.3.2.5.

4.4.2.1.6 Relative-relative lineto `rrcurveto` (original syntax is described in 2.7.3.2.7)

```
* {dxa dya dxb dyb dxc dyc}+ rrcurveto (00/08) *
```

This operator appends one or more Bezier curves to the current point. Each set of 6 operands is interpreted by original rrcurveto operator syntax described in 2.7.3.2.7.

4.4.2.2 Stem hinting operators

In Type 1 glyph reprentation, the following operators take two operands that specifies the zone to apply the hinting parameter. To set hinting parameters to parallel lines, the following operaters are enhanced to take multiple sets of operands.

4.4.2.2.1 Horizontal stem `hstem` (original syntax is described in 2.7.3.3.2)

This operator is interpreted by most appropriate syntax in following syntaxes:

```
* y dy hstem (00/01) *
```

* y dy {dya dyb}+ **hstem** (00/01) *

and specifies one or more horizontal stem hints. The initial set of 2 operands specifies the zone from y to y+dy as the original syntax described in 2.7.3.3.2. Following operands are interpreted as relative values to preceding zone. For example, the second set of 2 operands dya dyb are interpreted to specify the zone from y+dy+dya to y+dy+dya+dyb. This syntax is different from hstem3 described in 2.7.3.3.3 that specifies all zones by a set of absolute height and relative height.

4.4.2.2.2 Vertical stem `vstem` (original syntax is described in 2.7.3.3.4)

This operator is interpreted by most appropriate syntax in following syntaxes:

* x dx **vstem** (00/03) *

* x dx {dxa dxb}+ **vstem** (00/03) *

and specifies one or more vertical stem hints. The interpretation is same with hstem except of a point the operands are x coordinate values for vertical hints.

4.4.3 Additional glyph procedure operators

The following operators are introduced in Open Type 3 glyph descriptions.

4.4.3.1 Additional graphical operators

4.4.3.1.1 Relative curveto-lineto `rcurveline`

* {dxa dya dxb dyb dxc dyc}+ dxd dyd **rcurveline** (01/08) *

The preceding sets of 6 operands are interpreted by the syntax for enhanced rrcurveto operator described in 4.4.2.1. The final pair of operands is interpreted by the syntax for rlineto described in 2.7.3.2.5.

4.4.3.1.2 Horizontal-horizontal curveto `hhcurveto`

* dy1? {dxa dxb dyb dxc}+ **hhcurveto** (01/11) *

This operator appends one or more Bezier curves from current point. Except of the first Bezier curve, beginning and ending tangents of all Bezier curves must be horizontal to specify a Bezier curve by only 4 operands. If extra operand (dy1) is given, the beginning tangent of first Bezier curve is slanted.

4.4.3.1.3 Vertical-vertical curveto `vvcurveto`

* dx1? {dya dxb dyb dyc}+ **vvcurveto** (01/10) *

This operator appends one or more Bezier curves from current point. Except of the first Bezier curve, beginning and ending tangents of all Bezier curves must be vertical to specify a Bezier curve by only 4 operands. If extra operand (dx1) is given, the beginning tangent of first Bezier curve is slanted.

4.4.3.1.4 Flex `flex`

* dx1 dy1 dx2 dy2 dx3 dy3 dx4 dy4 dx5 dy5 dx6 dy6 fd **flex** (00/12 02/03) *

This operator appends a flex line (described in 2.8.3) consists of 2 Bezier curves, from current point (the position of current point is described by x0, y0). 2 Bezier curves are joint so 12 operands are required to specify 2 Bezier curves, and an operand is used to specify the flex depth.

4.4.3.1.5 Flex 1 `flex1`

* dx1 dy1 dx2 dy2 dx3 dy3 dx4 dy4 dx5 dy5 d6 **flex1** (00/12 02/05) *

This operator appends a flex line (described in 2.8.3) consists of 2 Bezier curves, from current point (the position of current point is described by x0, y0). In comparisn with flex, the flex depth is fixed to 0.5 (it corresponds to the case fd=50 is given to flex). The interpretation of the final operand d6 is dependent with the geometry of 5 control points. If abs(dx1+dx2+dx3+dx4+dx5) > abs(dy1+dy2+dy3+dy4+dy5), the ending point of the flex line is defined by (x0+d6, y0+dy1+dy2+dy3+dy4+dy5). Otherwise, the ending point of the flex line is defined by (x0+dx1+dx2+dx3+dx4+dx5, y0+d6).

4.4.3.1.6 Horizontal flex `hflex`

* dx1 dx2 dy2 dx3 dx4 dx5 dx6 **hflex** (00/12 02/02) *

This operator appends a flex line (described in 2.8.3) consists of 2 Bezier curves from current point. In comparison with flex, the heights of the beginning and ending points of the flex must be same (dy6=0), and the tangents at beginning, ending and joining point must be horizontal (dy2=dy3=dy4, dy1=dy5=0). By these restrictions, the number of operands to specify 2 Bezier curves is reduced to 6. The flex depth is fixed to 0.5 (it corresponds to the case that fd=50 is given to flex).

4.4.3.1.7 Horizontal Flex 1 `hflex1`

* dx1 dy1 dx2 dy2 dx3 dx4 dx5 dy5 dx6 **hflex1** (00/12 02/04) *

This operator appends a flex line (described in 2.8.3) consists of 2 Bezier curves from current point. In comparison with flex, the heights of the beginning and ending points of the flex must be same (dy6=0) and the tangents at joining point must be horizontal (dy2=dy3=dy4). By these restrictions, the number of operands to specify 2 Bezier curves is reduced to 9. The flex depth is fixed to 0.5 (it corresponds to the case that fd=50 is given to flex).

4.4.3.2 Additional hinting operators

4.4.3.2.1 Horizontal stem hintmask `hstemhm`

This operator is interpreted by most appropriate syntax in following syntaxes:

* y dy **hstemhm** (01/02) *

* y dy {dya dyb}+ **hstemhm** (01/02) *

and specifies hint zones in the syntax same with enhanced hstem. hstemhm is used to register the hint zone information to be used by hintmask operator.

4.4.3.2.2 Vertical stem hintmask `vstemhm`

This operator is interpreted by most appropriate syntax in following syntaxes:

* y dy **vstemhm** (01/07) *

* y dy {dya dyb}+ **vstemhm** (01/07) *

and specifies hint zones in the syntax same with enhanced vstem. vstemhm is used to register the hint zone information to be used by hintmask operator.

4.4.3.2.3 Hintmask `hintmask`

This operator is interpreted by most appropriate syntax in following syntaxes:

* **hintmask** (01/03) **bitmask** *

and enable/disables the hint zones (previously declared by hstemhm and vstemhm) by bitmask. The bitmask object following to hintmask operator is a part of the operator. The length of bitmask object must be the minimum octet length to cover all hint zones, by bit per zone. In the bitmask, the most significant bit of the first octet enables/disables the first hint zone which is previously declared. If the corresponding bit is set to 1, the hint zone is enabled. Otherwise, the hint zone is ignored. The hint zones must not overlap.

4.4.3.2.4 Counter mask `cntrmask`

* **cntrmask** (01/04) **bitmask** *

This operator specifies the priorities of the hint zones (previously declared by hstem, hstemhm, vstem and vstemhm) by bitmask. The bitmask object following to cntrmask operator is a part of the operator. The length of bitmask object must be the minimum octet length to cover all hint zones, by bit per zone. In the bitmask, the most significant bit of the first octet enables/disables the first hint zone which is previously declared. If the corresponding bit is set to 1, preceding cntrmask operator specifies the priority of the hint zone. The hint zones specified by the first cntrmask has the highest priority. The hint zones specified by the second cntrmask have the second priority. The hint zones may overlap.

4.4.3.3 Additional control operators

4.4.3.3.1 Additional conditional operators

The following operators are conditional operates that creates and consumes numerical object on the stack. The non-zero numerical object is recognized as boolean true, and zero is recognized as boolean false. The operators and, or, and not are not bitwise operators.

4.4.3.3.1.1 And `and`

```
num1 num2 and (00/12 00/03) 1_or_0
```

This operator puts a 1 on the operand list when both of num1 and num2 are not zero. Otherwise, a 0 is put.

4.4.3.3.1.2 Or `or`

```
num1 num2 or (00/12 00/04) 1_or_0
```

This operator puts a 0 on the operand list when both of num1 and num2 are zero. Otherwise, a 1 is put.

4.4.3.3.1.3 Not equal `not`

```
num1 not (00/12 00/05) 1_or_0
```

This operator puts a 1 on the operand list when num1 is not zero. Otherwise, a 0 is put.

4.4.3.3.1.4 Equal `eq`

```
num1 num2 eq (00/12 00/15) 1_or_0
```

This operator puts a 1 on the operand list when num1 is equal to num2. Otherwise, a 0 is put.

4.4.3.3.1.5 If else `ifelse`

```
res1 res2 num1 num2 ifelse (00/12 01/06) res1_or_res2
```

This operator leaves the result res2 if num1 is greater than num2. Otherwise, res1 is left. This operator is usually used to select the number to call subroutine.

4.4.3.3.2 Additional operand-list manipulate operators

The following operators are introduced to manipulate the objects in the operand list. It should be noted that some of following operators are not equal to the operators defined in ISO/IEC 10180:1995 21.1 with same name. For example, index operator of Open Type 3 glyph procedure operator is different from index operator in ISO/IEC 10180:1995 21.1.13, but it is identical to copy operator specified by ISO/IEC 10180:1995 21.1.8.

4.4.3.3.2.1 Drop `drop`

```
any drop (00/12 01/02)
```

This operator discards an object on the head of the operand list.

4.4.3.3.2.2 Duplicate `dup`

```
num dup (00/12 01/11) num num
```

This operator duplicates the top object in the operand list.

4.4.3.3.2.3 Exchange `exch`

```
num1 num0 exch (00/12 01/12) num1 num0
```

This operater exchanges the top 2 objects in the operand list.

4.4.3.3.2.4 Index `index`

```
num(N-1) ... num0 J index (00/12 01/13) num(N-1) ... num0 num(J)
```

This operator duplicates the object stored the depth i in the operand list and puts it in the head of the operand list.

4.4.3.3.2.5 Roll `roll`

```
num(N-1) ... num0 N J roll (00/12 01/14) num((J-1) mod N) ... num0 num(N-1) ... num(J mod N)
```

This operator performs a circular shift of the elements num(N-1) ... num0 on the operand list by the amount J. Positive J indicates upward motion of the stack; negative J indicates downward motion. The value N must be a non-negative integer, otherwise the result is undefined.

4.4.3.4 Additional arithmetic operators

4.4.3.4.1 Absolute `abs`

```
num1 abs (00/12 00/09) num2
```

This operator calculates the absolute value (num2) of num1.

4.4.3.4.2 Add `add`

```
num1 num2 add (00/12 00/10) sum
```

This operator calculates the sum of two numbers (num1 and num2) in operands.

4.4.3.4.3 Substract `sub`

```
num1 num2 sub (00/12 00/11) difference
```

This operator calculates the difference that num2 is substracted from num1.

4.4.3.4.4 Negate `neg`

```
num1 neg (00/12 00/14) -num1
```

This operator puts the negative of num1 on the operand list.

4.4.3.4.5 Random `random`

```
random (00/12 01/07) num
```

This operator generates a pseudo random number num. The range of num is greater than 0 and less than or equal to 1.

4.4.3.4.6 Multiply `mul`

```
num1 num2 mul (00/12 01/08) product
```

This operator calculates the product of num1 and num2. The operand num1 is pushed on the operand list, followed by num2. After execution, the result (product) is pushed on the head of the operand list. The operands num1 and num2 are of type Number, and the product is of type number. If overflow occurs, the result is undefined. If underflow occurs, the result is zero.

4.4.3.4.7 Square root `sqrt`

```
num sqrt (00/12 01/10) square_root
```

This operator calculates the square root of num. The operand num is pushed on the head of the operand list. After execution, the result is pushed on the head of the operand list. The operand num is of type Number, and the result (square_root) is of type Number. If num is negative number, the result is undefined.

4.4.3.5 Additional storage operators

The transient array is manipulated only by following 2 operators to store and load with entry index.

4.4.3.5.1 Put `put`

```
any i put (00/12 01/04)
```

This operator overwrites the i-th entry of the transient array by the operand (any).

4.4.3.5.2 Get `get`

```
i get (00/12 01/05) array_entry(i)
```

This operator copies i-th entry of the transient array and pushed on the head of the operand list. If i-th entry is not written by put operator during the glyph procedure, the result is undefined.

4.5 Deprecated operators in Open Type 3 glyph descriptions

4.5.1 Dot section `dotsection` (described in 2.7.3.3.1)

```
- dotsection (00/12 00/00) *
```

The Open Type 3 glyph procedure interpreter ignores this operator. It is interpreted as a no-op. The dotsection operator is used as a delimiter between the end of preceding closed subpath and the beginning of following closed subpath. The interval between a subpath to another subpath can be detected dynamically, thus the insertion of dotsection is not essential.

4.5.2 Return value `retval` (described in 2.7.3.5.4)

```
– retval (00/12 01/01) number
```

This operator is used in the subroutine caller, just after callutilsubr operator (described in 2.7.3.5.3), to retrieve the implicit result calculated in callee subroutine and place it to the operand list. In Open Type 3 glyph description, the operand list manipulated in the callee subroutine is automatically carried over to the caller routine. Therefore, retval operator is not required. Adobe TechNote #5177 defines (00/12 01/01) as reserved sequence. It does not mention this sequence is just ignored or causes any error. Even if (00/12 01/01) is simply ignored, the interpretation can be different from the original Type 1 glyph description, because the number of repeating retval operators defines how many result objects are retrieved from the callee subroutine. Such limitation should be coded by drop operator.

4.5.3 Horizontal and vertical stem 3 operators

`* y0 dy0 y1 dy1 y2 dy2` **`hstem3`** `(00/12 00/02) *` (described in 2.7.3.3.2)

`* x0 dx0 x1 dx1 x2 dx2` **`vstem3`** `(00/12 00/01) *` (described in 2.7.3.3.5)

These 2 operators take 6 operands to specify 3 stem zones. The requirement of them are supported by the enhancement of two stem hinting operators. As a result, these operators are deprecated in Open Type 3 glyph description. Adobe TechNote #5177 defines the sequences (00/12 00/01) and (00/12 00/02) as reserved sequences. It does not mention these sequences are just ignored, ignored with 6 preceding operands, or cause any error.

4.6 Interchange format

4.6.1 Extensions to font interchange format for Open Type 3 glyph shape information

The difference between original Type 1 glyph shape representation and Open Type 3 glyph shape representation is only the glyph operators and the interpreter for the glyph operators. As a result, most part of the interchange format for original Type 1 glyph shape representation described in 2.9 can be shared. Thus, only the differences are described in following sections.

4.6.1.1 ASN.1 interchange format

The initial 2 lines of ASN.1 interchange format described in 2.9.1.1 are replaced by following lines.

```
ISO9541-GST1X { 1 0 9541 3 X X } DEFINITIONS ::= BEGIN
IMPORTS Structured-Name   FROM ISO9541-GST1 { 1 0 9541 3 0 0 }
```

4.6.1.2 SGML interchange format

The DOCTYPE declaration of SGML interchange format in 2.9.1.2 is replaced by following line.

```
<!DOCTYPE ot3shapes
          PUBLIC "ISO 9541-3:1994 AM2:2008//DTD
          Open Type 3 Glyph Shapes//EN">
```

4.6.1.3 Interchange format for glyph procedures

The interchange format for glyph procedure for Open Type 3 glyph shape representation is exactly same with the representation of original Type 1 glyph procedure number representation described in 2.9.2.