Xgridfit Reference

For a more discursive treatment, see the full documentation at http://xgridfit.sourceforge.net/ (where the sections on installation, configuration and command-line arguments are not applicable to Xgridfit 3).

Installation

1. Set up a virtual environment for Python 3, if you don't already have one, or start the one you have.

2. Unpack the archive and change to the Xgridfit root directory (the one containing setup.py).

3. At the command line, type `pip install .` (don't forget the period!)

4. Type `xgridfit -h` for a usage summary.

Command line

```
usage: xgridfit [-h] [-e] [-c] [-n] [--nocompilation] [-m] [-r]
                [--initgraphics {yes,no}] [-a {yes,no}] [-q] [-g GLYPHLIST]
                [-i INPUTFONT] [-o OUTPUTFONT] [-s] inputfile [outputfile]
```

Only `inputfile` (the name of the file containing Xgridfit programming) is required.

-e, --expand:	Convert file to expanded syntax, save, and exit. Supply the filename after the name of your Xgridfit program, e.g. `$xgridfit --expand myinstrs.xgf myinstrs_expanded.xgf`
-s, --compact	Convert file to compact syntax, save, and exit. Works like --expand, but in reverse.
-n, --novalidation	Skip validation of the Xgridfit program. Ordinarily, Xgridfit validates your .xgf file before compiling. Use this option to skip that step.
--nocompilation	Skip compilation of the Xgridfit program. No new font file will be written, but Xgridfit will validate your .xgf file.
--nocompact	Do not compact glyph programs. Ordinarily Xgridfit compacts glyph instructions by consolidating all PUSH instructions into one. This

	produces a significant reduction in file size, but can make debugging difficult.
-m, --merge	Merge the instructions generated from your .xgf file with existing instructions in a font. Your glyph programs will replace those in the font, your functions will be appended to those in the font, your prep program will either be appended to or replace the one in font (depending on --replaceprep), your cvt entries will be appended to those in the font, and the same will be done with the cvar table (if any). In effect, your own programming will be layered on top of the programming already in the font. Indexes of functions, control values and storage locations will be adjusted so that neither layer of programming interferes with the other.
-r, --replaceprep	In merge-mode (with the --merge option), your prep programming is ordinarily appended to what it already in the font. Use this option to replace it instead. (*Caution: it is important to understand what the prep program is doing in the font you are merging with. This is true whether you are replacing or supplementing it.*)
--initgraphics	Whether to initialize graphics-tracking variables at the beginning of glyph programs. The default is "yes" for compatibility with older Xgridfit programs, but for new fonts this should ordinarily be "no." Better to set it via a <default> in the .xgf file than here.
-a, --assume_y	Whether the compiler should assume that your hints are all vertical. The default is "no," but in new fonts it should be "yes" (it can be overridden for individual glyph programs). Better to set it via a <default> in the .xgf file than here.
-q, --quiet	Suppresses progress messages. Include it twice (-qq) to suppress warnings as well. Error messages cannot be suppressed.
-g, --glyphlist	Space-separated list of glyphs to compile. By default, Xgridfit compiles every glyph program; using this option can make for faster testing.
-i, --inputfont	The font file to add instructions to. This must be specified either here or in the .xgf file's <inputfont> element.
-o, --outputfont	The font file to write. This should be specified either here or in the .xgf file's <outputfont> element. If omitted, no font will be written.
-s, --saveprograms	Save generated instructions to text files. fpgm instructions are saved in `fpgm.instructions`, prep in `prep.instructions`. Each compiled glyph program (use --glyphlist to limit the number of these) is saved in a file with a name matching that of a .glif file in a UFO, and with the extension `.instructions`.

Normal vs. Compact Syntax

To speed the writing of code, Xgridfit 3 introduces a **compact syntax**, which produces source files a fraction of the size of those using the **normal syntax** of Xgridfit 2. Either normal or compact syntax will work in Xgridfit 3, but you must use one or the other: you cannot mix them in the same file. To help keep them straight, Xgridfit comes with two separate Relax-NG schemas: xgridfit.rnc (or .rng) for normal syntax, and xgridfit-sh.rnc (or .rng) for compact.

The compact syntax features shorter tag names for the most commonly used elements and attributes, and sometimes attributes can be substituted for elements. For example, a <move> element like this

```
<move distance="xheight">
  <point num="5"/>
  <shift>
    <point num="10"/>
  </shift>
</move>
```

Can instead be written like this:

```
<mv di="xheight" p="5">
  <sh p="10"/>
</mv>
```

Function, macro, and other calls can be radically different in compact syntax. Normal syntax requires a <with-param> element for every function or macro parameter, but in compact syntax you can either use the shorter parameter tag <wpm> or a list of parameters somewhat resembling a Python dictionary. For example, this macro call (valid in compact syntax)

```
<callm nm="mid-stem">
  <wpm nm="ref1" val="36"/>
  <wpm nm="ref2" val="100"/>
  <wpm nm="pt1" val="23"/>
  <wpm nm="pt2" val="75"/>
</callm>
```

can be written like this:

```
<callm nm="mid-stem">
  ref1: 36, ref2: 100, pt1: 23, pt2: 75
</callm>
```

Notice the lack of quotation marks for both keys and values.

To convert compact syntax to normal and vice versa, use the --expand or --compact command-line arguments.

Xgridfit Elements

<absolute>

Converts negative to positive numbers; positive numbers stay positive.

```
<absolute value="line-width" result-to="lw"/>
```

Content

None.

Attributes

value
: Required, except when <absolute> is the child of a <formula>. Any value or expression. The value to operate on.

result-to
: Optional; not allowed when <absolute> is the child of a <formula>. The name of a variable or control value in which to store the result. If `result-to` is omitted where allowed and `value` is a variable or control value, the result is written to `value`. If `value` cannot be written to, the compiler issues a warning and the result is left on the stack.

<add>

Adds two numbers together.

```
<add value1="line-width" value2="1p"/>
```

Content

None.

Attributes

value1, value2
: Required, except when <add> is the child of a <formula>. Any value or expression. These are the values to add together.

result-to
: Optional; not allowed when <add> is the child of a <formula>. If `result-to` is omitted where allowed, Xgridfit attempts to write the result to `value1`. If `value1` cannot be written to, the compiler issues a warning and the result is left on the stack.

<alias>

Provides access to a value under some other name. The value can be a control-value, constant, variable, or any value that can be resolved to a number either at compile time or run time. The <alias> element can appear at the top level of a program (as a child of <xgridfit>), or along with other declarations at the beginning of <glyph>, <function>, <macro> or <pre-program>

elements. An alias takes precedence over all other elements, so in the case of name collisions the alias is always used. Here is a simple example:

```
<control-value name="lc-vert-stroke" value="0"/>
<alias name="lc-vert-stem" target="lc-vert-stroke"/>
```

Now a <move> element with attribute `distance="lc-vert-stem"` will use the control-value named `lc-vert-stroke`. If another control-value is named `lc-vert-stem` it will be invisible. If you want the alias to be used in just one glyph program, declare it as a child of <glyph> rather than as a child of <xgridfit>.

Content

None.

Attributes

name (or nm)
: Required. A name: any sequence of letters, numbers, hyphens, periods; no spaces permitted. This is the name under which the program may now access the value.

target
: Required. A name: any sequence of letters, numbers, hyphens, periods; no spaces permitted. The name of the value which is being renamed.

<align> (short form <al>)

Moves one or more points along the freedom vector until aligned with a reference point. Points are "aligned" when their distance from each other, measured along the projection vector, is zero. When the projection vector is "x," aligned points end up stacked vertically; when it is "y" they end up in a horizontal line. When the projection vector is set to a line, the aligned points end up arrayed along an imaginary line orthogonal to the projection vector.

The <align> element must contain at least one object to align. It may contain any number of <point>s, <range>s and <set>s. An optional <reference> element contains the point to align with. If the reference point is omitted, the current setting of RP0 is used.

<align> can and frequently should be nested inside a <move> element, in which case points are aligned relative to the point that is the target of the <move>. In both of the following cases point "m" is aligned with point "r":

```
<align>
  <reference>
    <point num="r"/>
  </reference>
  <point num="m"/>
```

```
</align>


<move>
  <point num="r"/>
  <align>
    <point num="m"/>
  </align>
</move>
```

Content

One or more <point>, <range> and <set> elements.

Attribute

compile-if
: Optional. Any value or expression that can be resolved to a number at compile time. If true (non-zero), the <align> element is compiled; otherwise the compiler passes it over.

<align-midway>

Must contain two <point> elements. Moves these along the freedom vector until they are aligned midway between their original positions. Measurement is along the projection vector.

```
<align-midway>
  <point num="a"/>
  <point num="b"/>
</align-midway>
```

Content

Two <point> elements, both required.

Attributes

None.

<call-function> (short form <callf>)

Calls a function (defined via the <function> element) by name. Parameters may be passed to the function by including several <with-param> (<wpm>) elements; or, if the function is to be called repeatedly, several <param-set> elements, each containing the <with-param> elements for one call to the function. To save space and time, you may write the parameters using a key-value syntax like that of a python dictionary. For example, instead of this:

```
<callm nm="serif-sh">
  <wpm nm="pt-a" val="41"/>
  <wpm nm="pt-b" val="42"/>
  <wpm nm="sh-a" val="bot-a"/>
</callm>
```

you can instead write

```
<callm nm="serif-sh">
  pt-a: 41, pt-b: 42, sh-a: bot-a
</callm>
```

You cannot mix this compact syntax with <with-param> or <wpm> elements in a single function call. It is available only if you're using Xgridfit's compact syntax.

If the function returns a value, it can be assigned to a variable via the `result-to` attribute. Note, however, that if a <call-function> element contains more than one <param-set>, only the value returned by the last iteration of the function is returned.

```
<call-function name="cap-serif-width">
  <param-set>
    <with-param name="ref-pt" value="left-left"/>
    <with-param name="move-pt" value="top-serif-end"/>
  </param-set>
  <param-set>
    <with-param name="ref-pt" value="left-left"/>
    <with-param name="move-pt" value="bottom-serif-end"/>
  </param-set>
</call-function>
```

Contents

Either several <with-param> elements (if the function is to be called just once) or several <param-set> elements.

Attribute

name (or nm)
: Required. The name of the function to call.

result-to
: Optional. The name of a variable or control value in which to store the value returned by the function.

<call-glyph> (short form <callg>)

The <call-glyph> element causes code for the whole of a glyph program to be compiled and inserted, in the manner of a macro. Like <call-macro>, <call-glyph> can contain <with-param>

(or <wpm>) elements, or the compact syntax described in <call-function>. For a detailed account of its use, see the chapter on functions, macros and glyph programs.

Content

If the glyph being called has one or more <param> elements, one or more <with-param> elements. <param-set> elements are not permitted here.

Attribute

ps-name (or pnm)
: Required. Must match the `ps-name` attribute of the <glyph> element being called.

<call-macro> (short form <callm>)

Causes a macro to be compiled and its code inserted at the present location.

```
<call-macro name="lc-vert-stem-with-serif">
  <with-param name="anchor" value="left-left"/>
  <with-param name="distance-from-anchor" value="hn-width"/>
  <with-param name="stem-a" value="right-right"/>
  <with-param name="serif-a" value="right-serif-right"/>
  <with-param name="stem-b" value="right-left"/>
  <with-param name="serif-b" value="right-serif-left"/>
</call-macro>
```

Content

Either one or more <with-param> elements, one for each parameter defined in the <macro> element, or several <param-set> elements if the macro is to be compiled and inserted at this place more than once. <with-param> may be omitted for each <param> element with a `value` attribute. You may use <wpm> for <with-param>, or the compact syntax described in <call-function>.

Attribute

name (or nm)
: Required. The name of the macro to call.

<call-param> (short form <callp>)

This element may be used within a <glyph> or <macro> element containing <param> elements. It will cause to be executed a snippet of code passed into a called <glyph> or <macro> via a

<with-param> element. For details, see the "Callable Parameters" section of Functions, Macros and Glyph Programs.

Content

None.

Attribute

name (or nm)
: Required. The name of the <param> to call.

<ceiling>

Yields the smallest integer greater than or equal to `value`.

Content

None.

Attributes

value
: Required, except when <ceiling> is the child of a <formula>. Any value or expression. The value to operate on.

result-to
: Optional; not allowed when <ceiling> is the child of a <formula>. If `result-to` is omitted where allowed, Xgridfit attempts to write the result to `value`. If `value` cannot be written to, the compiler issues a warning and the result is left on the stack.

<command>

Causes any TrueType instruction (except the PUSHB and PUSHW instructions) to be inserted at this place in the output. `name` is the name of the command; `modifier` is simply copied into brackets after the instruction that is generated. This element

```
<command name="MIRP" modifier="10110"/>
```

is compiled to "MIRP[10110]". The `modifier` attribute lacks portability, since it is copied in literally. A better solution is to use <modifier> elements within the <command> element, thus:

```
<command name="MIRP">
  <modifier type="rp0" value="yes"/>
```

```
    <modifier type="minimum-distance" value="no"/>
    <modifier type="round" value="no"/>
    <modifier type="color" value="black"/>
  </command>
```

Though verbose, this style allows Xgridfit to check the input code and has the potential to allow Xgridfit to vary the syntax of its output (though at present it formats output only for fontTools).

Content

One or more <modifier> elements, if required by the instruction and if the `modifier` attribute is not used.

Attributes

name (or nm)
: Required. The name of the TrueType instruction to be output. The compiler checks this against a list of instructions.

modifier
: A modifier string to be inserted *literatim* in the TrueType instruction. For example, if `command` is "MIRP" and `modifier` is "rnd", then `MIRP[rnd]` is output.

<compile-if>

Code within <compile-if> is compiled only if the `test` attribute evaluates to true (non-zero). The compiler must be able to evaluate `test` at compile time: thus it may contain only constants, number literals, control value indexes and a few operators (`+ - = != > < >= <= or and not`).

Here is a simple example of <compile-if>:

```
<compile-if test="bold-italic">
  <delta>
    <delta-set size="3" distance="-2">
      <point num="pt"/>
    </delta-set>
  </delta>
</compile-if>
```

The delta is compiled and inserted in the output code only if `bold-italic` (a global constant) is non-zero.

The <compile-if> element may also contain an <else> element, which must come last. If `test` evaluates to false (zero), the code contained in <else> is compiled.

Content

Programming to be conditionally compiled.

Attribute

test
: Required. Any value or expression that can be resolved to a number at compile time. If it evaluates as true (non-zero), the content of this element is compiled.

<constant> (short form <cn>)

A constant is a named number. The `value` can be an integer, either of the two kinds of fixed-point number (e.g. “2.3” for a distance on the grid or “1.0v” for a component of a vector), or the name of another constant, in which case it creates an alias for that constant. It can also be a simple expression (usually addition or subtraction) based on another constant:

```
<constant name="bottom-left” value="3"/>
<constant name="bottom” value="bottom-left"/>
<constant name="bottom-right” value="bottom + 4"/>
```

Constants can be referenced just about anywhere that numbers are called for. To refer to a constant belonging to another glyph program, use the glyph's `ps-name` followed by a slash and the name of the constant, e.g.

```
<point num="macron/bottom"/>
```

This is useful when instructing composite glyphs.

Constants can be declared at the beginning of a <glyph> program, or as a child of the <xgridfit> element.

Content

None.

Attributes

name (or nm)
: Required. A name: any sequence of letters, numbers, hyphens, periods; no spaces permitted. Other elements can refer to the constant by this name.

value (or val)
: Required. A number or simple expression. It should be possible to resolve this attribute to a number at compile time.

<contour>

Specifies a contour to be shifted by a <shift> instruction.

Content

None.

Attributes

num (n)
: Required. A number or simple expression, the number of the contour. It should be possible to resolve this attribute to a number at compile time.

zone
: Optional. Possible values are “glyph” and “twilight.” Include the attribute `zone="twilight"` if this contour is in the twilight zone; otherwise the compiler will assume that it is in the glyph zone.

<control-value> (short form <cv>)

The font's Control Value Table is built from the <control-value> elements. Each <control-value> has an `name` (which must be unique) and a numerical value. The index of the <control-value> is generated by Xgridfit, and no attempt should be made to predict it: Xgridfit instructions should use only the names of <control-value>s, though the index may be derived and used at run time.

```
<control-value name="curved-char-bottom” value="-25"/>
<control-value name="lc-x-height” value="850"/>
<control-value name="lc-descender-depth” value="-555"/>
<control-value name="lc-ascender-height” value="1485"/>
```

Content

None.

Attributes

name (or nm)
: Required. A name: any sequence of letters, numbers, hyphens, periods; no spaces permitted.

value (or val)
: Required. A number in font units (the units of the grid on which the font was designed, normally 1000 units per em or 2048 units per em).

color (or col)
: The color of the distance that this control value represents: it affects only the <move> element. Where this is missing, the default color (gray, unless you change it via a <default> element) will be used. A `color` attribute on the <move> element overrides this.

index
: Optional, and consulted only in merge-mode. Specifies how to determine a control-value's index. "Auto" (the default) means that if a control-value with this one's `value` is found in the font, it should be re-used; otherwise this control value should be appended to the end of the control-value table. "Append" means that the control-value should always be appended to the end of the control-value table. A number specifies the index at which the control-value should be placed. If the `value` of this control-value is different from the value of that of the control-value at that index in the existing font, the control-value is updated.

<control-value-delta>

The <control-value-delta> element works like the <delta> element, but operates on the Control Value Table rather than on a point. It should normally be invoked in the <pre-program>. Each <delta-set> inside a control-value-delta element must have `cv`, `size` and `distance` attributes, but it may not contain a point.

The setting of the vectors has no effect on the <control-value-delta>. Rather, the <delta-set> specifies an amount to add to or subtract from the value stored there.

```
<control-value-delta>
  <delta-set cv="pq-char-width" size="6" distance="-8"/>
  <delta-set cv="pq-char-width" size="9" distance="-8"/>
</control-value-delta>
```

Content

One or more <delta-set> elements.

Attributes

None.

<control-value-index>

Assigns the index of a control value to a variable. Use this if you need to get such an index for any reason, since the <set-equal> instruction yields the value, not the index, of a control value.

Content

None.

Attributes

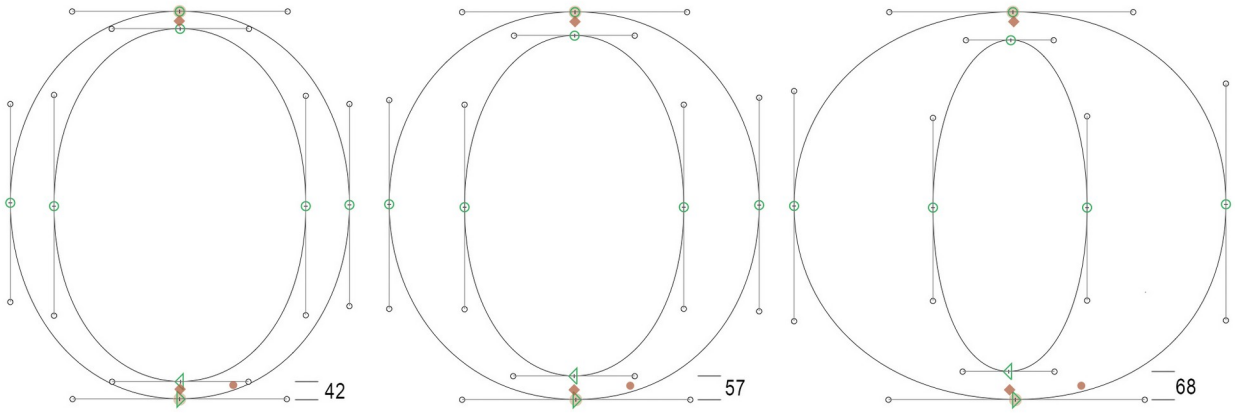
value (or val)
: Required. The name of the control value whose index you want to retrieve. No other kind of value is recognized here.

result-to
: Required. The name of a <variable> in which to store the index of the control value.. Only a variable is permitted here—not, for example, the name of another control value.

<cvar>

Contains information needed to construct a `cvar` table, which variable fonts need in order to take account of the way the distances recorded in the Control Value Table differ along the font's various axes.

The <cvar> element will normally contain at least two <tuple> elements, each one of these described by one or more <region> elements. These <tuple> elements will usually correspond to the font's various masters (but the normal control values record the distances in the default master, which requires no entry in the <cvar> table). For example, suppose your font contains a "weight" axis with a scale running 200–800, and three masters—one for the minimum weight, one for the maximum weight, and one for the regular. Here is the letter o from the three masters, showing the width (in font units) of the bottom curve:



The regular weight, in the middle, will supply the figure in the standard Control Value Table, which you must define whether this is a variable font or not:

```
<control-value name="lc-curved-stem" value="57"/>
```

or

```
<cv nm="lc-curved-stem" val="57"/>
```

The width of the stem in the other masters will be recorded in the <cvar> element:

```
<cvar>
  <tuple>
    <region tag="wght" bot="-1.0" peak="-1.0 top="0.0"/>
    <cvv nm="lc-curved-stem" val="42"/>
  </tuple>
  <tuple>
    <region tag="wght" bot="0.0" peak="1.0 top="1.0"/>
    <cvv nm="lc-curved-stem" val="68"/>
  </tuple>
</cvar>
```

The OpenType specification recommends that the ranges of each region should run from -1.0 to 1.0. Although the weight axis runs from 200 to 800, for the purposes of the <cvar> table use a normalized scale of -1.0 to 1.0. The `bot` and `top` attributes of each <region> specify the points on this axis that define the region: -1.0–0.0 for the first and 0.0–1.0 for the second. The `peak` attribute defines the "peak" point in the region: the point where this master belongs. (Since the masters are at the corners of the design space, the minimum weight master has a `peak` equal to `bot` and the maximum weight master has a `peak` equal to `top`.)

The <cvv> ("Control Value Variant") element is like <control-value> (or <cv>) except for the different name. It is matched to its corresponding control value by name, not by index, so the names must match. There is no need to record any control values that do not differ from the default, and there is no need to keep the <cvv> elements of a <region> in any particular order.

Content

Two or more <tuple> elements.

Attributes

None.

<cvv>

A Control Value Variant belonging to one of the <region>s of the <cvar> table. Except for the element tag, this is exactly like a <control-value> (<cv>). See <cvar> for further details.

Content

None

Attributes

nm
: The name of the control value for which this is a variant. The name here must match that of the <control-value> (or <cv>) exactly.

val
: The value of this variant, i.e. an alternative control value.

<default>

This element, which may appear as a child of <xgridfit>, declares a default value. If the type is "minimum-distance," "control-value-cut-in," "single-width," "single-width-cut-in," "delta-base," "delta-shift" or "round-state," Xgridfit also inserts code in the prep program to set a font-wide default in the TrueType engine. Another way of setting these defaults is simply to include elements that set these values in the <pre-program>, and the effect is the same. If you want the TrueType engine to reject all attempts to set defaults (that is, if you want to use only the defaults that are standard for TrueType), include this:

```
<default type="use-truetype-defaults" value="yes"/>
```

In addition to TrueType defaults, stored in the font file, this element controls several defaults that govern how Xgridfit operates.

Content

None.

Attributes

type

Required. Must be one of the following: “minimum-distance”, “control-value-cut-in”, “single-width”, “single-width-cut-in”, “delta-base”, “delta-shift”, “delta-break”, “max-twilight-points”, “max-storage”, “max-stack”, “use-truetype-defaults”, “round-state”, “function-base”, “init-graphics”, “color”, “assume-always-y”. For details about the meanings of these types, see The Graphics State and Xgridfit Defaults and Merge-mode.

value

Required. For permissible values, see The Graphics State and Xgridfit Defaults.

Additional notes:

The “assume-always-y” default is new and not documented at the old site. Its effect is to alert the compiler that you will be hinting entirely on the y axis (as if you executed `<set-vectors axis="y"/>` at the beginning of a glyph program and stayed with that axis until the end, where you executed `<interpolate-untouched-points axis="y"/>`). When the “assume-always-y” default is set to “yes,” the compiler optimizes certain rounding operations and inserts `<set-vectors axis="y"/>` and `<interpolate-untouched-points axis="y"/>` at the beginning and end of each glyph program, relieving you of a certain amount of tedious typing or cutting and pasting. The default for “assume-always-y” is “no.” To override, add the attribute `assume-y="yes"` or `assume-y="no"` to any `<glyph>` (or `<gl>`) element.

The “cleartype” default is also new: add the line

```
<default type="cleartype" value="yes"/>
```

or the shorter form

```
<default type="cleartype" val="yes"/>
```

to the file to signal that your font is compatible with ClearType. Use of this default is recommended for best performance in Windows.

A new value has been added to the “color” default. Use “auto” to instruct Xgridfit to guess at the correct value for the `color` (or `col`) attribute of the <move> element. At present the algorithm is simple: the color of a <move> is black whenever the <move> either contains a <reference> element (or has an `r` attribute) or is the child of another <move>; otherwise the color is gray. The `color` (or `col`) attribute of a <control-value> (or <cv>) element overrides this, as does the `color` (or `col`) attribute of the current <move>.

<delta>

A delta instruction moves points at particular sizes. The <delta> element may contain any number of <delta-set> elements, each one specifying a point to move, a size at which to move it, and a distance to move it.

The direction of the move is determined by the current setting of the freedom vector. The available specifications are fuzzy as to the details. You will experience no surprises when the vectors are set to x or y; you may experiment with the vectors set at other angles.

When the first element of a <delta> is a <point>, that point is the default, which every <delta-set> element will move unless it contains its own <point>. These two <delta> elements are equivalent:

```
<delta>
  <delta-set size="3" distance="8">
    <point num="p1"/>
  </delta-set>
  <delta-set size="4" distance="8">
    <point num="p1"/>
  </delta-set>
</delta>
<delta>
  <point num="p1"/>
  <delta-set size="3" distance="8"/>
  <delta-set size="4" distance="8"/>
</delta>
```

Note that a <move> element may contain <delta> elements. When a <delta-set> element inside one of these <delta> elements lacks a <point>, it operates on the point moved by the parent <move>.

Content

An optional <point> element, followed by one or more <delta-set> elements.

Attributes

compile-if

Optional. Any value or expression that can be resolved to a number at compile time. If true (non-zero), the <delta> element is compiled; otherwise the compiler passes it over.

<delta-set>

The <delta-set> element encapsulates the essential information about a single delta move or adjustment: the resolution at which to apply the delta, the magnitude of the adjustment, and the point or control value that will be affected.

The resolution is determined by the `size` attribute, which can be a number from 0 to 47. It is added to the value set by the <set-delta-base> or <with-delta-base> instruction to obtain the resolution (in pixels per em or "ppem") at which the move should take place. The default delta base is 9; if you don't change it, a `size` of "0" means 9 ppem, "9" means 18 ppem, and so forth up to "47," which means 56 ppem.

The `distance` attribute is the distance to shift the point along the freedom vector, or the amount to add to or subtract from the control value. Legal values are from -8 to 8 (excluding 0). When moving points, negative numbers shift against the direction of the freedom vector (generally down or left) and positive numbers shift in the direction of the freedom vector.

The default unit by which pixels are moved and control values adjusted is 1/8 pixel. The unit is controlled by means of the <set-delta-shift> or the <with-delta-shift> instruction.

A <delta-set> that is the child of a <delta> will normally contain a single <point>. However, the <point> may be omitted in either of two circumstances: First, when the <delta> is the child of a <move> element, the <point>, when not specified, is implicitly the <point> that is the child of the parent <move>. Second, when the first child element of the <delta> is <point>, that point will be moved by any <delta-set> that lacks a child <point>. A <delta-set> that is the child of a <control-value-delta> element may not contain a <point>, but it must have a `cv` attribute.

All attribute values in a <delta-set> and a child <point> must be capable of being resolved to numerical values at compile time. Variables and function parameters are not permitted.

Content

When the <delta-set> is the child of a <delta> element, it may contain a <point>; this is the point to move. It *must* contain a <point> when it the parent <delta> is not the child of a <move> and the first child of the <delta> is not a <point>. When the <delta-set> is the child of <control-value-delta> it has no content.

Attributes

`cv`

Required when the <delta-set> is the child of <control-value-delta>. The name of the control value to adjust.

size
: Required. An integer from 0 to 47. The resolution at which to move the point or adjust the control value, as explained above.

distance
: Required. An integer from -8 to 8, excluding 0. The distance to move the point or the amount to adjust the control value, as explained above.

<diagonal-stem>

Given two lines (making up a diagonal stem), makes the second line parallel to the first, subject to the operation of the Control Value cut-in. If one <align> element is present, the points it contains are aligned with the second line; if there are two, the first set of points is aligned with the first line and the second set with the second line. You may, and often should, set a new minimum distance value with the `min-distance` attribute. At the end of this instruction the minimum distance is reset to its former value.

Usually it doesn't make a lot of sense to round the distance when calling this instruction; and yet the default value of `round` is `yes` for compatibility with other, similar instructions. You'll probably want to set the `round` attribute to `no`; but if you have several <diagonal-stem> instructions together, enclose them in a `<with-round-state round="no">` element to turn off rounding beforehand and on again afterwards. In this case, do not include the `round` attribute with the <diagonal-stem> elements.

By default this instruction does not set the Freedom Vector, since the best setting of that vector varies with circumstances. If you want the Freedom Vector to be the same as the Projection Vector, set `freedom-vector="yes"`.

This instruction is not suitable for use inside a function (though you may do so if the <line> elements contain points rather than `ref` attributes). Also, I'm not sure whether it will work if the various points are in different zones. It may, but I don't guarantee it.

```
<with-minimum-distance value="diag-min-dist">
  <with-round-state round="no">
    <diagonal-stem distance="cap-thick-diag" save-vectors="yes">
      <line ref="left-diag-left-line"/>
      <line ref="left-diag-right-line"/>
    </diagonal-stem>
    <diagonal-stem distance="cap-thin-diag" save-vectors="yes">
      <line ref="right-diag-right-line"/>
      <line ref="right-diag-left-line"/>
    </diagonal-stem>
  </with-round-state>
</with-minimum-distance>
```

Content

Two <line> elements, the second to be made parallel to the first. Optionally, one or two <align> elements, the first containing points to be aligned with the first line and the second with the second line.

Attributes

distance (or di)
: Required. The name of a control value which determines the distance to place the second line from the first.

round
: Optional. The value may be “yes” or “no”, the name of a standard or custom round state, or any number or expression to use as an input for SROUND. If the value is anything but “no”, rounding is used.

cut-in
: Optional. The value may be “yes” or “no”; the default is “yes”. Determines whether to use the control value cut-in.

min-distance
: Optional. The value may be “yes” or “no” or any value or expression to be used to set the minimum distance for this operation.

color
: Optional. Permitted values are “black”, “white” and “gray”. The default is “black.” The kind of distance between the points of the two <line> elements.

freedom-vector
: Optional. The value may be “yes” or “no”; the default is “no”. Determines whether to set the freedom vector to the same angle as the projection vector.

save-vectors
: Optional. The value may be “yes” or “no”; the default is “no”. If “yes”, both the projection vector and the freedom vector are guaranteed to be the same after this intruction as they were before.

<disable-instructions>

Disables the instructions associated with glyphs. The TrueType specification does not say that instructions in the <pre-program> are disabled: presumably they are not. This instruction is available only in the <pre-program>.

Content

None.

Attributes

None.

<divide>

Divides `dividend` by `divisor`. If `result-to` is not specified, Xgridfit attempts to write the result to `dividend`.

Content

None.

Attributes

dividend
: Required, except when <divide> is a child of the <formula> element. Any value or expression. The number to be divided.

divisor
: Required, except when <divide> is a child of the <formula> element. Any value or expression. The number by which the dividend is to be divided.

result-to
: Optional; not allowed when <divide> is the child of a <formula>. The name of a variable or control value in which to store the result. In a <formula> element the result can be passed automatically to the next arithmetic element; otherwise, if this attribute is missing, the compiler tries to store the result in `dividend`. Failing that, it issues a warning and leaves the result on the stack.

<else>

Provides the "else" clause for an <if> or <compile-if> element. The <else> must be the last child of the parent element.

Content

Programming to be conditionally compiled.

Attributes

None.

<enable-instructions>

Enables the instructions associated with glyphs. This instruction is available only in the <pre-program>.

Content

None.

Attributes

None.

<entry>

An entry in the PostScript private dictionary.

Content

None.

Attributes

name (or nm)
: Required. The name of the entry.

value (or val)
: Required. The value of the entry. Usually this is a space-delimited list of numbers; for BlueFuzz it must be an integer.

<flip-off>
<flip-on>

"Flips" a range of points so that they all become either on-line points or off-line points. The <flip-off> and <flip-on> elements must contain a single <range> element. The range operated upon always runs from the lowest point to the highest.

Content

One <range> element.

Attributes

None.

<floor>

Yields the greatest integer less than or equal to `value`, which is either an integer representing 64ths of a pixel or a fixed-point number. The returned value, if looked at as an integer, is either zero or a multiple of 64.

Content

None.

Attributes

value
: Required, except when <floor> is the child of a <formula>. Any value or expression. The value to operate on.

result-to
: Optional; not allowed when <floor> is the child of a <formula>. The name of a variable or control value in which to store the result. If `result-to` is omitted where allowed, Xgridfit attempts to write the result to `value`. If `value` cannot be written to, the compiler issues a warning and the result is left on the stack.

<formula>

Note: This element is obsolete, though it is still supported. It is better to use expressions (see the last section of this document) where calculations are necessary.

A formula is a block containing arithmetical instruction elements. Within the <formula> the behavior of these elements is modified slightly: when a `result-to` attribute is missing, they do not attempt to write their result back to one of the values passed to them, but rather leave it on the stack, thus making it available to the following instruction element. Further, while Xgridfit normally issues a warning when an arithmetic element takes a value from the stack or leaves a result on the stack, within a formula this is the expected behavior and so the warning is suppressed. The instructions within the formula should be chained, the result of one operation being used as an argument for the following one; this results in tight code being generated. Example:

```
<formula result-to="minimum-distance">
  <round value="lc-vert-stem"/>
```

```
  <multiply value2="0.8"/>
</formula>
```

This rounds the control-value "lc-vert-stem," multiplies it by 0.8, and sets the minimum distance in the graphics state to the result. The original entry in the control-value table is unchanged. By contrast, if this <round> element were not the child of a <formula>, the result would be written back to the control-value table.

Content

A sequence of elements that perform arithmetic: set-equal, add, subtract, divide, multiply, absolute, negate, floor, ceiling, minimum, maximum, round, no-round, control-value-index.

Attributes

result-to

Optional. The name of a variable or control value in which to store the result. If there is no `result-to` attribute, the compiler issues a warning and the result is left on the stack.

<function> (short form <fn>)

The <function> element is used to define functions. Functions are called by name using the <call-function> (or <callf>) element. Xgridfit takes care of indexing functions and making sure the right number is used to call them.

A function that returns a value must have the attribute `return="yes"`; within such a function an instruction may assign a value to the special variable "return". The return value must be a single number; lines, vectors, and other structures consisting of two or more numbers cannot be returned.

A <function> may contain any number of <param> elements and any number of <variable> elements, followed by programming.

```
<function name="lc-standard-serif">
  <!-- Regulates vertical distances within a serif -->
  <param name="base"/>
  <param name="bottom-left"/>
  <param name="bottom-right"/>
  <param name="top-left"/>
  <param name="top-right"/>
  <with-vectors axis="y">
    <mdap>
      <point num="base"/>
    </mdap>
    <mirp distance="lc-serif-height" set-rp0="yes">
      <point num="top-left"/>
```

```
      </mirp>
      <align>
        <point num="top-right"/>
      </align>
      <mirp distance="lc-serif-height" set-rp0="yes">
        <point num="bottom-left"/>
      </mirp>
      <align>
        <point num="bottom-right"/>
      </align>
    </with-vectors>
  </function>
```

You may define function variants, that is, alternative versions of a function to be used at certain sizes or resolutions. To do so, simply include one or more <variant> elements as the last children of the <function> element.

Content

First optional <param> elements; next optional <variable> and <alias> elements (in any order); then elements containing programming to be executed by default; finally optional <variant> elements defining versions of the function to be used if the `test` attribute of the <variant> element evaluates as true (non-zero) when the <pre-program> is run.

Attributes

name (or nm)
: Required. A name: any sequence of letters, numbers, hyphens, periods; no spaces permitted. The name of this function.

xml:id
: Optional. A valid XML ID, i.e. any sequence of letters, numbers, hyphens, periods; no spaces permitted; and unique within the program. This ID may be used if a function is imported into a file using XInclude.

num (or n)
: No longer supported.

return
: Optional. Permitted values are "yes" and "no"; "no" is the default. If "yes", the function may return a value by assigning a value to the special "return" variable.

stack-safe
: Optional. Permitted values are "yes" and "no"; "no" is the default. When "yes," this attribute signals to the compiler that this function doesn't alter the stack (except to pop the parameters that were passed to it) and that it doesn't leave anything on the stack after execution—that is, it doesn't return values on the stack. These two things will always be true if you avoid the low-level elements <push> and <command> when

writing your function. A value of "yes" enables optimizing of the glyph program that calls the function, reducing file size significantly, especially if the function is called often.

<get-coordinate>

Gets the coordinate of a point, measured along the current projection vector. If the projection vector is set to `x`, for example, the result will be the horizontal distance of the point from the grid origin (0,0). The `result-to` attribute may be the name of a variable, a control value, or a graphics state variable.

```
<get-coordinate result-to="v">
  <point num="p"/>
</get-coordinate>
```

Most of the time it will be more convenient to use one of the "coord" operators in an expression.

Content

The <point> to get the coordinate of.

Attributes

grid-fitted
: Optional. Permitted values are "yes" and "no"; the default is "yes". Whether to use the point's current (grid-fitted) position or its original position (before grid-fitting).

result-to
: Optional. The name of a variable or control value in which to store the result. If there is no `result-to` attribute, the compiler issues a warning and the result is left on the stack.

<getinfo>

The <getinfo> element provides access to the TrueType GETINFO instruction, which queries the TrueType engine about the state of the current glyph, about the engine version, or about whether ClearType is enabled. In general it is more convenient to use Xgridfit's built-in graphics variables to obtain such information, but the <getinfo> element may be useful to combine selectors in ways that Xgridfit does not anticipate; and future versions of the TrueType may introduce selectors that Xgridfit does not know about.

<getinfo> takes two attributes: `selector` indicates which query or queries to submit to GETINFO. Its value can be any number or expression; and note that the relevant graphics variables can be used here, and the value of these variables in this context is the selector rather than the result of a GETINFO query, as is the case elsewhere. Selectors for GETINFO can be

combined by ANDing them; this can be simulated in Xgridfit by adding selectors together (since TrueType programming does not have arithmetical AND). This query

```
<getinfo result-to="v" selector="is-glyph-rotated + is-glyph-stretched"/>
```

will return a non-zero (true) value if either condition is true.

The second attribute, `result-to`, works like the same attribute elsewhere: that is, it is a variable in which to store the result of the query, and if it is omitted the compiler issues a warning and the value is left on the top of the stack.

Content

None.

Attributes

selector
: One or more of the following, added together: is-glyph-rotated, is-glyph-stretched, is-glyph-grayscale, is-cleartype-enabled, is-compatible-width-enabled, is-symmetrical-smoothing-enabled, is-BGR-order. Or, by itself, is-glyph-transformed.

result-to
: Optional. The name of a variable in which to store the result. If there is no `result-to` attribute, the compiler issues a warning and the result is left on the stack.

<glyph> (short form <gl>)

The <glyph> element contains instructions relating to an individual glyph. Example:

```
<glyph ps-name="sample-glyph">
  <constant name="last" value="50"/>
  ...
  <variable id="v"/>
  <with-vectors axis="y">
    <!-- programming here -->
  </with-vectors>
  <with-vectors axis="x">
    <!-- more programming here -->
  </with-vectors>
  <interpolate-untouched-points/>
</glyph>
```

To make a glyph program callable in the manner of a macro (see <call-glyph>), include one or more <param> elements for key values. These should have `value` attributes, since the glyph program will have to be compiled and run even when it is not called.

Content

<param> elements (if any) must come first, followed by <constant>, <range>, <set>, <line>, <alias> and <variable> elements in any order. Finally programming for the glyph.

Attributes

ps-name (or pnm)
: Required. The PostScript name of the glyph. It must match the PostScript name of the glyph in the font.

xml:id
: Optional. A valid XML ID, i.e. any sequence of letters, numbers, hyphens, periods; no spaces permitted; and unique within the program. This ID may be used if a <glyph> is imported into a file using XInclude.

init-graphics
: Optional. Permitted values are “yes” and “no”. This attribute determines whether variables used by Xgridfit to track the graphics state are initialized at the beginning of the glyph program. Such initialization is not needed when instructions are imported from another font. The default value is normally “yes,” but the default may be set with the `init-graphics` <default>.

assume-y
: Optional. Overrides any “assume-always-y” default. If “yes,” some rounding operations are optimized, and `<set-vectors axis="y">` and `<interpolate-untouched-points axis="y">` are inserted at the beginning and end of the glyph program. If “no,” you must insert those instructions yourself, where appropriate.

<glyph-select>

No longer supported. Use the --glyphlist or -g command-line argument instead.

<if>

Contains instructions that are executed only if the expression in the `test` attribute is true. If the test is false, and an <else> element is the last child of this <if>, the instructions in the <else> element are executed. <if> elements may be nested.

The `test` attribute contains an expression, e.g.

```
<if test="minimum-distance = 1p">
```

For more on expressions, see the Expressions section of this documentation. In the TrueType language, where all data is numeric, a non-zero value is considered true and zero false. So the `test` attribute can be used to tell whether a value is zero. This

```
test="round(lc-vert-stem) != 0"
```

is equivalent to this:

```
test="round(lc-vert-stem)"
```

The <if> element may contain an <else> clause, which must come last.

```
<if test="boolean-expression">
  <!-- programming -->
  <else>
    <!-- alternative programming -->
  </else>
</if>
```

Content

Programming, followed optionally by an <else> element.

Attributes

test
: Required. Any value or expression. The condition to test for.

<inputfont>

Contains the name of the TrueType font to which instructions are to be added.

```
<inputfont>MyFont.ttf</inputfont>
<outputfont>MyFont-hinted.ttf</outputfont>
```

Content

The name of a file with extension `.sfd` or `.ttf`.

Attributes

None.

<interpolate> (short form <ip>)

To “interpolate” a point is to move it so that its position relative to two reference points is what it was in the original outline. If the distance between the two reference points is not what it was in the original outline, the point is positioned so that its relationship to the reference points is proportionally correct.

The <interpolate> element must contain at least one point to interpolate. It may contain any number of <point>s, <range>s and <set>s. Like most other elements that move points, it may contain a <reference> element; but this element must contain two <point>s, not just one.

This instruction may be nested inside a <move> element containing a reference point, in which case no reference points are needed in the <interpolate> element: the points it contains are automatically interpolated between the <move> element's reference point and its moved point. These two <interpolate> elements do the same thing:

```
<interpolate>
  <reference>
    <point num="top"/>
    <point num="bottom"/>
  </reference>
  <point num="bar-top-left"/>
</interpolate>

<move>
  <reference>
    <point num="bottom"/>
  </reference>
  <point num="top"/>
  <interpolate>
    <point num="bar-top-left"/>
  </interpolate>
</move>
```

If you want to round points after interpolating them, simply include the attribute `round="yes"` on the <interpolate> element. The points will be moved along the freedom vector to the nearest rounded position. Note that this has no effect on <range> or <set> elements. If you want to use a round state other than the current one, use the appropriate value for `round`:

```
<interpolate round="to-half-grid">
  <reference>
    <point num="a"/>
    <point num="b"/>
  </reference>
  <point num="c"/>
</interpolate>
```

Content

An optional <reference> element (containing two <point> elements), followed by one or more <point>, <range> and <set> elements.

Attributes

compile-if
: Optional. Any value or expression that can be resolved to a number at compile time. This <interpolate> element is compiled only if the `compile-if` attribute evaluates as true (non-zero). If this attribute is not present, the <interpolate> is compiled.

round
: Optional. The value may be “yes” or “no”, the name of a standard or custom round state, or any number or expression to use as an input for SROUND. If the value is anything but “no”, any points referred to by <point> elements (but not <range> or <set> elements) are moved to the nearest rounded position. Points are not rounded when this attribute is not present.

<interpolate-untouched-points> (short form <iup>)

Interpolates all points that have not been moved or “touched” by instructions so that they are positioned correctly relative to points that have been moved. Most of the time you will want to place this instruction at the end of the program for each glyph.

Content

None.

Attribute

axis
: Optional. Permitted values are “x” and “y”. Determines the axis along which interpolation is performed. If this attribute is omitted, interpolation is performed along both x and y axes.

<legacy-functions>

No longer supported.

<line>

A <line> is defined by its two end-points. These points need not be adjacent.

When a <line> has a `name` attribute, another <line> may refer to it by name. You may name the <line> the first time you use it or declare it in a <line> element among the declarations at the beginning of a <glyph>. For example, if you declare the line thus:

```
<line name="line-2b">
  <point num="b"/>
  <point num="a"/>
</line>
```

then you can use an abbreviated form whenever you need it:

```
<set-freedom-vector>
  <line ref="line-2b"/>
</set-freedom-vector>
```

If both points that define a <line> are in the same zone, you may use the optional `zone` attribute on the <line> to indicate this. Most instructions that take a <line> as an argument allow one point to be in one zone and the other point in the other: in such cases place the `zone` attributes on the <point>s. The exception is the <move-point-to-intersection> instruction, which requires that each of the two lines it takes as arguments be entirely in a zone. For this instruction, place the `zone` attributes on the <line>s, never on the <point>s.

Content

Two <point> elements, one at each end of the line. These are required if no `ref` attribute is present.

Attributes

name (or nm)
: Optional, but necessary if this <line> is referred to elsewhere. A name: any sequence of letters, numbers, hyphens, periods; no spaces permitted. Names this <line>.

ref
: Required if no <point> elements are present. A reference to a named <line>.

zone
: Optional. Permitted values are “twilight” and “glyph”. Include the attribute `zone="twilight”` if both end-points of the line are in the twilight zone.

<macro> (short form <mo>)

A macro is a stretch of code that is compiled and inserted into the program stream wherever a <call-macro> element is encountered. Macros resemble functions in syntax: they begin with <param> definitions followed by program code, which can access all global variables and values (e.g. control values, graphics variables, constants and variables declared as children of <xgridfit>). Macros can be called from a <function>, <pre-program> or <glyph>, and variables and values local to these structures can be accessed by passing them as parameters.

Macro parameters are fundamentally different from function parameters. A function parameter is passed to the function on the stack at run time; the macro parameter, on the other hand, is swapped at compile time for the `value` parameter, which is then evaluated as any value is evaluated at run time. When passing variables or control values, there is no rule governing whether they are passed to the macro by value or by reference; rather, they are evaluated according to the rule that governs evaluation of variables for the particular instruction where they occur.

Content

Optional <param> elements, followed by optional <alias> elements, and then by programming.

Attributes

name (or nm)
: Required. A name: any sequence of letters, numbers, hyphens, periods; no spaces permitted. Names this <macro>.

xml:id
: Optional. A valid XML ID, i.e. any sequence of letters, numbers, hyphens, periods; no spaces permitted; and unique within the program. This ID may be used if a <macro> is imported into a file using XInclude.

<maximum>

Yields the greater of the two values `value1` and `value2`.

```
<maximum value1="lc-vert-stem" value2="1p" result-to="lc-vert-stem"/>
```

Content

None.

Attributes

value1
: Required except when <maximum> is the child of a <formula> element. Any value or expression.

value2
: Required except when <maximum> is the child of a <formula> element. Any value or expression.

result-to
: Optional; not allowed when <maximum> is the child of a <formula>. The name of a variable in which to store the result. If `result-to` is omitted where allowed, the compiler issues a warning and the result is left on the stack.

<mdap>

Corresponds to the TrueType MDAP instruction. Normally <move> is a better choice for moving points, but this element is provided to facilitate low-level programming.

This element rounds a point to the grid if the `round` attribute is not `no`; otherwise it "touches" the point (marks it as moved).

```
<mdap>
  <point num="p"/>
</mdap>
```

Content

A single <point>.

Attributes

round
: Optional. The value may be "yes" or "no", the name of a standard or custom round state, or any number or expression to use as an input for SROUND. The default value is "yes." This attribute determines whether and how to round the point.

<mdrp>

Corresponds to the TrueType MDRP instruction. Normally <move> is a better choice for moving points, but this element is provided to facilitate low-level programming.

This element will also take care of setting RP0 beforehand if a reference point is supplied. If no reference point is supplied, the current value of RP0 is used.

```
<mdrp>
  <reference>
    <point num="r"/>
  </reference>
  <point num="p"/>
</mdrp>
```

Content

An optional <reference> element (containing one point), followed by a single <point>.

Attributes

round
: Optional. The value may be “yes” or “no”, the name of a standard or custom round state, or any number or expression to use as an input for SROUND. The default value is “yes.” This attribute determines whether and how to round the distance between the reference point and the moved point.

min-distance
: Optional. Permitted values are “yes” and “no”. The default is “yes.” Whether or not to maintain a minimum distance between the reference point and the moved point.

set-rp0
: Optional. Permitted values are “yes” and “no”. The default is “no.” Whether or not to set the RP0 pointer to the moved point after the instruction is executed.

color
: Optional. Permitted values are “black”, “white” and “gray”. The default is “gray.” The kind of distance between the reference point and the moved point. This is used by some TrueType engines to compensate for engine characteristics.

<measure-distance>

Measures the distance between two points and returns an F26dot6 number. The measurement is performed along the projection vector, and the result may be positive or negative depending on the relationship of the distance to the direction of the vector. Another way of putting it is that the order of the two points in this instruction is significant; reverse them and you reverse the sign of the result.

```
<measure-distance result-to="v">
  <point num="p1"/>
  <point num="p2"/>
</measure-distance>
```

Usually it will be more convenient to use the `--` or `---` operator in an expression.

Content

Two <point> elements.

Attributes

grid-fitted
: Optional. Permitted values are “yes” and “no”; the default is “yes”. Whether to use the point's current (grid-fitted) position or its original position (before grid-fitting).

result-to
: Optional. The name of a variable or control value in which to store the result. If there is no `result-to` attribute, the compiler issues a warning and the result is left on the stack.

<message>

The <message> element produces no TrueType code, but rather causes a message to be output at compile-time via the <xsl:message> element. This may help with debugging.

Content

Text of a message to be output.

Attributes

None.

<miap>

Corresponds to the MIAP instruction. Normally <move> is a better choice for moving points, but this element is provided to facilitate low-level programming.

Positions a point a specified distance from the grid origin.

Content

A single point.

Attributes

distance
: Required. The name of a control value.

round
: Optional. The value may be “yes” or “no”, the name of a standard or custom round state, or any number or expression to use as an input for SROUND. The default value is “yes.” This attribute determines whether and how to round the distance between the reference point and the moved point.

cut-in
: Optional. The value may be “yes” or “no”. The default is “yes.” Whether to use the control value cut-in. If rounding is used, this will always be `yes`, even if you set it to `no`.

<minimum>

Yields the lesser of the two values `value1` and `value2`.

```
<minimum value1="lc-vert-stem” value2="1p” result-to="lc-vert-stem"/>
```

Content

None.

Attributes

value1
: Required except when <minimum> is the child of a <formula> element. Any value or expression.

value2
: Required except when <minimum> is the child of a <formula> element. Any value or expression.

result-to
: Optional; not allowed when <minimum> is the child of a <formula>. The name of a variable in which to store the result. If `result-to` is omitted where allowed, the compiler issues a warning and the result is left on the stack.

<mirp>

Corresponds to the MIRP instruction, but attempts, insofar as it is practical, to separate rounding and the cvt cut-in. You can specify `round=“no”` and `cut-in=“yes”` or both `no` or both `yes`, but not `round=“yes”` and `cut-in=“no”`. That produces an error-message.

This will take care of setting RP0 beforehand if a reference point is included.

```
<mirp distance="lc-x-height">
  <reference>
    <point num="p1"/>
  </reference>
  <point num="p2"/>
</mirp>
```

Content

An optional <reference> element (containing one point), followed by a single <point>.

Attributes

distance (or di)
: Required. The name of a control value. Distance (from a <control-value> element) relative to the reference point (or to RP0 if that was set by a previous instruction).

round
: Optional. The value may be “yes” or “no”, the name of a standard or custom round state, or any number or expression to use as an input for SROUND. The default value is “yes.” This attribute determines whether and how to round the distance between the reference point and the moved point.

cut-in
: Optional. The value may be “yes” or “no”. The default is “yes.” Whether to use the control value cut-in. If rounding is used, this will always be `yes`, even if you set it to `no`.

min-distance
: Optional. Permitted values are “yes” and “no”. The default is “yes.” Whether or not to maintain a minimum distance between the reference point and the moved point.

set-rp0
: Optional. Permitted values are “yes” and “no”. The default is “no.” Whether or not to set the RP0 pointer to the moved point after the instruction is executed.

color
: Optional. Permitted values are “black”, “white” and “gray”. The default is “gray.” The kind of distance between the reference point and the moved point. This is used by some TrueType engines to compensate for engine characteristics.

<modifier>

When used as the content of a <command> element, controls one or two bits of the output instruction.

Content

None.

Attributes

type
: Required. Must be one of the following: “set-rp0”, “round”, “minimum-distance”, “color”, “grid-fitted”, “to-line”, “axis”, “ref-ptr”. This determines which bits are affected.

value
: Required. The possible values vary with the `type`. For details and defaults, see the “Low-Level Elements” section.

<move> (short form <mv>)

Moves a point and, optionally, aligns other points with it or moves other points in relation to it. For details, see the section on moving points.

Content

An optional <reference> (or <ref>) element (or, in compact syntax, an `r` attribute) containing one <point> (<pt>); then a required <point>. When you are using compact syntax, you may use a `p` attribute instead of <point> elements. These are equivalent:

```
<mv>
  <pt n="5"/>
</mv>

<mv p="5"/>
```

The <move> element can contain any number of the following elements, in this order:

- <delta>
- <align> (<al>), <interpolate> (<ip>), <shift> (<sh>), in any order,
- <move>
- <delta>

The first group of <delta> elements is executed before the embedded <align>, <interpolate>, <shift> and <move> elements; the second group is executed afterwards. A <move> may contain <interpolate> only if <reference> is present, or if the <move> element is the child of another <move>.

Attributes

distance (or di)

Optional. The name of a <control-value> element. If a `distance` is specified, the target point is positioned that distance either from the reference point or from the grid origin. If a `distance` is not specified, the distance from the original outline is used. In either case, the `distance` is measured along the projection vector.

pixel-distance

Optional. A distance in pixels. If a `pixel-distance` is specified, the target point is positioned that distance either from the reference point or from the grid origin. The `distance` and `pixel-distance` attributes are not compatible, and the schema does not permit both to be present.

round

Optional. The value may be "yes" or "no", the name of a standard or custom round state, or any number or expression to use as an input for SROUND. The default is "yes." Whether and how to round the `distance` or `pixel-distance`. "yes" means round the distance according to the current round state (to-grid, if you haven't changed it). If you specify "no", no rounding is done. To use one of the standard round states, use `to-grid, to-half-grid, to-double-grid, down-to-grid` or `up-to-grid`. To use a custom round state defined in the top level of the program (as a child of <xgridfit>), use its name. Finally, any number (constant, variable) is passed to SROUND for the TrueType engine to interpret. Setting the round state with this attribute has no effect except in this instruction: the round state returns to its former value after the instruction is executed. If several <move> instructions use the same round state, it is more efficient to enclose them in a <with-round-state> element than to include a `round` attribute with each one. That is also true if the `round` value is to be `no`: in that case use `<with-round-state round="no">` and omit the `round` attribute for the <move> instructions.

cut-in

Whether to use the Control Value cut-in; or a cut-in value to use. Legal values are "yes", "no" or any value or expression; the default value is "yes". If the value of this attribute is `no`, the value of the `round` attribute must also be "no". (This is a peculiarity of the TrueType instruction set and has nothing to do with Xgridfit.) This attribute has an effect only when the `distance` attribute is present.

min-distance

Optional. The value may be "yes" or "no" or any value or expression to be used to set the minimum distance for this operation. This attribute has an effect only when there is a reference point.

color (or col)

Optional. Permitted values are "black", "white" and "gray". The default is "gray." The kind of distance between the reference point and the moved point. This is used by some

TrueType engines to compensate for engine characteristics. This applies only when there is a reference point or when this <move> is the child of another <move>. When this attribute is missing, the color associated with a control value will be used, and failing that, the default color (gray if you don't change it via a <default> element).

r

A substitute for the <reference> element, available only when you are using compact syntax.

p

A substitute for the <point> element, available only when you are using compact syntax.

<move-point-to-intersection>

Moves a point to the intersection of two lines. Each of the lines must be wholly in a single zone, so if specifying the zone use the `zone` attribute of the <line> elements rather than the `zone` attributes of the <point> elements that make up the lines.

```
<move-point-to-intersection>
  <point num="p"/>
  <line>
    <point num="l1p1"/>
    <point num="l1p2"/>
  </line>
  <line>
    <point num="l2p1"/>
    <point num="l2p2"/>
  </line>
</move-point-to-intersection>
```

Content

A single <point> and two <line> elements.

Attributes

None.

<multiply>

Multiplies `value1` by `value2`. If there is no `result-to` attribute, Xgridfit attempts to write the result to `value1`.

```
<multiply value1="lc-vert-stem" value2="3.3" result-to="v"/>
```

Content

None.

Attributes

value1
: Required except when <multiply> is the child of a <formula> element. Any value or expression.

value2
: Required except when <multiply> is the child of a <formula> element. Any value or expression.

result-to
: Optional; not allowed when <multiply> is the child of a <formula>. The name of a variable in which to store the result. If `result-to` is omitted where allowed, the compiler issues a warning and the result is left on the stack.

<negate>

Converts positive to negative numbers; negative numbers stay negative. If the `result-to` attribute is not present, Xgridfit attempts to write the result back to `value`. Failing that, it issues a warning and leaves the result on the stack.

```
<negate value="v"/>
```

Content

None.

Attributes

value
: Required, except when <negate> is the child of a <formula>. Any value or expression. The value to operate on.

result-to
: Optional; not allowed when <negate> is the child of a <formula>. The name of a variable or control value in which to store the result. If `result-to` is omitted where allowed, and `value` is a variable or control value, the result is written to `value`. If `value` cannot be written to, the compiler issues a warning and the result is left on the stack.

<no-compile>

This element, which must always be a child of <xgridfit>, contains <glyph> elements (usually imported via XInclude) which are visible for reference purposes but not compiled. If a file `MyFont-Basic.xgf` contains a glyph program with an opening tag that looks like this:

```
<glyph ps-name="a" xml:id="a">
```

then that glyph program can be made visible within another file thus:

```
<no-compile>
  <xi:include href="MyFont-Basic.xgf#a"/>
</no-compile>
```

Now the following <point> will compile correctly:

```
<point num="a/top + another-num"/>
```

The glyph included in this way can also be compiled via <call-glyph>:

```
<call-glyph ps-name="a">
  <with-param name="left-sidebearing" value="111"/>
</call-glyph>
```

The <no-compile> element can be overridden with the <glyph-select> element (`glyph-select` parameter or `-g` option). This behavior may ease the testing of programs for composite glyphs.

Content

One or more <glyph> elements; more typically, <xi:include> elements importing <glyph> elements from other files.

Attributes

None.

<no-round>

Like round, but without the rounding. That is, it may apply a correction for the "color" of the distance, but it will not round the distance. If the `result-to` attribute is not present, Xgridfit attempts to write the result back to `value`. Failing that, it issues a warning and leaves the result on the stack.

```
<no-round value="v1" color="black" result-to="v2"/>
```

Content

None.

Attributes

value
: Required, except when <no-round> is the child of a <formula>. Any value or expression. A distance on the current grid; the value to operate on.

color
: Optional. Permitted values are “black”, “white” and “gray”. The default is “gray.” The kind of distance that the `value` represents. This is used by some TrueType engines to compensate for engine characteristics.

result-to
: Optional; not allowed when <no-round> is the child of a <formula>. The name of a variable or control value in which to store the result. If `result-to` is omitted where allowed, and `value` is a variable or control value, the result is written to `value`. If `value` cannot be written to, the compiler issues a warning and the result is left on the stack.

<no-warning>

Inside a <no-warning></no-warning> block, warning messages are suppressed. Use this if you find a particular warning message annoying and you want to assure the compiler that you know what you're doing.

Content

Programming of any kind.

Attributes

None.

<outputfont>

Contains the name of a TrueType font to be written by the Python script output by Xgridfit.

```
<outputfont>MyFont-hinted.ttf</outputfont>
```

Content

The filename of a TrueType font.

<outfile-base>

No longer supported.

<outfile-script-name>

No longer supported.

<param> (short form <pm>)

A <param> element is a declaration that a value, a structure or some code may be passed to the <function>, <macro> or <glyph> that contains it by the "call" element that calls it. The <param> elements should be the first children of <function>, <macro> and <glyph> elements.

In the case of a <function>, only a value (a single number) may be passed. A <macro> or <glyph> is much more flexible: <line>, <range>, <set> and fragments of code may be passed. Within the <macro> or <glyph>, the structures can be referenced via `ref` attributes; code passed as a parameter can be called via a <call-param> element.

A <param> may contain a default `value`—that is, a value to be used in the event that the call element does not pass a value. The <param> may also contain code to be used when the call element does not pass code; but it may not contain a <line>, <range> or <set>.

Content

Normally this element is empty; but a <param> intended to pass code to a <macro> or <glyph> may contain code to be used when the <call-macro> or <call-glyph> element lacks a matching <with-param> element.

Attributes

name (or nm)
: Required. A name: any sequence of letters, numbers, hyphens, periods; no spaces permitted. The name of the <param> is used both by the call element and by the code that requires the value or item passed.

value (or val)
: Optional. A default value.

<param-set> (short form <pmset>)

Contains a set of parameters (each encoded as a <with-param> element) to be passed to a function or macro. There may be more than one of these in a <call-function> or <call-macro> element; and in that case the function is called repeatedly (via LOOPCALL) until the list of <param-set> elements is exhausted; or the macro code is compiled and inserted repeatedly.

Content

One or more <with-param> (<wpm>) elements; or, in compact syntax, a list of key-value pairs, as with <call-function> (<callf>) or <call-macro> (<callm>).

Attribute

opt

Optional. If set to "yes," forces optimization in pushing function parameters onto the stack. That is, instructs Xgridfit to push all parameters with a single PUSHB command. Use this if you are sure that all parameters can be determined by the compiler (i.e. they are not determined at run-time) and that they are all between 0 and 255, but Xgridfit is not optimizing. If you get this wrong, that is, if you force optimization when it really ought not to be used, incorrect code will be generated and your glyph program will fail. Still, it may be worth a try since the failure will probably be obvious. Conversely, set this to "no" if Xgridfit is incorrectly optimizing the parameters in a function call. This probably won't happen, since Xgridfit optimizes rather conservatively, but it's here just in case.

<point> (short form <pt>)

The <point> element defines a point. It is used in all instructions that manipulate or refer to points. In a <mv>, <ip>, <sh>, or <al> element (short forms of <move>, <interpolate>, <shift>, and <align>), and also in <srp>, <set>, and <range>, you may include a single p attribute instead of several <pt> elements, e.g. `p="base 12 (top - 1)"`. instead of

```
<pt n="base"/>
<pt n="12"/>
<pt n="top - 1"/>
```

This attribute is a space-delimited set of values (each of them anything you can put in the n attribute of a <pt> element): remember that if any expression here contains spaces, it must be enclosed in parentheses. Note also that you cannot specify a zone with the attribute notation.

Content

None.

Attributes

num (or n)

Required. Any value or expression. The number of a point. To refer to a point in a glyph other than the one whose glyph program is currently running (as you may have occasion to do when instructing composite glyphs), use the syntax “g/p”, where g is the ps-name of the other glyph, and p is the point being referred to (it must be the name of a <constant> declared in the other glyph).

zone

Optional. Permitted values are “twilight” and “glyph”. The zone that contains this point. Instructions will take note of this attribute, when present, and adjust the zone pointers appropriately. The glyph zone is always the default zone. When a point is in the glyph zone it is generally redundant to include an attribute `zone="glyph"`, and doing so may also cause unnecessary (though harmless) code to be generated. N.B. For instructions that deal with lists of points, include the zone attribute only in the first.

<pre-program> (short form <prep>)

The prep table (or CVT program) is made from the <pre-program> element. It contains instructions that are executed before a font is rasterized, or whenever it is about to be rasterized in a new size. A typical thing to do in the <pre-program> is to adjust control values, e.g. rounding or applying deltas to them. Another is to set defaults: an instruction that assigns a value to a graphic variable sets a default when it is executed in the <pre-program>. For example, if executed in the <pre-program> this instruction:

```
<set-minimum-distance value="0.9">
```

ensures that the minimum-distance graphics variable is always 0.9 pixels at the beginning of any glyph program.

The <pre-program> element must be present in a complete Xgridfit program, even if it is empty.

Content

One or more optional <variable> and <alias> elements, followed by optional programming.

Attribute

xml:id

Optional. A valid XML ID, i.e. any sequence of letters, numbers, hyphens, periods; no spaces permitted; and unique within the program. This ID may be used if the <pre-program> is imported into a file using XInclude.

<ps-private>

No longer supported.

<push>

The <push> element does the work of the various PUSHB and PUSHW instructions, and it can generate the code to move any value (e.g. a variable or control-value) onto the stack. It may be used in combination with <command> to insert low-level TrueType commands into your Xgridfit programming in a portable way. Its content is a whitespace-delmited list of numbers, identifiers and expressions. These are valid Xgridfit <push> instructions:

```
<push>2 5 89 67</push>
```

```
<push>
  left
  right
  lc-vertical-stem
  -1
</push>
```

```
<push> 0.58p 2.0 to-grid </push>
```

```
<push>1 (top + 3) 512</push>
```

It is essential that all expressions containing whitespace be enclosed in parentheses.

Content

A space-delimited list of values and expressions.

Attributes

None.

<range>

A <range> is a collection of contiguous points defined by its end-points. It can be used in any instruction that operates on more than one point: <shift>, <align>, <interpolate>, <shift-absolute>, <toggle-points>. The order of points in the <range> is not significant. Example:

```
<align>
  <reference>
    <point num="bottom"/>
  </reference>
  <range>
    <point num="bottom - 2"/>
    <point num="bottom + 2"/>
  </range>
</align>
```

If "bottom" is point 17, the <range> begins with 15 and ends with 19. But any reference point in the parent element of the <range> is excluded from the <range>, so this <range> actually represents points 15, 16, 18 and 19. The same is true of implicit reference points supplied by a <move> element that is the parent of the parent of the <range>:

```
<move>
  <point num="bottom"/>
  <align>
    <range>
      <point num="bottom - 2"/>
      <point num="bottom + 2"/>
    </range>
  </align>
</move>
```

Here the points in the <range> are aligned with "bottom" after it has been moved by the <move> instruction; but "bottom" itself is not part of the <range>. The code above is functionally identical to this:

```
<move>
  <point num="bottom"/>
  <align>
    <point num="bottom - 2"/>
    <point num="bottom - 1"/>
    <point num="bottom + 1"/>
    <point num="bottom + 2"/>
  </align>
</move>
```

The latter generates more efficient code than the example with the <range>, but the <range> is more flexible, since its endpoints, its size, and the points to be excluded need not be known until run-time. This makes the <range> ideal for use in functions.

All the points in a <range> must be in the same zone, determined by the optional zone attribute on the <range> element. Any zone attributes on the <point>s within the <range> are ignored.

Content

Two <point> elements. These are required if no `ref` attribute is present.

Attributes

name (or nm)
: Optional, but necessary if this <range> is referred to elsewhere. A name: any sequence of letters, numbers, hyphens, periods; no spaces permitted. Names this <range>.

ref
: Required if no <point> elements are present. A reference to a named <range>.

zone
: Optional. Permitted values are “twilight” and “glyph”.

<restore-default>

To reset any of the graphics variables tracked by Xgridfit to its default value (that is, the value set by your Xgridfit program via a <default> element or by setting it in the <pre-program>--failing that, the TrueType default), use the <restore-default> element. Use the value `all` to reset *all* graphics variables, including the freedom and projection vectors and the auto-flip state, which Xgridfit does not track. The `all` option sets these last to their TrueType defaults: the x axis for the vectors and “on” for auto-flip.

```
<restore-default name="minimum-distance"/>
```

Content

None.

Attribute

name (or nm)
: The name of the graphics variable to restore to its default value. Possible values are “minimum-distance”, “control-value-cut-in”, “single-width”, “single-width-cut-in”, “delta-base”, “delta-shift”, “round-state”, “all”.

<reference> (short form <ref>)

In any instruction that can position a point or other structure with reference to one or more points, the <reference> element holds the reference point(s). This element generally contains precisely one point, but when it is the child of an <interpolate> element it must contain two points. In the following example, point p2 is moved relative to point p1:

```
<move>
  <reference>
    <point num="p1"/>
  </reference>
  <point num="p2"/>
</move>
```

Content

A single <point>. When the <reference> element is the child of <interpolate>, two points.

Attributes

None.

<region>

Defines a region of an axis and the point in that region belonging to an interpolated outline (usually at an extreme of the axis). For details, see <cvar>

Content

None.

Attributes

tag
: Required. The tag that identifies the axis (e.g. “wght,” “opsz”).

bot
: Required. The bottom of the region.

top
: Required. The top of the region.

peak
: Required. The defined point in the region (usually equal to `top` or `bot`).

<round>

Rounds a number representing a distance according to the current round state, applies whatever correction is appropriate for the “color” of the distance, and returns the result.

```
<round value="lc-vert-stem” color="black” result-to="new-cvt"/>
```

Content

None

Attributes

value
: Required, except when the child of <formula>. Any value or expression, understood as a distance on the current grid. This is the number to round.

color
: Optional. Permitted values are “black”, “white” and “gray”. The default is “gray.” The kind of distance that the `value` represents. This is used by some TrueType engines to compensate for engine characteristics.

result-to
: Optional; not allowed when <round> is the child of a <formula>. The name of a variable or control value in which to store the result. If `result-to` is omitted where allowed, and `value` is a variable or control value, the result is written to `value`. If `value` cannot be written to, the compiler issues a warning and the result is left on the stack.

<round-state>

Declares a custom round state whose name can be passed to <set-round-state>, <with-round-state>, or any element that takes a `round` attribute. For an explanation of the `period`, `phase` and `threshold` attributes, see “Rounding.” Note that only a limited number of values is permitted for each of these attributes. This element is permitted only at the top level of a program, as a child of <xgridfit>.

```
<round-state name="my-round” period="two-pixel"
  phase="three-quarters"
  threshold="three-quarters"/>
```

Content

None.

Attributes

name (or nm)
: Required. A name: any sequence of letters, numbers, hyphens, periods; no spaces permitted. This is the name under which the program may refer to this round state.

period
: Required. One of the following: “half-pixel”, “one-pixel”, “two-pixel”.

phase
: Required. One of the following: “zero”, “one-quarter”, “one-half”, “three-quarters”.

threshold
: Required. One of the following: “period-minus-one”, “minus-three-eighths”, “minus-one-quarter”, “minus-one-eighth”, “zero”, “one-eighth”, “one-quarter”, “three-eighths”, “one-half”, “five-eighths”, “three-quarters”, “seven-eighths”, “one”, “nine-eighths”, “five-quarters”, “eleven-eighths”.

<set>

A <set> is an arbitrary collection of points, defined by the <point> elements contained in the <set> element. The <set> can be used by any element that accepts a collection of points: <align>, <interpolate>, <shift>, <shift-absolute>, <toggle-points>.

A <set> can be used only in a <glyph> program, or it can be referenced in a <macro> called by a <glyph> program. The name of a <set> can be passed to a macro as a parameter, and a <set> can be the content of a <with-param> element.

Xgridfit must be able to resolve the num attributes of all <point>s in a <set> at compile time. It is an error to attempt to reference a variable in a <set>.

When a <set> is used in an element that has reference points, either explicitly via the <reference> element or implicitly via an enclosing <move> element, any reference points repeated in the set are excluded. This works only when the <reference> points can be resolved at compile time.

A <set> may be preferable to a <range> when all point numbers are known at compile time and the range is short, including perhaps three or four points. The code generated by Xgridfit on encountering a <set> is less flexible, but vastly more efficient than that generated on encountering a <range>.

A <set> may be defined thus among the declarations at the beginning of a <glyph>:

```
<set name="bar-bottom-left-corner">
  <point num="bar-bottom-left"/>
  <point num="bar-bottom-left + 1"/>
  <point num="bar-bottom-left + 2"/>
</set>
<set name="bar-bottom-right-corner">
  <point num="bar-bottom-right"/>
  <point num="bar-bottom-right - 1"/>
  <point num="bar-bottom-right - 2"/>
</set>
```

It can then be referenced whenever needed:

```
<move distance="cap-horz-stem">
  <reference>
```

```
    <point num="bar-top-left"/>
  </reference>
  <point num="bar-bottom-left"/>
  <shift>
    <set ref="bar-bottom-left-corner"/>
    <set ref="bar-bottom-right-corner"/>
  </shift>
</move>
```

Content

One or more <point> elements. These are required if no `ref` element is present. If using compact syntax, use either <pt> elements or a `p` attribute.

Attributes

name (or nm)
: Optional, but necessary if this <set> is referred to elsewhere. A name: any sequence of letters, numbers, hyphens, periods; no spaces permitted. Names this <set>.

p
: In compact syntax, a space-delimited list of the points in the set. Remember that any expression containing spaces (e.g. `top - 2`) must be enclosed in parentheses.

ref
: Required if no <point> elements are present. A reference to a named <set>.

zone
: Optional. Permitted values are “twilight” and “glyph”.

<set-auto-flip>

When “on” (the default setting), the TrueType engine automatically adjusts the signs of control values when executing MIRP and MIAP instructions. This works very well, so there is rarely a reason to set this to “off.”

Content

None.

Attribute

value
: Required. Permitted values are “on” and “off”.

<set-control-value> (short form <setcv>)
<with-control-value> (short form <wcv>)

You can assign a value to a control value anywhere: in the <pre-program>, a <function>, or a <glyph> program. The value you assign can be either in font units (the units of the grid on which you designed the font) or in pixel units (the grid on which the glyph is now being rasterized). To specify which, include the attribute `unit="font"` or `unit="pixel"` ("font" is the default). You must specify the name of the control value with the `name` attribute and the value (an integer in font units or an "F26Dot6" number in pixel units) with the `value` attribute.

You can use <set-control-value> to make a control value simulate a local variable.

Use <with-control-value>, which takes attributes exactly like those of <set-control-value>, to assign a control value to be used only within the <with-control-value> element. After this element, the value will be the same as it was before.

```
<set-control-value name="myval" unit="font" value="850"/>
<set-control-value name="otherval" unit="pixel"
      value="control-value(otherval) * 2"/>
<with-projection-vector axis="y">
  <set-control-value name="myval" unit="pixel"
      value="point(a) --- point(b)"/>
</with-projection-vector>
```

Content

None for <set-control-value>; <with-control-value> contains programming of any kind.

Attributes

name (or nm)
: Required. The name of the control value to set.

value
: Required. Any value or expression. The value to write to the control value.

unit
: Optional. Permitted values are "font" and "pixel"; the default is "font". Whether the value is in font units or pixel units.

<set-control-value-cut-in>
<with-control-value-cut-in>

The `value` is a distance on the grid. If the difference between a distance from a <control-value> element and the original distance is greater than this, the original distance is used. The effect is

generally to use the <control-value> distance at low resolutions and the original distance at high resolutions. This can be used to promote evenness at small sizes, where a 1-pixel difference between the width of (say) p and b can look bad. The default value is 17/16: that is, 1.0625p or 68.

```
<set-control-value-cut-in value="1.1"/>
```

Content

<set-control-value-cut-in> has no content; <with-control-value-cut-in> contains programming.

Attribute

value
: Required. Any value or expression. This is the new control value cut-in.

<set-coordinate>

Moves a point to a coordinate determined by the freedom and projection vectors. On the rare occasions when you need a command like this one, it is probably better to use <move> with the `pixel-distance` attribute.

Content

A <point> to move.

Attribute

coordinate
: Required. Any value or expression. This is the new coordinate of the point.

<set-delta-base>
<with-delta-base>

Sets the number that is added to the “size” attribute of a <delta-set> element to get the resolution at which an adjustment should take place. The default value is 9, and that rarely needs to be changed.

```
<with-delta-base value="56">
  <delta>
    <delta-set size="12” distance="-8">
```

```
        <point num="p"/>
      </delta-set>
    </delta>
  </with-delta-base>
```

Content

<set-delta-base> has no content; <with-delta-base> contains programming.

Attribute

value
: Required. Any value or expression. This is the new delta base.

<set-delta-shift> <with-delta-shift>

The unit by which a delta instruction shifts a point. If the unit is "2," the smallest shift is half a pixel; if "4," it is a quarter of a pixel; if "8" (the default) it is one eighth, and so on.

```
<set-delta-shift value="32"/>
```

Content

<set-delta-shift> has no content; <with-delta-shift> contains programming.

Attribute

units-per-pixel
: Required. These values are permitted: "2", "4", "8", "16", "32", "64".

<set-dropout-control>

Sets up dropout control. The `threshold` is a number between 0 and 254 in pixels per em. The `flags` attribute is a number that tells how to set up dropout control relative to the threshold. Here are the flags as explained in the Apple TrueType Reference:

- *1. Set dropout control to TRUE if other conditions do not block and ppem is less than or equal to the threshold value.*
- *2. Set dropout control to TRUE if other conditions do not block and the glyph is rotated.*
- *4. Set dropout control to TRUE if other conditions do not block and the glyph is stretched.*

- *8. Set dropout control to FALSE unless ppem is less than or equal to the threshold value.*
- *16. Set dropout control to FALSE unless the glyph is rotated.*
- *32. Set dropout control to FALSE unless the glyph is stretched.*

Note that, for the sake of simplicity, the numbers used here are different from those implied in the Apple TrueType Reference. Xgridfit adjusts them before passing them to the TrueType engine.

To turn off dropout control, set both `threshold` and `flags` to zero.

Content

None.

Attributes

threshold
: Required.

flags
: Required.

<set-dropout-type>

Sets dropout type. The Microsoft TrueType Reference describes the action of the possible values as follows:

- *if n=0 rules 1 and 2, and 3 are invoked (dropout control scan conversion including stubs)*
- *if n=1 rules 1, 2 and 4 are invoked (dropout control scan conversion excluding stubs)*
- *if n=2 rules 1 and 2 only are invoked (fast scan conversion; dropout control turned off)*
- *if n=3 same as n = 2*
- *if n = 4 rules 1, 2, and 5 are invoked (smart dropout control scan conversion including stubs)*
- *if n = 5 rules 1, 2, and 6 are invoked (smart dropout control scan conversion excluding stubs)*
- *if n = 6 same as n = 2*
- *if n = 7 same as n = 2*

The scan conversion rules are shown here:

Rule 1
: If a pixel's center falls within the glyph outline, that pixel is turned on.

Rule 2

If a contour falls exactly on a pixel's center, that pixel is turned on.

Rule 3

If a scan line between two adjacent pixel centers (either vertical or horizontal) is intersected by both an on-Transition contour and an off-Transition contour and neither of the pixels was already turned on by rules 1 and 2, turn on the left-most pixel (horizontal scan line) or the bottom-most pixel (vertical scan line). This is "Simple" dropout control.

Rule 4

Apply Rule 3 only if the two contours continue to intersect other scan lines in both directions. That is, do not turn on pixels for 'stubs.' The scanline segments that form a square with the intersected scan line segment are examined to verify that they are intersected by two contours. It is possible that these could be different contours than the ones intersecting the dropout scan line segment. This is very unlikely but may have to be controlled with grid-fitting in some exotic glyphs.

Rule 5

If a scan line between two adjacent pixel centers (either vertical or horizontal) is intersected by both an on-Transition contour and an off-Transition contour and neither of the pixels was already turned on by rules 1 and 2, turn on the pixel which is closer to the midpoint between the on-Transition contour and off-Transition contour. This is "Smart" dropout control.

Rule 6

Apply Rule 5 only if the two contours continue to intersect other scan lines in both directions. That is, do not turn on pixels for 'stubs.'

Content

None.

Attribute

value

Required. Possible values are from "0" to "7," with meanings as described above.

<set-dual-projection-vector>

Like <set-projection-vector>, but the dual projection vector can be set only from a line, and it uses the original positions in the outline of the points that constitute the line rather than their current positions (assuming they have moved).

The dual projection vector is not used by every instruction: just by <interpolate>, <get-coordinate>, <measure-distance>, <mirp>, <mdrp>, and <move> (only when a "relative-to"

point is present). This vector lasts only until a new projection vector is set; then it gets canceled.

One or both points in the line may be in the twilight zone. See the explanation for <set-vectors>.

```
<with-projection-vector>
  <set-dual-projection-vector to-line="orthogonal">
    <line ref="line-a"/>
  </set-dual-projection-vector>
  <!-- programming here; then the end of the with-projection-vector
       block returns the dual projection vector to its former value. -->
</with-projection-vector>
```

Content

One <line>.

Attributes

to-line

Optional. Possible values are “orthogonal” and “parallel”; the default is “parallel”. Determines whether the dual projection vector will be orthogonal or parallel to the <line> from which it is set.

<set-equal>

Set `target` (variable, control value, or any of the graphics state variables that Xgridfit can write to) equal to `source`, which can be an expression or any number type that Xgridfit can handle.

```
<!-- This is the equivalent of a = b in Python. -->
<set-equal target="a” source="b"/>
```

Content

None.

Attributes

source

Required. Any value or expression.

target

Required. The name of a variable, control value or graphics variable to write to.

<set-freedom-vector>
<with-freedom-vector>

Just like <set-vectors>, but sets only the freedom vector.

<set-minimum-distance>
<with-minimum-distance>

The minimum-distance property is used by several instructions when the “min-distance” attribute is “yes.” The default minimum distance is one pixel (1.0, 1p, 64), but can be set to another value here.

```
<with-minimum-distance value="0.85">
  <!-- The distance between p1 and p2 must
       be at least 0.85 pixel. -->
  <move round=“no”>
    <reference>
      <point num="p1"/>
    </reference>
    <point num="p2"/>
  </move>
</with-minimum-distance>
```

Content

<set-minimum-distance> has no content; <with-minimum-distance> contains programming.

Attributes

value

Required. Any value or expression. This is the new minimum distance.

<set-projection-vector>
<with-projection-vector>

Just like <set-vectors>, but sets only the projection vector.

<set-round-state>
<with-round-state>

Sets the round state. If the round attribute matches the name of a <round-state>, that round state is used. If not, one of TrueType's prefabricated round states may be used:

- to-grid
- to-half-grid
- to-double-grid
- up-to-grid
- down-to-grid

If the round attribute is not one of these, and not the name of one of the custom round-states, Xgridfit tries to resolve it as a number, constant, variable or function parameter and use that as an argument to SROUND. You had better know what you're doing if you intend to use a raw number in this way; it is safer, more intelligible and just as effective to supply a custom <round-state> element.

The distinction between the element beginning with “set” and the one beginning with “with” is the same as it is for the vector-setting elements: briefly, the round state set by the “set” element affects the instructions that follow it; the round state set by the “with” instruction affects only the instructions that it contains.

Xgridfit generates instructions that keep track of the round state (since the TrueType engine provides no way to read it), but it may lose track if Xgridfit instructions are not used exclusively.

```
<with-round-state round="to-half-grid">
  <move>
    <point num="v-point"/>
  </move>
</with-round-state>
```

Content

<set-round-state> has no content; <with-round-state> contains programming.

Attributes

round

Required. The name or a standard or custom round state; any value or expression. This is the new round state.

<set-single-width>
<with-single-width>

The size of the single width, in FUnits, i.e. the units of the grid the font was designed on (usually 2048 or 1000 units per em). Presumably this width is converted to the current grid, and it is that converted value that the single-width cut-in is compared to.

Content

<set-single-width> has no content; <with-single-width> contains programming.

Attributes

value

Required. Any value or expression, interpreted as font units. This is the new single width.

<set-single-width-cut-in>
<with-single-width-cut-in>

When the <mirp> or <mdrp> instruction is used, or when <move> is used relative to a point, a single width (determined by <set-single-width> or <with-single-width>) may be used rather than a control value or the original distance if this condition is met: the absolute (either positive or negative) difference between the original outline and the single width is less than the single-width cut-in. The relevant distances are in pixels.

The single width feature appears to be used rarely.

Content

<set-single-width-cut-in> has no content; <with-single-width-cut-in> contains programming.

Attributes

value

Required. Any value or expression. This is the new single-width cut-in.

<set-vectors> (short form <setvs>)
<with-vectors> (short form <wvs>)

Sets both the projection vector and the freedom vector to the same value. They can be set to "x" or "y" via the `axis` attribute; to a line by including a line element as the content of the <set-vectors> element or the first child of the <with-vectors> element; or by passing "raw" values via the `x-component` and `y-component` attributes.

Xgridfit looks first for an `axis` attribute, next for a <line>, and finally for `x-component` and `y-component` attributes (neither is used unless both are present). If it finds none of these and the present element is <with-vectors>, Xgridfit simply stores the present vectors on the stack and restores them at the end of the block. If the present element is <set-vectors>, Xgridfit prints a warning and attempts to find "raw" vector values on the stack.

The "raw" values passed in via `x-component` and `y-component` are constrained in ways that make them difficult to calculate, at least in a TrueType program, but the `x-component/y-component` method is useful to restore values that have been saved via <store-projection-vector> or <store-freedom-vector>. For example, to copy one vector to another, you can do this:

```
<variable name="x-comp"/>
<variable name="y-comp"/>
<store-freedom-vector x-component="x-comp" y-component="y-comp"/>
<set-projection-vector x-component="x-comp" y-component="y-comp"/>
```

But because of the way these instructions can leave values on the stack and take them from the stack again, this is easier and more efficient:

```
<no-warning>
  <store-freedom-vector/>
  <set-projection-vector/>
</no-warning>
```

When setting vectors to a line, one or both points in the line can be in the twilight zone. You can include a `zone` attribute in the <line> element or one in either or both <point> elements. Include a `zone` attribute in the <line> element if both points are in the twilight zone. This is the same as including an attribute `zone="twilight"` in both points. If only one point is in the twilight zone, include the `zone` attribute for that point.

Here are several examples:

```
<with-vectors axis="x">
  <!-- programming that moves points horizontally. -->
</with-vectors>

<with-vectors to-line="orthogonal">
  <line ref="diagonal-line"/>
  <!-- programming that moves points along a line orthogonal to
       diagonal-line. -->
```

```
</with-vectors>

<set-vectors to-line="parallel">
  <line ref="diagonal-line"/>
</set-vectors>
<!-- Subsequent programming will move points along a line
     parallel to diagonal-line. -->
```

Content

<set-vectors> has no content if an `axis` attribute or the `x-coordinate` and `y-coordinate` attributes are present; otherwise it may contain a <line> element. <with-vectors> works the same way, but also contains programming.

Attributes

axis
: Optional, and incompatible with other attributes. Possible values are “x” and “y”.

to-line
: Optional, and permitted only when a <line> is present. Possible values are “orthogonal” and “parallel”; the default is “parallel”. Determines whether the vectors will be orthogonal or parallel to the <line> from which they are set.

x-component, y-component
: Optional, but if one of these attributes is present, the other must be as well. Not permitted with other attributes or when a <line> is present.

<shift> (short form <sh>)

Shifts one or more points, ranges, sets, contours and zones by the distance between the current position of the reference point and its original position. Note that this does not guarantee that the shifted elements will maintain their original distance from the reference point (use <move> or <mdrp> for that).

The <shift> element may contain points, ranges, sets, contours and zones in any combination and order. The following is perfectly correct:

```
<shift>
  <reference>
    <point num="ref-pt"/>
  </reference>
  <point num="move-pt-1"/>
  <range ref="move-rg-1"/>
  <contour num="0"/>
  <point num="move-pt-2"/>
  <range ref="move-rg-2"/>
</shift>
```

But note that all the points are shifted first, then all the ranges or sets, then all the contours, and finally any zones. The order of child elements in the <shift> element is not significant.

If you want to move points to the nearest rounded position after the shift, include a round attribute. This works exactly like the round attribute on the <interpolate> element.

Content

An optional <reference> (or <ref>) element, followed by any number of <point>, <range>, <set>, <contour> and <zone> elements, in any order. In compact syntax, you may include an r attribute instead of the <reference> element and a p element instead of a collection of points.

Attributes

compile-if
: Optional. If present, the <shift> element is compiled only if this attribute evaluates as true (non-zero) at compile time.

round
: Optional. Possible values are "yes", "no", one of the standard or custom round states, or any value or expression yielding a number to pass to SROUND. The default value is "no", since by default no rounding is done on the <point>s contained in a <shift> element.

reference-ptr
: Optional. Possible values are "1" and "2". This determines which reference pointer (RP1 or RP2) to use. Normally Xgridfit decides which pointer is appropriate in the context; otherwise RP1 is used.

r
: Optional. The reference point, used in compact syntax instead of the <ref> element.

p
: Optional. In compact syntax, a space-delimited list of points to move, used instead of a collection of <pt> elements.

<shift-absolute>

Moves one or more points along the freedom vector by a fixed amount (expressed in pixels); it does not use the projection vector. The <shift-absolute> element must contain at least one point to shift: that is, a <range>, <set> or <point> element. It may contain any number of <point>s, <range>s and <set>s.

Content

Any combination of <range>, <set> and <point> elements in any order.

Attributes

pixel-distance
: Required. Any value or expression, understood as a distance on the grid.

<srp>

Does the work of SRP0, SRP1, SRP2. But it should rarely be necessary to set the reference pointers explicitly.

Content

One <point> element; the reference pointer is set to point to this.

Attribute

whichpointer
: Required. The reference pointer to set. Possible values are “0”, “1” and “2”.

<store-projection-vector>
<store-freedom-vector>

These instructions store a vector as two numbers, an x-component and a y-component. The `x-component` and `y-component` attributes, if given, must be identifiers for variables:

```
<store-projection-vector x-component="vx” y-component="vy"/>
```

If these attributes are not given, a warning is printed and the values are left on the stack, where they will be picked up correctly by a following `set` instruction. For example, this code sets the projection vector to be the same as the freedom vector:

```
<store-freedom-vector/>
<set-projection-vector/>
```

Note that a `with` block will not pick up the components of a vector from the stack.

Content

None.

Attributes

x-component, y-component
: Optional, but if one attribute is present the other must be as well. The names of variables in which to store the components of the vector. If these attributes are not present, the compiler displays a warning and the values are left on the stack.

<subtract>

Subtracts `minuend - subtrahend`. If `result-to` is not specified, Xgridfit attempts to write the result to `minuend`.

Content

None.

Attributes

minuend
: Required, except when <subtract> is the child of a <formula>. Any value or expression. The value to subtract from.

subtrahend
: Required, except when <subtract> is the child of a <formula>. Any value or expression. The value to subtract from the minuend.

result-to
: Optional; not allowed when <subtract> is the child of a <formula>. The name of a variable of control value in which to store the result of this operation. If `result-to` is omitted where allowed, and `minuend` is a variable or control value, the result is written to `minuend`. If `minuend` cannot be written to, the compiler issues a warning and the result is left on the stack.

<szp>

Does the work of SZP0, SZP1, SZP2. But these should rarely be needed (use the zone attributes of the point element instead).

Content

None.

Attributes

zone
: Required. Possible values are “twilight” and “glyph”. The zone to set the zone pointer to.

whichpointer
: Required. Possible values are “0”, “1” and “2”. Identifies the pointer to set.

<toggle-points>

Any of the points that are on-line become off-line, and any that are off-line become on-line. The <toggle-points> element must contain at least one point to toggle: that is, a <range>, <set> or <point> element. It may contain any number of <point>s, <set>s and <range>s.

Content

One or more <point>, <set> and <range> elements.

Attributes

None.

<to-stack>

The <to-stack> element moves a single value onto the stack. This can be any kind of value or expression, e.g. a number literal, variable, control value or graphics variable. Use <push> instead when more than one value needs to be placed on the stack.

Content

A single value or expression.

Attributes

None.

<tuple>

The top-level child of the <cvar> element, describing variations from the control-value table for one or more extremes of a variable font’s axes. See <cvar> for details.

Content

One or more <region> elements, followed by one or more <cvv> elements.

Attributes

None.

<untouch>

A point that has been moved is "touched." This untouches it so that it will be affected by the <interpolate-untouched-points> instruction.

Content

One <point> element.

Attributes

None.

<variable> (short form <var>)

Variables are spaces in the TrueType Storage Area. They are declared in <variable> elements; Xgridfit takes care of indexing the Storage Area.

```
<variable name="var-name"/>
```

Most variables are local to a glyph program, function or pre-program. Variable declarations may come among the declarations at the beginning of a <glyph> program; after <param> elements in a <function>, and at the beginning of the <pre-program>. A global variable may be declared anywhere in the top level of the program, as a child of <xgridfit> (it is good form to group variable declarations together); a value may be assigned to a global variable in the <pre-program> and read by code in any <glyph> or <function> or elsewhere in the <pre-program>.

Variables must be written to before they can be read from. (Some versions of Freetype initialize them to zero, but the Microsoft rasterizer yields an error if a variable is read before it is written.) Local variables may be initialized with a `value` attribute in the declaration.

A variable is named via its `name` attribute. This must be unique in the file in the case of a global variable, but in the case of local variables unique only in the <glyph> program or <function>.

Several names are reserved and should be avoided when naming variables and constants, since they belong to pre-defined variables and constants:

- Current size (all read-only):
    - pixels-per-em
    - point-size
- Values of the round-state variable (constants):
    - to-grid
    - to-half-grid
    - to-double-grid
    - down-to-grid
    - up-to-grid
    - no
    - custom
- Graphics variables:

Writing to these changes the graphics state. The “default” variables can be written to only in the <pre-program>. Trying to write to them elsewhere produces a compile error. Actually, it is never necessary to write to a “default” variable since writing to one of the other variables in the <pre-program> automatically writes to the “default” variable as well. So treat the “default” variables as read-only variables.

    - round-state (can be written to only via <set-round-state> and <with-round-state>)
    - custom-round-state (can be written to only via <set-round-state> and <with-round-state>)
    - minimum-distance
    - minimum-distance-default
    - control-value-cut-in
    - control-value-cut-in-default
    - single-width
    - single-width-default
    - single-width-cut-in

- single-width-cut-in-default
- delta-base delta-base-default
- delta-shift delta-shift-default

Content

None.

Attributes

name (or nm)
: Required. A name: any sequence of letters, numbers, hyphens, periods; no spaces permitted. The name of this variable.

value
: Optional. Any value or expression. A value with which to initialize the variable.

<variant>

A <variant> is an alternative version of a function to be used at certain sizes or resolutions. Include one or more <variant> elements as the last children of any <function> element.

Content

Programming. <param> and <variable> elements are not permitted; the <variant> must use those of the parent <function>.

Attribute

test
: Required. Any value or expression. The variant is used if this attribute evaluates as true (non-zero) when the <pre-program> is run.

<with-control-value>

See <set-control-value>

<with-control-value-cut-in>

See <set-control-value-cut-in>

<with-delta-base>

See <set-delta-base>

<with-delta-shift>

See <set-delta-shift>

<with-freedom-vector>

See <set-freedom-vector>

<with-minimum-distance>

See <set-minimum-distance>

<with-param> (short form <wpm>)

Defines a value to be passed to a function, macro or glyph program. The `value` may be any of the value-types that Xgridfit handles. Note that all values are resolved to numbers before a call to a function takes place: Xgridfit does not pass parameters to functions by reference. A result of this is that if a variable or control value is passed to a function, these things cannot be written to.

In general there will be a <with-param> element for every <param> element that appears in the function or macro being called. However, a <with-param> element may be omitted if the matching <param> contains a default `value`.

The order of <with-param> elements in a <param-set>, <call-function>, <call-macro> or <call-glyph> element is not significant.

If the <with-param> element is part of a call to a macro, the value passed can be a <set>, <range> or <line>. The `name` of such a structure can be passed via the `value` attribute, or the structure itself can be passed as the child of <with-param>. When <with-param> contains a <set>, <range> or <line>, the `value` attribute is optional.

Programming can also be passed via a <with-param> element, and in this case too the `value` attribute is optional.

Content

Usually none, but if the call is to a <macro> or <glyph>, the element can contain a <range>, <set> or <line>, or a fragment of programming.

Attributes

name (or nm)
: Required. A name: any sequence of letters, numbers, hyphens, periods; no spaces permitted. This must match the name of <param> in the <function>, <macro> or <glyph> being called.

value
: Required except when content is present. Any value or expression. The value to pass as a parameter.

<with-projection-vector>

See <set-projection-vector>

<with-round-state>

See <set-round-state>

<with-single-width>

See <set-single-width>

<with-single-width-cut-in>

See <set-single-width-cut-in>

<with-vectors>

See <set-vectors>

<xgridfit>

The root element of an Xgridfit program file.

Namespace declaration

This element must contain the namespace declaration xmlns="http://xgridfit.sourceforge.net/Xgridfit2".

Content

One at most of each of these: <glyph-select>, <infile>, <outfile>, <outfile-base>, <outfile-script-name>, <no-compile>, <pre-program>, <cvar>. Any number of these: <constant>, <alias>, <variable>, <round-state>, <default>, <control-value>, <function>, <macro>, <glyph>, <xi.include>.

Attribute

xml:id
: Optional. A valid XML ID, i.e. any sequence of letters, numbers, hyphens, periods; no spaces permitted; and unique within the program. This ID may be used if the <xgridfit> element is imported into a file using XInclude.

<zone>

A zone to be shifted by a <shift> instruction.

Content

None.

Attribute

zone
: Required. Must be “twilight” or “glyph”.

Expressions

Expressions
Formulas

The TrueType engine is capable of performing simple arithmetical operations, and it is easy to combine these to perform more complex operations. You must always remember, however, that TrueType arithmetic operates on F26dot6 fixed-point numbers, and all operations return the same kind of fixed-point numbers. There is no way to obtain results with higher precision.

Thus certain kinds of calculations are impossible in TrueType, and you must always be careful, when combining the operations that TrueType can do, to consider the limited precision of the intermediate results that get passed from operation to operation.

Expressions in Xgridfit resemble expressions in other programming languages: they consist of numbers and identifiers coordinated with operators; they can be simple (e.g. “bottom-pt + 1”) or complex; Xgridfit parses them according to certain rules of precedence which are worth knowing; and the rules of precedence may be overridden by using parentheses, which can be nested.

There is little point in describing the syntax of expressions in detail, since they are so familiar to everyone who has done any programming; instead this section will list the operators, note a few peculiarities, and present some examples .

Operators	
First precedence	
and	Logical and. **Example:** `pixels-per-em > 10 and pixels-per-em < 20`
or	Logical or. **Example:** `pixels-per-em < 10 or pixels-per-em > 20`
Second precedence	
=	Equals. **Example:** `pixels-per-em = 15`
>	Greater than. **Example:** `control-value(lc-vert-stem) > 1p`
<	Less than. **Example:** `control-value(lc-vert-stem) < 1p`
>=	Greater than or equal. **Example** (where `v` has previously been declared as a variable): `v >= 0.35`
<=	Less than or equal. **Example:** `v <= 0.35`
!=	Not equal. **Example:** `round(control-value(left-side)) != control-value(left-side)`
Third precedence	
+	Addition. **Example:** `top-point + 1`
-	Subtraction. **Example:** `top-point - 1`
*	Multiplication. **Example:** `lc-vert-stem * 1.2`
/	Division. **Example:** `lc-vert-stem / 2.0`
Fourth precedence	
--	Treats the arguments on both sides of the operator as point numbers and returns

Operators	
	the current distance (in pixels) between them, as measured on the projection vector. Ordinarily the argument on the left-hand side of the operator should be the point on the left or bottom; reverse the numbers to change the sign of the result. **Example:** `round(stem-left -- stem-right)`
`---`	Like `--`, but returns the distance between points in the original outline. **Example:** `absolute((stem-left --- stem-right) - (stem-left -- stem-right))`
Fifth precedence	
odd	True if the argument is odd. **Example:** `odd(v)`
even	True if the argument is even. **Example:** `even(v)`
not	Reverses the boolean value of the argument. **Example:** `not(v > 4.0)`
floor	The greatest integer value less than the argument. **Example:** `floor(control-value(lc-vert-stem))`
ceiling	The smallest integer value greater than the argument. **Example:** `ceiling(control-value(lc-vert-stem) / 2)`
absolute	The absolute value of the argument. **Example:** `absolute(control-value(lc-vert-stem))`
negative	The negation of the argument. **Example:** `negative(v)`
round	The argument rounded according to the current round state. The "color" is the <default> set with `type="color"` (gray if not set). **Example:** `round(control-value(lc-vert-stem))`
round-gray	The argument rounded according to the current round state. The "color" is gray. **Example:** `round-gray(control-value(lc-vert-stem))`
round-black	The argument rounded according to the current round state. The "color" is black. **Example:** `round-black(control-value(lc-vert-stem))`
round-white	The argument rounded according to the current round state. The "color" is white. **Example:** `round-white(control-value(lc-vert-stem))`
index	Returns an index of (pointer to) a control value or variable. **Example:** `index(lc-vert-stem)`
control-value	The control value at the index represented by the argument. **Example:** `control-value(lc-vert-stem)`
coord	The current coordinate (x or y depending on the projection vector) of a point. **Example:** `coord(stem-left)`
initial-	Like `coord`, but returns the original coordinate of a point: that is, its coordinate at

Operators	
coord	the beginning of the glyph program. **Example:** `initial-coord(bottom-point + 1)`
x-coord	Like `coord`, but always returns the current coordinate of a point on the x axis. The setting of the projection vector is the same after the operation as before. `coord` is more efficient than this and the following three operators when the coordinate of a point on the current projection vector is needed.
y-coord	Like `coord`, but always returns the current coordinate of a point on the y axis. The setting of the projection vector is the same after the operation as before.
initial-x-coord	Like `initial-coord`, but always returns the original coordinate of a point on the x axis. The setting of the projection vector is the same after the operation as before.
initial-y-coord	Like `initial-coord`, but always returns the original coordinate of a point on the y axis. The setting of the projection vector is the same after the operation as before.
variable	Treats the argument as an index of (pointer to) a variable and returns its value. **Example:** `variable(v)`
nan	Returns true (1) if the argument cannot be resolved to a number at compile-time (e.g. it is the name of a variable, or the name of a <range>). If the argument is a number, returns false (0). **Example:** `nan(v)`
point	Causes the compiler to regard the argument as a point number. In a <glyph> program with an `offset` parameter, the `offset` is automatically added to the argument. Use this operator in contexts where the compiler does not automatically recognize a number as a point number. Do not use it in the `num` attribute of a <point> element, as this will cause the `offset` to be added twice. **Example:** `point(a) -- point(b)`

Here is an example of precedence:

```
<if test="pixels-per-em &lt; 10 or pixels-per-em &gt; 20 and
          round-state = to-grid">
```

Xgridfit breaks the expression at the “or” (which has the same precedence as the “and” but occurs farther to the left); it evaluates first “`pixels-per-em < 10`”, second “`pixels-per-em > 20 and round-state = to-grid`”, and finally executes OR on the two values. If that is not what you want, you may use parentheses to alter the order in which constituents are evaluated:

```
<if test="(pixels-per-em &lt; 10 or pixels-per-em &gt; 20) and
          round-state = to-grid">
```

Now Xgridfit breaks the expression at the “and” and evaluates everything to the left of it (inside the parentheses), then everything to the right of it, and finally executes AND.

Fifth-precedence operators are all unary: they operate on a single value. If this is a simple value it may be separated from the operator by a space; if it is an expression it must be enclosed in parentheses.

Binary operators (those that operate on two values) must always be surrounded by whitespace. This will not work:

```
<point num="top+2"/>
```

It must be like this:

```
<point num="top + 2"/>
```

However, the whitespace may be any number of spaces, tabs, a line break, and so on, for the spacing of an expression is always normalized before it is evaluated.

Note that when all of the values in an expression are number literals, constants or other identifiers that can be resolved to numbers at compile time, and the only operators are first- or second-precedence operators and the arithmetic operators "+" and "-", Xgridfit resolves the whole expression to a single number at compile time. This optimizes the most common cases, where a point number is expressed by addition to or subtraction from a constant.