Enabling Typography: towards a general model of OpenType Layout

John Hudson, Tiro Typeworks Ltd., 15 April 2014, revision 1.2 (change log)

This discussion paper on OpenType Layout follows from my earlier document Problems for Indic typography in current OpenType Layout implementations. *This paper seeks to expand the observations and recommendations in that document towards a general model for enabling typography for any script and language (presuming an existing Unicode character encoding). This paper is intended to be of use primarily for shaping engine and application makers.*

Background

When the OpenType font format and Layout model were introduced by Microsoft and Adobe in the second half of the 1990s, there was a perceptible difference in the priorities of the two companies: respectively, complex script shaping and typographic design support. This found expression in the initial set of OpenType Layout (OTL) features registered by each company, and in the priorities adopted in the implementation of OTL in each's software. So, for example, Microsoft registered features and implemented support for Arabic and Indic script shaping, but was slow to implement support for typographic features for European scripts, such as ligatures and smallcaps, while Adobe registered and supported features for many typographic refinements for European and East Asian scripts, but was slow to enable complex script shaping or even some kinds of OTL lookup types.

Within their respective areas of priority, each company established de facto standards for feature implementation via their software, in the absence of any formal specification of OTL behaviour beyond the font format documentation and supplementary, script-specific shaping specifications produced by Microsoft. Unfortunately, these implementations tended to perpetuate the initial difference in priorities between script shaping and typography, such that very different levels of typographic sophistication now pertain to different writing systems, affecting the ability of publishers, typographic designers, and font makers to produce consistent quality in their work across multiple scripts and languages.

In this paper—which is intended to provoke discussion and action, not to provide a fully realised solution to all issues—, I will attempt to suggest a path towards better integration of script and language shaping with full typographic sophistication, via a general model of OpenType Layout. Such a model is complicated in some aspects by the existing feature set, which has been registered and supported in an ad hoc manner without a unified guiding principle such as this general model seeks, belatedly, to apply. It is easy to conceive of entirely fresh sets of features and layout behaviours based on a strict set of architectural principles—*e.g.* by making a systematic distinction between required, default and discretionary settings for any feature—but that is not what this paper seeks to do. For better or for worse, we have an existing set of features, significant investment in their implementation, and existing documents in need of ongoing support. It is from this position that I begin.

What is typography, and how do we get to it?

In its broadest sense, I understand typography to be the articulation of text through the design and arrangement of type. This means that typography is concerned with all aspects of text, from the shape of individual characters and their immediate interaction within words, up to the design of whole documents, publication series, or dynamic text presentation systems. In this paper, I am primarily concerned with what is sometimes referred to as microtypography.[1] At this level, typography is concerned primarily with the relationships of visually adjacent typeforms; note, however, that due to required reordering, the presence of combining marks, or the nature of the shapes involved, these may not be sequentially adjacent in either the encoded character string or the glyph run). It is this level of typography that is—or should be—directly addressable via OpenType Layout, either automatically in the case of required or default features, or as directed by the typographer.

The results of typography are integrated: all the parts contributing together to the reader's experience of the text. The process of typography, though, is dis-integrated. Not only does the process of displaying text involve multiple, more-or-less distinct stages, but in today's digital typography these stages are handled by several different technologies, with collective responsibility for the results shared by operating systems, layout engines, applications and fonts.[2] Optimally, these technologies work together and support each other; when this fails, though, they may actively work against each other, leading to attempts to work around limitations or bugs in one part of the system elsewhere. This, predictably, results in incompatibilities, requiring yet further workarounds.

OpenType Layout, it has to be said, is not a tidy technology. It requires that three separate pieces of software, often made by three separate entities, work seamlessly together, and that they do so without any formal specification beyond that of the font format. There has never been documentation that tells a shaping engine or application developer how to implement OTL or how, through that implementation, to enable typography. Instead, there is a mix of script-specific shaping documentation for some complex writing systems, and the de facto behaviour norms established by particular shaping engine implementations and fonts, reverse-engineered by other parties with an eye on compatible outcomes rather than identical procedures. One result of this is that problems in particular layout engines or fonts will proliferate without correction to others, especially if they are seen as de facto standards. Another result is that software developers responsible for supporting specific writing systems, referring to the particular and partial script shaping documentation, will

1. I sometimes use the term subatomic typography, in the context of an analogy in which words are understood to be the atoms of text: the basic distinguishable unit of semantic content. Phrases and sentences constitute various kinds of simple and complex molecules, making up the compounds of paragraphs, chapters, and the organisms of whole documents. Subatomic typography, then, is that which concerns itself with what happens within the word, at the level of the interaction of typeforms. In OTL terms, this is the arena of glyph processing.

2. All technologies for typesetting text have involved some 'division of labour'. Handset metal type, of the kind introduced in Europe in the mid-15th Century, located most of the responsibility in the hands of the typesetter, and most of the intelligence of the system in his brain. Even in that earliest typographic technology, though, default spacing intelligence was located in the font, embodied in the widths of the metal sorts and, hence, the responsibility of the type founder rather than the typesetter.

tend to ignore those aspects of OTL beyond that documentation, overlooking the fact that script and language shaping is not an end in itself: *it is preparation for typography.*

As discussed in my earlier document, the present state of OTL for Indic writing systems presents an unfortunate example of both these results. Focus on the orthographic cluster as the necessary unit of script and language shaping has resulted in failure to enable interaction between the clusters: interaction that is necessary to crucial aspects of micro-typography including substitution of appropriate contextual forms and spacing adjustments to avoid gaps or collisions between adjacent shapes. Failure of Microsoft's Uniscribe layout engine to apply cross-cluster lookups has resulted in other shaping engine makers including the same limitation in their own software, or has put them under pressure to make their software compatible with the outcomes of Uniscribe, even if their own outcomes are typographically preferable.

In order to enable typography to a consistent level for all scripts and languages, a number of things are desirable. Perhaps most challenging, a typographic mentality needs to be inculcated in all areas of software development affecting the display of text, such that program managers, developers and testers recognise that the purpose of Open*Type* Layout is typography. The existing implementations of OTL, especially at the script-specific layout engine level, need to be reviewed and behaviour that limits or prevents typography needs to be identified and fixed, *even if this means changes to behaviour that might affect backwards compatibility.*[3] And a more systematic and better documented approach to future OTL implementation needs to be adopted, especially for newly supported scripts and languages so that problems that have afflicted development so far are not repeated.

I believe all of these initiatives, and others, would benefit from greater collaboration between interested parties, including through standards organisations and open sourcing. There was a period, during the early years of OpenType, and during development of core support for major writing systems and languages in operating systems and applications, when it made sense for companies to treat text processing and display as an area of competition. I believe that time has past. The focus of new text processing is increasingly on minority and historical scripts and languages, for which competitive business cases will be harder and harder to make. The inconsistencies and incompatibilities in levels of support and behaviours that have resulted from two decades of competition in this area are sources of frustration to users and, it must be said, to font developers.

Outline of a general model

The specification of a general model for OpenType Layout, applicable to all writing systems, would have been made easier by consistent categorisation of each registered feature with regard to its rôle in orthographic shaping and typographic layout (and hence what limitations might apply to the feature with regard to orthographic units of a given writing system). Instead, some features apply to orthographic unit shaping for some writing systems and to typographic layout for others. This

3. Backwards compatibility issues can be handled in a variety of ways, and should be examined in terms of policies for managing backwards support, not as restrictions on forward development. In the contect of OTL we have already encountered such issues, arising from changes in Indic script shaping models between Windows XP and Windows Vista, and these were managed through the introduction of new script tags in the OpenType registry. At worst, tertiary script tags might be required in some instances to resolve current problems in shaping models. Some backwards compatibility issues might simply be accepted, especially insofar as they result from obvious improvements.

is notably the case for what I call the topographical features—<isol> <init> etc.—, which not only have varying rôles for different writing systems but also apply to varying elements of the text; these features are discussed at length later in this paper.

This being the case, some special exceptions to a general categorisation of existing features is unavoidable. With which caveat in mind, I propose the following as a general model for applying features to text:

1. *Default glyph pre-processing.* This stage directly modifies the outcome of character-to-glyph mapping from the font 'cmap' table. This may involve language-specific preferred glyph substitutions triggered by the OT language system tag[4] and implemented in the <locl> or character set specific layout feature. It may involve composition or decomposition of 'cmap' default glyph representations into more convenient glyphs for subsequent processing, using the <ccmp> layout feature.[5] Note that while <locl> substitutions are unlikely to be contextual, it should not be presumed that they will never be, and <ccmp> substitutions are very likely to be contextual if they, for instance, perform decompositions of precomposed diacritic glyphs only in the context of following combining marks. I believe this stage is probably the most logical place to also handle mirrored form substitutions.
2. *Orthographic unit shaping.* This applies particularly to complex scripts requiring interaction between character string analysis and specific glyph substitution features. In some cases this interaction may be iterative, involving reordering of output from features within the glyph run, and then application of additional features. This is the core work of script shaping, in preparation for the application of typographic features. This stage broadly corresponds to what Microsoft's script specifications refer to as variously as 'Basic shaping forms' and 'Language based forms', but with the difference that it seeks to limit the features involved to those that must be applied to analysed units of writing systems *and never involve interaction between those units.* The unit in question will vary depending on the writing system: *e.g.* orthographic syllables or 'clusters' for Brahmi-derived scripts, or letter groups defined by character joining behaviours for scripts such as Arabic. This kind of shaping is, obviously, most important when application of individual features results in outcomes that require glyph re-ordering by the shaping engine within the orthographic unit, when clearly defining the beginning and end of the unit is essential to correct outcomes. Individual features in this stage must be applied sequentially and typically in a specific order. Note that although lookups for these features should not include inter-unit contextual triggers, and any such should be ignored, intra-unit context strings may occur and should be respected (examples of the latter would be varying <pref> and <blwf> shaping of Telugu *-ra* depending on preceding letter(s), or disabling some Malayalam ligatures in the <akhn> feature

4. The relationship of OT language system tags to document and text language tagging is a matter for applications to manage, but obviously it is essential to getting correct results from even this initial stage of OTL. Because the typeform preference captured by OT language system is a conventional cultural preference rather than linguistically determined per se, it should ideally be possible to independently specify or override automated relationships. The W3C CSS font spec is admirable in this regard.

5. At present, Microsoft's script specifications do not include <ccmp> for Indian writing systems. Instead, some aspects of glyph pre-processing composition are applied using the <nukt> and <akhn> feature. I'm not aware of a good reason why the Indic shaping model should differ from that of other scripts in this regard.

depending on position within longer conjunct sequences).

After orthographic unit shaping is completed, all restrictions particular to the length of the unit should be relaxed and no longer applied to subsequent features.

3. *Typographic presentation.* For ease of discussion, I am separating this stage into two parts, but it is important to note that features applied in this stage should be considered to apply simultaneously; that is, the order of substitutions, their outcomes and interactions should be governed by the order of associated lookups in the font. This order may involve standard and discretionary feature lookups being staggered, such that the outcomes of some standard features may be affected by the application of discretionary features.

   a) *Standard typographic presentation.* For some writing systems, this includes what Microsoft's Indic specifications call 'Mandatory presentation forms', but it also inherits substitution features that some of Microsoft's specifications classify with 'Basic shaping forms', but which do not fit in the proposed orthographic unit shaping stage because they may involve inter-unit interaction. The purpose of this stage is to arrive at a typical standard of text presentation according to the typographic norms of the writing system, the nature of the individual typeface design, and the capability of the individual font. This means that this stage involves both required layout features that must always be applied, such as <rlig>, and default or standard features that should be applied but that the user may opt to turn off, such as <liga>. It should be obvious by now, I hope, that no restriction should be placed on the extent of interactions between glyphs in these features.[6]

   b) *Discretionary typographic presentation.* All these features are presumed to be inactive by default, and need to be turned on by the user for selected text (this might be through an application UI, or via feature tagging of text as in the case of CSS).

4. *Positioning.* Although not formally required, it typically makes sense for positioning feature lookups to be ordered in a font according to lookup type, which can also be arrived at by thinking of a progression from positioning of connecting base forms, to non-connecting, to marks relative to base forms, to marks relative to marks. Presuming the font maker has succeeded in getting the 'GPOS' table to compile without overruns—not always the easiest thing—, software should apply the lookups as ordered in the font. Since many aspects of positioning are concerned with resolving relationships of visual adjacency, it is particularly important that restrictions not be placed on inter-unit interaction or context length during this stage. For many writing systems, it is necessary to affect contextual kerning or adjustments to mark positioning based on visually but not sequentially adjacent glyphs. This is only possible if lookup contexts are tracked across orthographic unit boundaries.

One thing that should be immediately apparent from this model is that it is progressive: features set up conditions for subsequent features. The process of OpenType Layout is not complete until the

6. No restriction, that is, other than the full extent of the glyph run as defined by font, size, direction and linebreak. It should be noted here, though, that division of glyph runs sometimes results in problems of adjacency that, hence, cannot be resolved in OTL other than by manual application of discretionary features. A good example of this is the display of traditional Arabic dates involving the enclosing year sign, one or more numerals, and a following abbreviation. The latter may require a special form of letter, which the directional division of glyph runs means cannot be triggered by the context of the year sign and numerals. A solution to this would be very welcome.

full progression of stages and their relevant features has been gone through and the desired typographic result achieved.

The tables that follow attempt to categorise and group the current set of registered OTL feature tags according to this model. This should be considered preliminary work, subject to review. The fourth column in the tables indicates recommended default state of the feature: 1/0 = on by default and cannot be turned off; 1/1 = on by default, but can be turned off; 0/1 = off by default but can be turned on.

1. Default glyph pre-processing features (recommended processing and font order)

Tag	Name	Comment	State
locl	Localized Forms		1/0
hngl	Hangul	Korean only	0/1
hojo	Hojo Kanji Forms (JIS X 0212-1990 Kanji Forms)	Japanese only	0/1
jp04	JIS2004 Forms	Japanese only	0/1
jp78	JIS78 Forms	Japanese only	0/1
jp83	JIS83 Forms	Japanese only	0/1
jp90	JIS90 Forms	Japanese only	0/1
nlck	NLC Kanji Forms	Japanese only	0/1
smpl	Simplified Forms		0/1
tnam	Traditional Name Forms	Japanese only	0/1
trad	Traditional Forms		0/1
ltrm	Left-to-right mirrored forms		1/0
ltra	Left-to-right alternates		1/0
rtlm	Right-to-left mirrored forms		1/0
rtla	Right-to-left alternates		1/0
ccmp	Glyph Composition/Decomposition		1/0
stch	Stretching Glyph Decomposition		1/0
nukt	Nukta Forms	Indic only	1/0
akhn	Akhands	Indic only	1/0

2. Orthographic unit shaping (required processing order)

Tag	Name	Comment	State
rphf	Reph Forms	South and Southeast Asian scripts; triggers reordering	1/0
pref	Pre-Base Forms	South and Southeast Asian scripts; triggers reordering	1/0
rkrf	Rakar Forms	South and Southeast Asian scripts	1/0
abvf	Above-base Forms	South and Southeast Asian scripts	1/0
blwf	Below-base Forms	South and Southeast Asian scripts	1/0
half	Half Forms	South and Southeast Asian scripts	1/0
pstf	Post-base Forms	South and Southeast Asian scripts	1/0

Tag	Name	Comment	State
vatu	Vattu Variants	Used, inconsistently, instead of <rkrf> for some Indic scripts	1/0
cfar	Conjunct Form After Ro	Currently Khmer only	1/0
cjct	Conjunct Forms	South and Southeast Asian scripts	1/0
med2	Medial Forms #2	Currently Syriac only; see discussion of topographical features	1/0
fin2	Terminal Forms #2	Currently Syriac only; see discussion of topographical features	1/0
fin3	Terminal Forms #3	Currently Syriac only; see discussion of topographical features	1/0
ljmo	Leading Jamo Forms	Korean only	1/0
vjmo	Vowel Jamo Forms	Korean only	1/0
tjmo	Trailing Jamo Forms	Korean only	1/0

3a. Standard typographic presentation (font order may vary; processed simultaneously with 3b)

Tag	Name	Comment	State
abvs	Above-base Substitutions		1/0
blws	Below-base Substitutions		1/0
calt	Contextual Alternates		1/1
clig	Contextual Ligatures		1/1
fina	Terminal Forms	May be (2) for some writing systems; see discussion of topographical features	1/0
haln	Halant Forms		1/0
init	Initial Forms	May be (2) for some writing systems; see discussion of topographical features	1/0
isol	Isolated Forms	May be (2) for some writing systems; see discussion of topographical features	1/0
jalt	Justification Alternates	Could be considered (3b) if not applied by standard justification algorithms	1/1
liga	Standard Ligatures		1/1
medi	Medial Forms	May be (2) for some writing systems; see discussion of topographical features	1/0
mset	Mark Positioning via Substitution	Legacy feature, superceded by <mark>	1/0
pres	Pre-base Substitutions		1/0
psts	Post-base Substitutions		1/0
rand	Randomize		1/1
rclt	Required Contextual Forms		1/0
rlig	Required Ligatures		1/0
vert	Vertical Writing	Applied based on text layout; use this or <vrt2>, not both; UTR50 implementation	1/0
vrt2	Vertical Alternates and Rotation	Applied based on text layout; use this or <vert>, not both	1/0

3b. Standard typographic presentation (font order may vary; processed simultaneously with 3a)

Tag	Name	Comment	State
afrc	Alternative Fractions		0/1
c2pc	Petite Capitals From Capitals		0/1
c2sc	Small Capitals From Capitals		0/1
case	Case-Sensitive Forms	Could be (3a) if applied heuristically	0/1
cpct	Centered CJK Punctuation	Mostly CJKV fonts	0/1
cpsp	Capital Spacing	Could be considered (3a) if applied heuristically	0/1
cswh	Contextual Swash	[Probably redundant feature]	0/1
cv01-cv99	Character Variants		0/1
dlig	Discretionary Ligatures		0/1
dnom	Denominators	Mostly superceded by contextual <frac> implementations	0/1
expt	Expert Forms	Currently Japanese only	0/1
falt	Final Glyph on Line Alternates	Might be considered (3a) in some implementations	0/1
frac	Fractions	Could be considered (3a) if applied heuristically	0/1
fwid	Full Widths	Mostly CJKV fonts	0/1
halt	Alternate Half Widths	See also <vhal> positioning	0/1
hist	Historical Forms		0/1
hkna	Horizontal Kana Alternates	Currently Japanese kana only; could be applied automatically based on text layout; cf. <vkna> vertical equivalent	0/1
hlig	Historical Ligatures	[Probably redundant feature]	0/1
hwid	Half Widths		0/1
ital	Italics	Mostly CJKV fonts; alternative to TTC/OTC implementation	0/1
lnum	Lining Figures		0/1
mgrk	Mathematical Greek		0/1
nalt	Alternate Annotation Forms		0/1
numr	Numerators	Mostly superceded by contextual <frac> implementations	0/1
onum	Oldstyle Figures		0/1
ordn	Ordinals		0/1
ornm	Ornaments		0/1
palt	Proportional Alternate Widths	Mostly CJKV fonts; see also <vpal> positioning	0/1
pcap	Petite Capitals		0/1
pkna	Proportional Kana	Japanese kana only	0/1
pnum	Proportional Figures		0/1
pwid	Proportional Widths	Mostly CJKV fonts	0/1

Tag	Name	Comment	State
qwid	Quarter Widths	Mostly CJKV fonts	0/1
ruby	Ruby Notation Forms		0/1
salt	Stylistic Alternates		0/1
sinf	Scientific Inferiors	Redundant feature, superceded by typical use of <subs>	0/1
smcp	Small Capitals		0/1
ss01–ss20	Stylistic Sets		0/1
subs	Subscript		0/1
sups	Superscript		0/1
swsh	Swash		0/1
titl	Titling		0/1
tnum	Tabular Figures		0/1
twid	Third Widths	Mostly CJKV fonts	0/1
unic	Unicase		0/1
vkna	Vertical Kana Alternates	Currently Japanese kana only; could be applied automatically based on text layout; cf. <hkna> horizontal equivalent	0/1
zero	Slashed Zero		0/1

4. Positioning (recommended font order)

Tag	Name	Comment	State
opbd	Optical Bounds	Applied as part of optical margin alignment; probably redundant, see <lfbd> & <rtbd>	0/1
lfbd	Left Bounds	Applied as part of optical margin alignment	0/1
rtbd	Right Bounds	Applied as part of optical margin alignment	0/1
valt	Alternate Vertical Metrics	Applied based on text layout	1/0
vpal	Proportional Alternate Vertical Metrics	Applied based on text layout	0/1
vhal	Alternate Vertical Half Metrics	Applied based on text layout	0/1
curs	Cursive Positioning		1/0
dist	Distances	Like <kern> but not subject to discretionary disabling	1/0
kern	Kerning		1/1
vkrn	Vertical Kerning	Applied based on text layout	1/1
mark	Mark Positioning		1/0
abvm	Above-base Mark Positioning	South and Southeast Asian scripts	1/0
blwm	Below-base Mark Positioning	South and Southeast Asian scripts	1/0
mkmk	Mark to Mark Positioning	Results may be subject to manual override or editing in some applications	1/0

Especially observant readers will have noticed that two registered features are not included in these tables: <aalt> Access All Alternates, and <size> Optical Size. The <aalt> feature is an access mechanism for presenting glyph variants in a user interface, so is not expected to be applied to text as part of layout. The <size> feature is a frankly ill-considered hijacking of the 'GPOS' table data structure to record the size range for which a given type design is intended. If it were implemented at all within the context of text layout, it would occupy a stage 0 (zero) affecting the selection of a particular font within a family before OpenType Layout proper is applied. Other, methods for size-specific design selection are available or proposed, and the <size> feature may be best deprecated.

Considering a special case: topographical features

I'm going to conclude by considering a notable special case among the OTL features: the <isol> <init> <medi> and <fina> topographical features[7] that substitute appropriate glyph variants based on location. In addition to these, there are three specifically Syriac topographical features used in orthographic unit shaping for that script, and several of the Indic orthographic unit shaping features are also of a topographical nature but not relevant to this discussion. The existence of script-specific topographical features for Syriac, though used in concert with the generic set, indicates a decision that could have been made in the design of OpenType Layout that would have avoided ambiguity about the rôle and application of the topographical features. It would have been possible to define a set of such features specifically for orthographic unit shaping of scripts with inherent joining behaviours defined in Unicode, such as Syriac, Arabic, Mongolian, etc., and for this to be separate from typographic presentation of topographical glyph variants that are a design element of particular typefaces rather than an aspect of a writing system. Instead, we currently have a set of features that are used for orthographic shaping for some writing systems and for typographic presentation for others, and which, more confusingly, apply to different topographies depending on the writing system (lettergroups in the case of joining scripts, words in the case of others, and presumably phrases if applied to a script such as Thai that does not visually break individual words).

There are a number of options for how topographical substitutions could be handled, moving forwards. One option is simply to continue to live with this dual use, but to better define the expected behaviour and proper application of the feature for various writing systems. The goal in that case would be to make clear that in a typographic presentation context these features may apply to any writing system, and will be appropriate to particular styles or individual typeface designs. The features may be used, for instance, for the Latin writing system in cursive 'script' style typefaces; while the <init> feature is currently spec'd by Microsoft for 'mandatory presentation forms' in the Bengali writing system, <fina> or other topographic substitutions would be appropriate to some letters in some styles of Bengali type.

The limitation of this option is that it enforces a distinction at the feature level between different topographies for different writing systems, which in effect means that word-level topographies—as distinct from lettergroup topographies—are not accessible to writing systems with standardised joining behaviours defined in Unicode. So, for example, application of the existing features to the Arabic writing system is based on joining behaviour analysis and hence lettergroup topography, meaning that there is no way to specifically address and substitute initial or final *word* glyphs in Arabic typeface designs.

7. These features are sometimes referred to as positional, but that term seems too similar to positioning. Locational has been suggested, but the notion of text consisting of topographies may be a helpful one in recognising new ways to address aspects of layout.

This suggests another option, which is to register a new set of specifically typographic presentation features for word-topographic substitutions, to be applied after orthographic unit shaping and independently of substitutions determined by joining behaviour. Once this is considered, features for other topographies suggest themselves: first glyph on line, for example, or even page or text block topographies enabling, for instance, taller display glyphs on the top line and deeper ones on the bottom line, as seen in many manuscript traditions. This begins to move the discussion in the direction of new features and alternative layout models, so seems a good place to end this paper.

CHANGE LOG

Revision 1.1
After consultation with Ken Lunde, Adobe, moved CJK form selection features to default glyph pre-processing stage (1). Note that this means this stage now contains both 1/0 (required) and 0/1 (discretionary) features.

Revision 1.2
Incorporated additional feedback from Ken Lunde to ordering and annotation of CJK layout features in tables. After discussion with David Lemon, Adobe, modified statement regarding <size> feature to reflect uncertain suitability of recent new size-selection mechanisms to all font families. Made minor revisions to some wording. Added change log.