Appendix C: How to Deal with Unknown or Missing Characters

This section briefly discusses how users or implementers might deal with characters that are not understood or which though understood, are unavailable for legible rendering. Characters not currently included in the Unicode standard are discussed in Appendix E.

Unassigned or Private Use Characters

There are two classes of characters which even a "complete" Unicode implementation cannot necessarily interpret correctly:

- Characters in the Private Use Area where there is no private agreement specifying use of that area.
- Potentially legal characters which, in the version of the Unicode standard being implemented, are not assigned.

An implementation should treat such a character as an independent graphic entity. It is generally safe to assume that the character is a simple weak left-to-right non-letter and non-digit character about which nothing else is known, unless the code is in the range of Unicode values reserved for right-to-left scripts, U+0500 → U+08FF, in which case the suitable default would be a strong right-to-left character. Some options for rendering such unknown characters include printing four hexadecimal digits, printing a black or white box, using a glyph (for example, that shown at U+FFFD), or simply displaying nothing. In no case should an implementation *assume* anything else about the character's properties, nor should it blindly delete such characters. It should not unintentionally transform them into something else.

Conflicting Uses of Private Use Area

Multiple assignments for the Private Use area may exist, where different domains overlap.

In this case, one of the uses could be re-mapped to another, unused, part of the User Space. Some implementations might also wish to provide higher level protocols to switch between interpretations of the User Space. If such instances become common, it may be that the groups involved should reach a new mutual private agreement about use of User Space, or if they form a sizeable community, that they should request the formal inclusion of the characters, or scripts in question in a future version of the Unicode standard.

Known Character

An implementation may receive a character which is an assigned character in the Unicode character encoding, but be unable to render it with the appropriate semantics because it does not have a font for it, or is otherwise incapable of rendering it appropriately.

An implementation should treat such a character code as if it were a character in the Private Use Area of the Unicode standard. In this case, an implementation might be able to provide further limited feedback to the user's queries such as being able to sort the data properly, show what script (or language) it is, or display it in a manner which is documented to be used only for known legal characters which cannot be appropriately rendered. An implementation might, for instance, distinguish between unrenderable characters in assigned zones and a code value with no character assignment by printing one as a box and the other with hexadecimal digits, or by printing the known characters with a special glyph that gives some general indication of their type.

Handling Numbers

There are many sets of characters that represent decimal digits in different scripts. Systems that interpret those characters numerically should provide the correct numerical values. For example, the sequence *Devanagari digit two, Devanagari digit zero* are numerically interpreted as having the value twenty. When converting binary numerical values to a visual form, digits can be chosen from different scripts. The value twenty can either be represented by *digit two, digit zero*, or by *Devanagari digit two, Devanagari digit zero* or by *Arabic-Indic digit two, Arabic-Indic digit zero.* It is recommended that systems allow users to choose the format of the resulting digits, including the script. Some existing systems allow users to override the appearance of digits in plain text. For example, they provide controls that allow users to choose to see *digit zero* as if it were *Arabic-Indic digit zero,* and so on. Such a change can be interchanged in plain text by replacing the appropriate occurrence of *digit zero* with *Arabic-Indic digit zero.*

Glyphs and Fonts

Any implementation of a rendering process makes use of a collection of glyphs. A *font* is a collection of glyphs, all in the same stroke style, containing at most one glyph from each set of variants. The word *font* suffers from a considerable amount of abuse; what is important in the usage is the restriction to at most one glyph from each set of variants.

A font implementation must index glyphs within the font by unique ID's. A *glyphID* is a private code name or number used to index the glyphs within a font.

In general, a font and its associated rendering process define an abitrary mapping between glyphIDs and Unicode values. Some of the glyphs in a font may be independent forms for individual characters, while others may be rendering forms that do not directly correspond to any one character. For those glyphs that are independent forms, it may be convenient for the glyphID to have the same numerical value as the Unicode value, but this is not required.