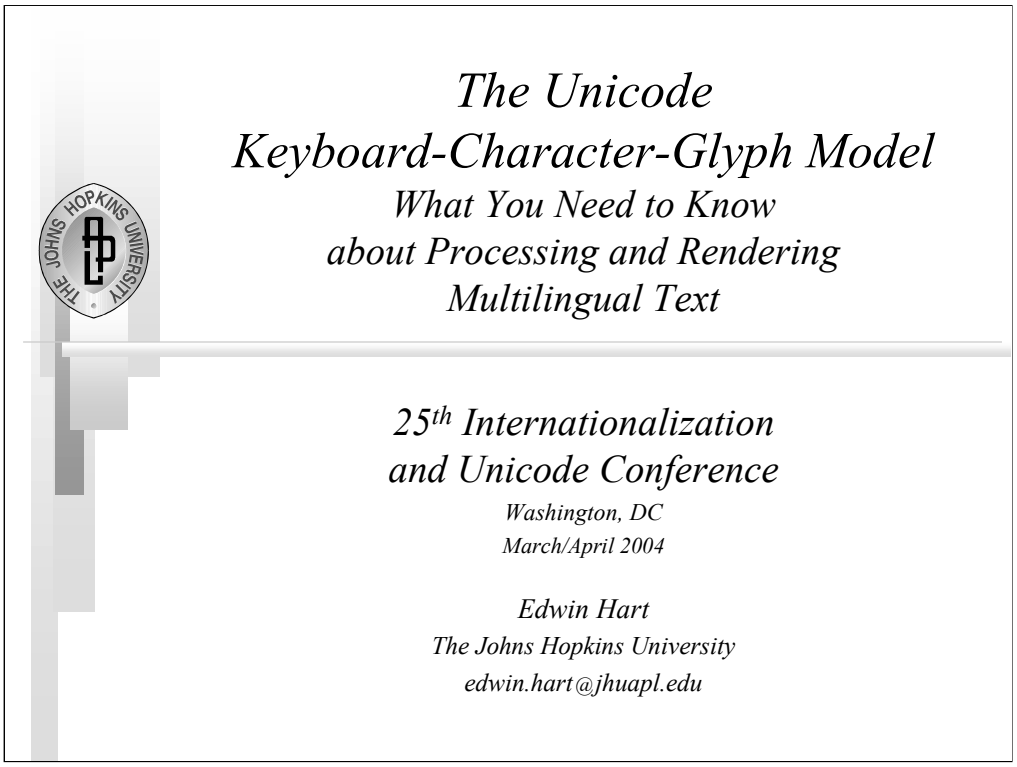
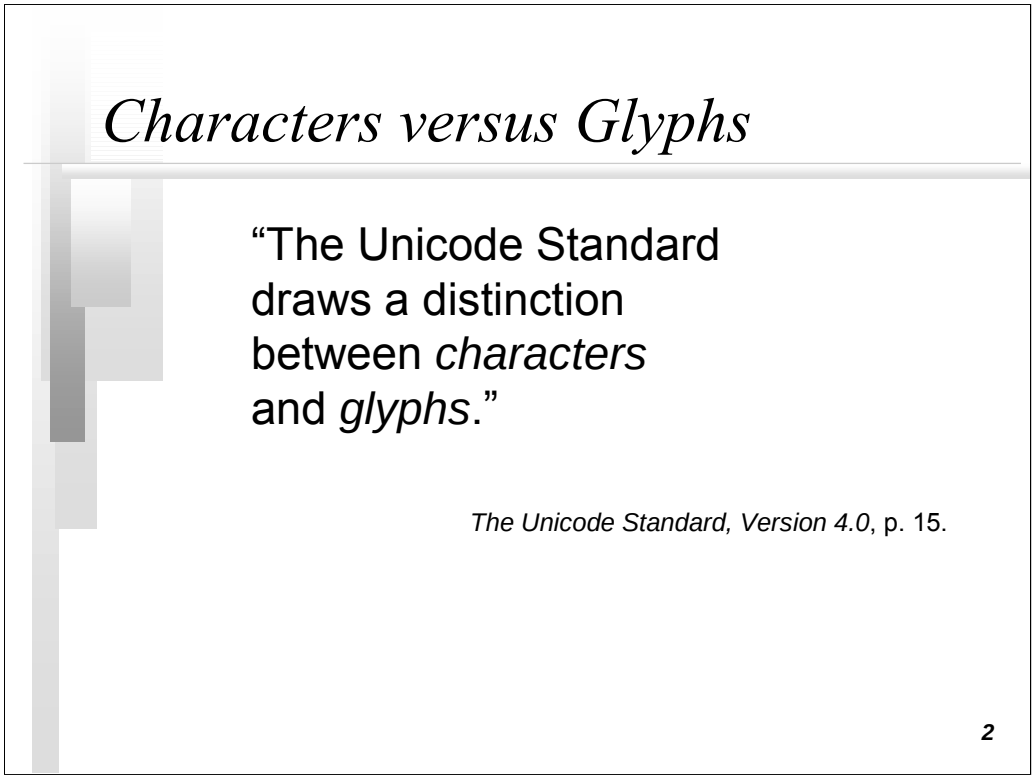


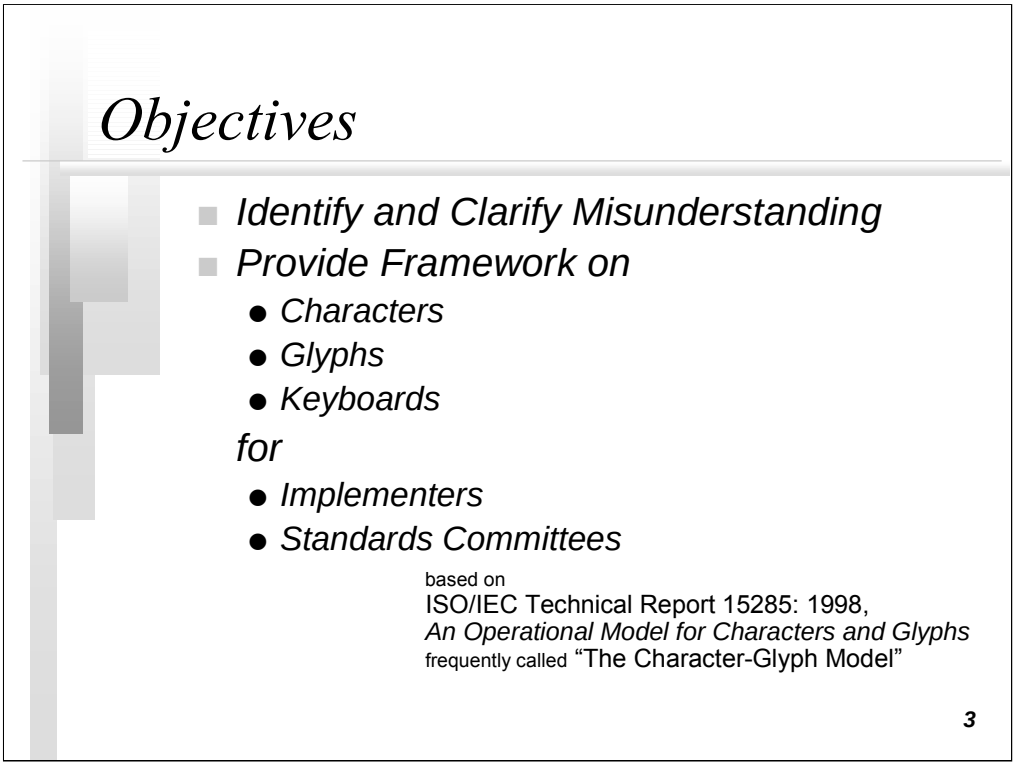
©2002, 2004 Edwin F. Hart, All Rights Reserved.
Global Meeting Services, Inc. has permission to reproduce this material
in *Proceedings of the 25th Internationalization and Unicode Conference*



"The Unicode Standard draws a distinction between *characters*,…, and *glyphs*,…."

- What does this mean?
- What are *characters*?
- What are *glyphs*?
- How are they related?
- How are they different?
- How are characters converted into glyphs?
- Why is this relevant to rendering multilingual text?

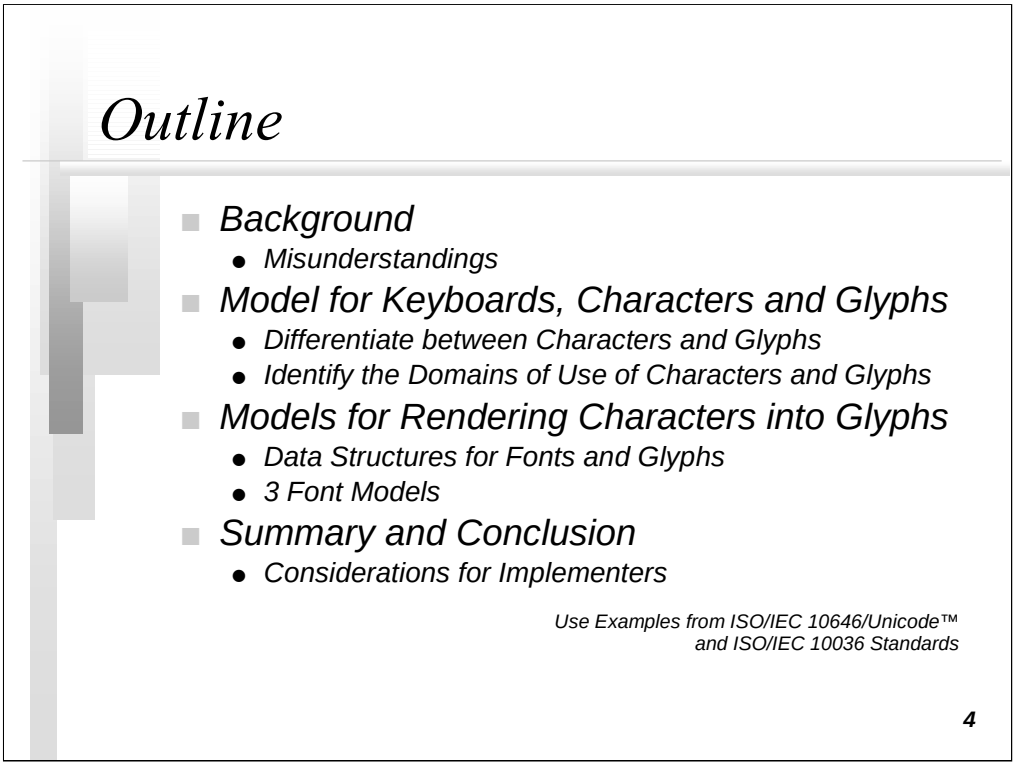
This presentation will answer these questions.



3

The original reason for developing this presentation is that the concepts behind characters and glyphs were misunderstood not only by implementers, but also by people on the standards committees. Understanding these concepts is particularly important for rendering multilingual text for people to read. Most recently, we added materials on keyboard input. If you are developing multilingual software, you need to be aware of these concepts. By the conclusion of this session, you should have a much better understanding of characters, glyphs, keyboard input, and how to render characters into glyphs.

The information in this presentation is based on an ISO Technical Report, ISO/IEC TR 15285, *An operational model for characters and glyphs*.



This presentation has four parts:

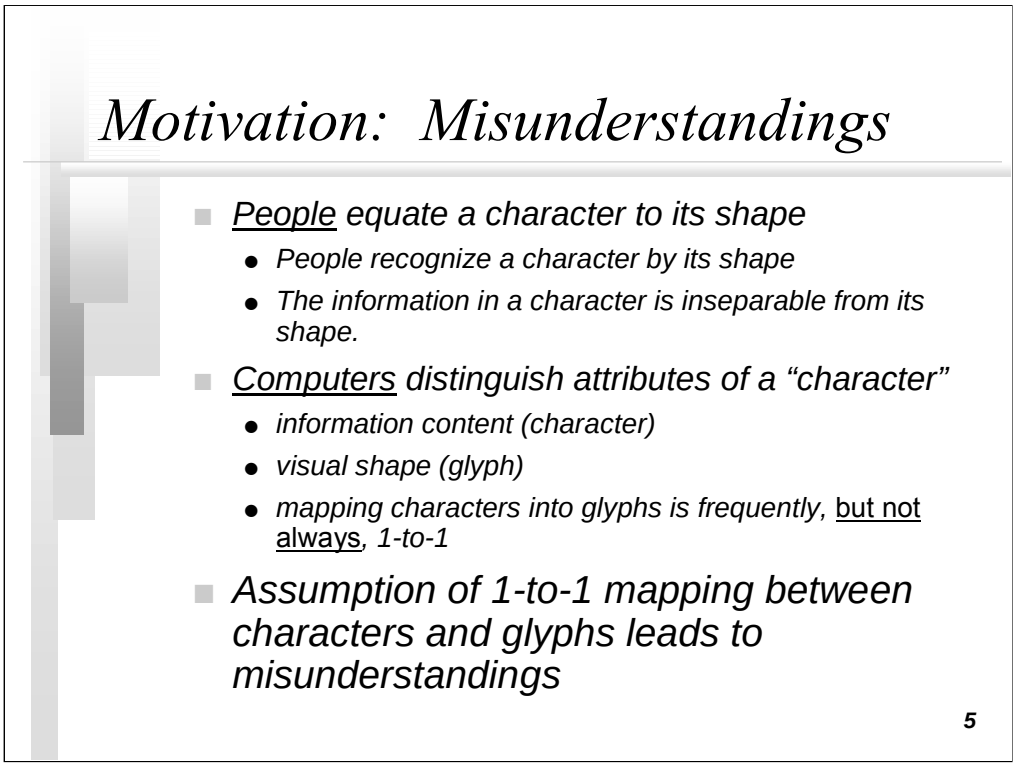
First, we describe the misunderstandings that motivated development of this session and provide several examples to help illuminate the concerns.

Second, we describe a model for keyboards, characters and glyphs to differentiate between them. We also identify where to use characters versus glyphs. We then show several examples of (1) the glyph-selection process of mapping characters into glyphs and (2) the keyboard input method process of mapping keyboard scan codes into characters.

Third, we describe three models for rendering characters into glyphs. We describe the data structures required for this process and then three font models. We conclude this section with a comparison of the models and a recommendation of which ones to use for multilingual text processing.

Fourth, we conclude with a summary and a list of considerations for implementers.

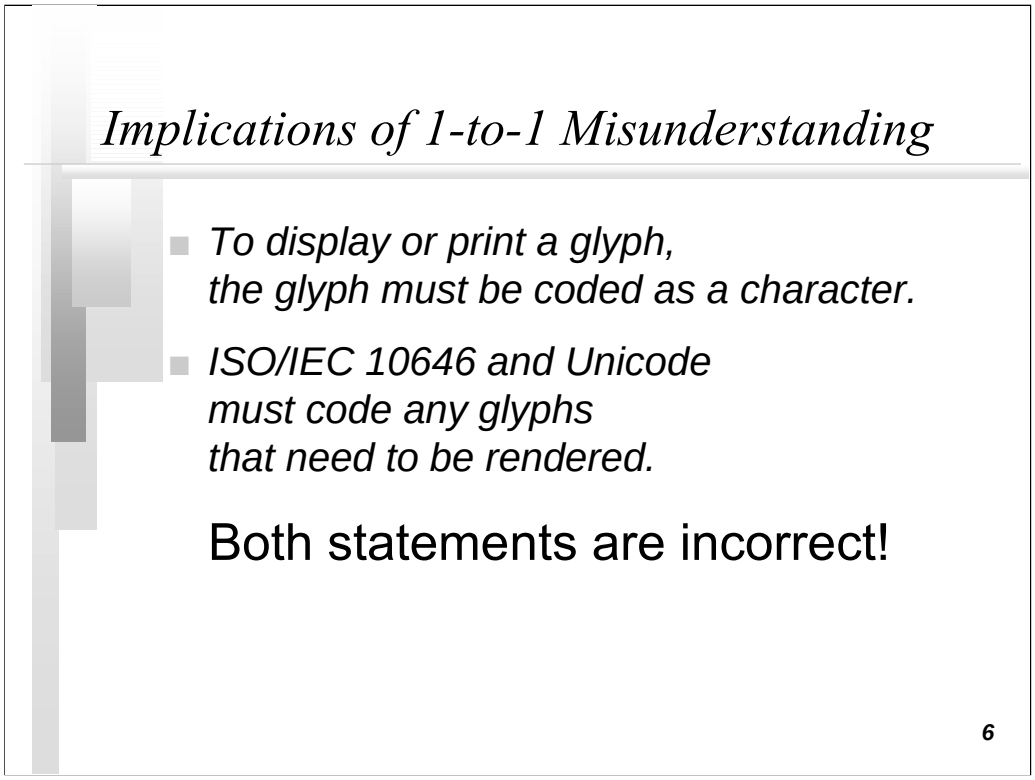
This presentation uses examples from the Unicode implementation of ISO/IEC 10646-1: 2000 and the ISO/IEC 10036 standard.



When you learned your letters you were taught to recognize a letter by its shape. As far as you were concerned, the letter equated to its shape and the two were inseparable.

Although this was fine for learning to read, in Information Technology we distinguish a character's attributes of information content versus its shape. We use the term *character,* to describe the information content attribute, and the term *glyph,* to describe the shape attribute. This distinction appears arbitrary because for most characters coded in Unicode, a one-to-one relationship exists between a character's information content and its shape. Notice that we said "for most characters" because, in multilingual text processing, the relationship is not always one-to-one. The incorrect assumption of a one-to-one relationship between the information content and shape attributes leads to misunderstandings.

One can liken this to a similar situation in Physics. For most of the world we perceive, Newtonian Physics works very well. However, in the late 1800s and early 1900s, Physicists identified situations where Newtonian Physics was inadequate. It took Einstein to discover Relativity theory that provided better answers. Returning to the domains of characters and glyphs, for the most part, a one-to-one is sufficient; however, in a number of situations, the relationship is more complex and a one-to-one relationship fails.



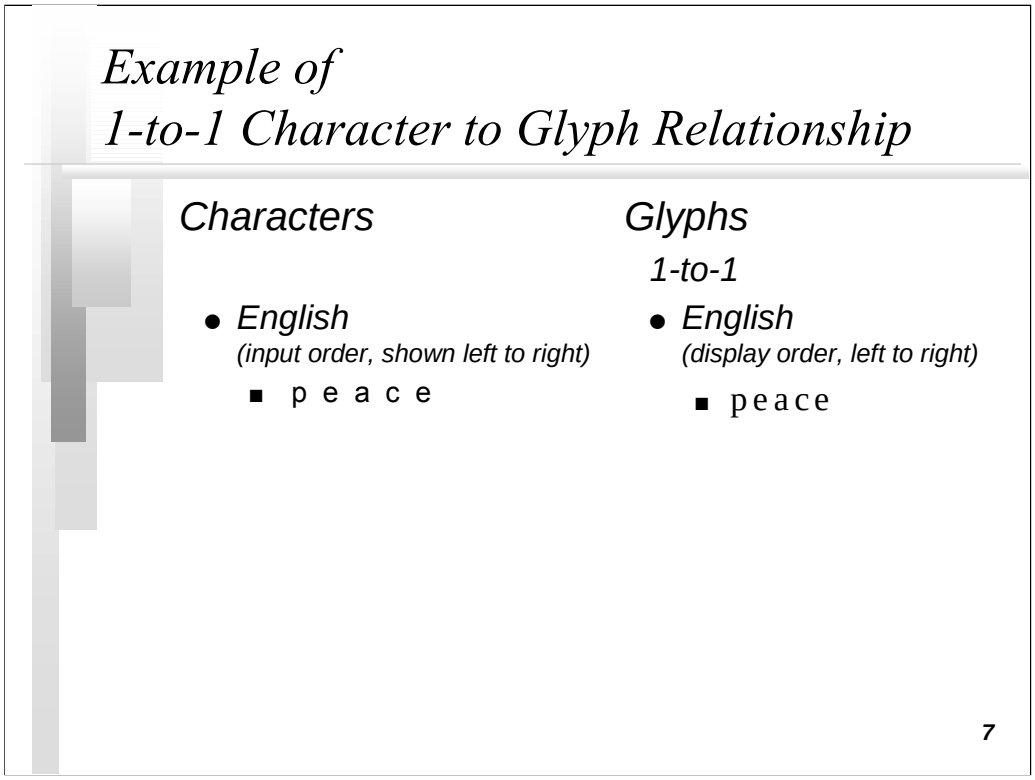
If you assume the 1-to-1 relationship as a requirement (and many people on the standards committee made this assumption when developing ISO/IEC 10646), then you obtain two conclusions:

First, if you want to see a particular shaped character, then it must be coded as a character in a code.

Furthermore, if this is true, then ISO/IEC 10646 and Unicode must code the shapes than need to be rendered.

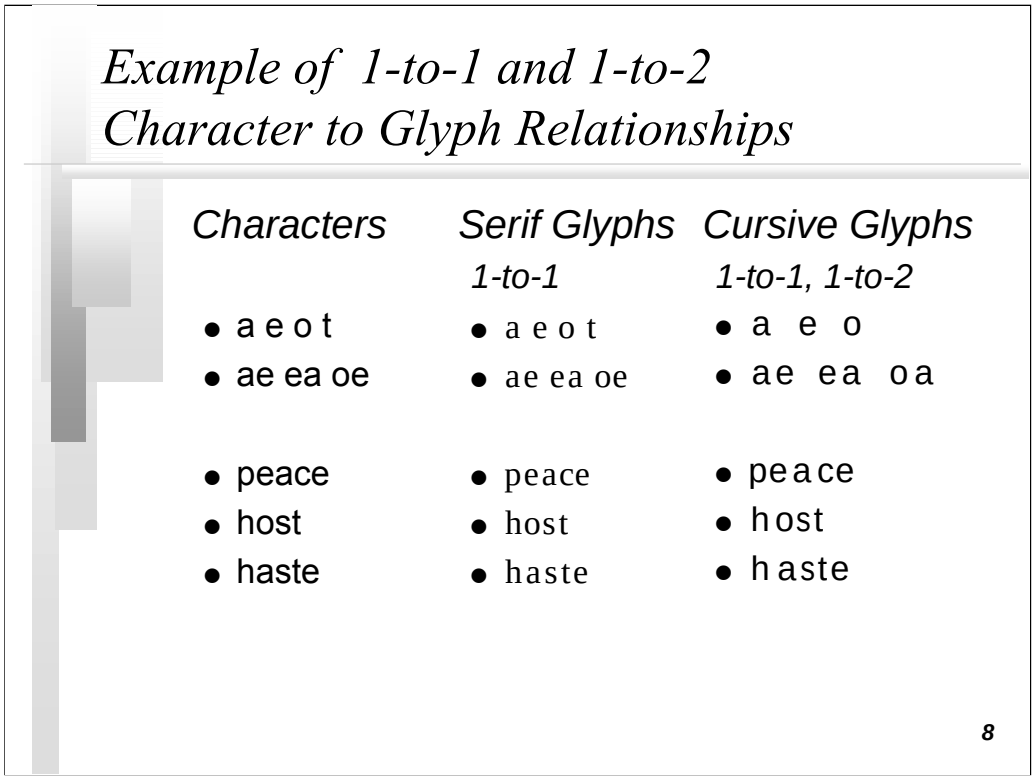
Let me assure you that both of these statements are false due to a false assumption.

The next several slides will illustrate some of the complexity of mapping characters to glyphs.



We will now look at three examples that illustrate increasing complexity in the rendering of characters into glyphs.

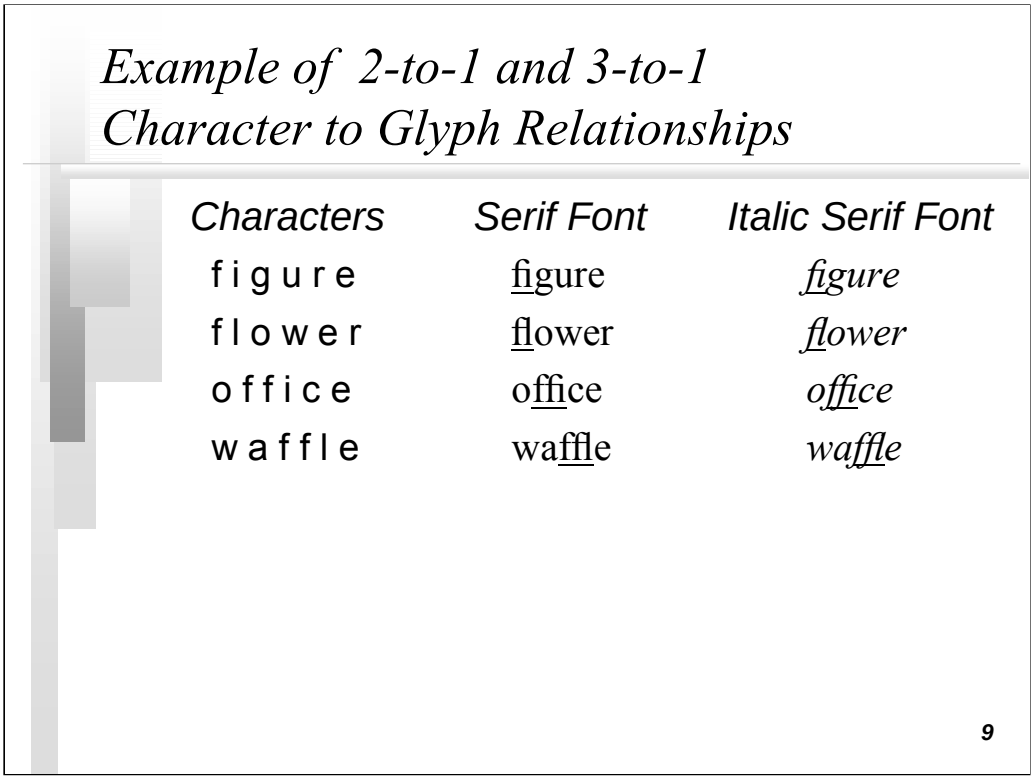
The first is a simple example in English. (The figure has characters on the left and glyphs on the right.) You type "p", "e", "a", "c", and then "e" to form the word "peace". Text is stored in logical or input order. In this case, you see it in the natural left-to-right order of the English language. On the right, you see the word "peace" displayed, also in left-to-right order. Here we have a 1-to-1 mapping from the 5 English characters into 5 glyphs. This is all very straightforward.



Let's continue with a second English example.

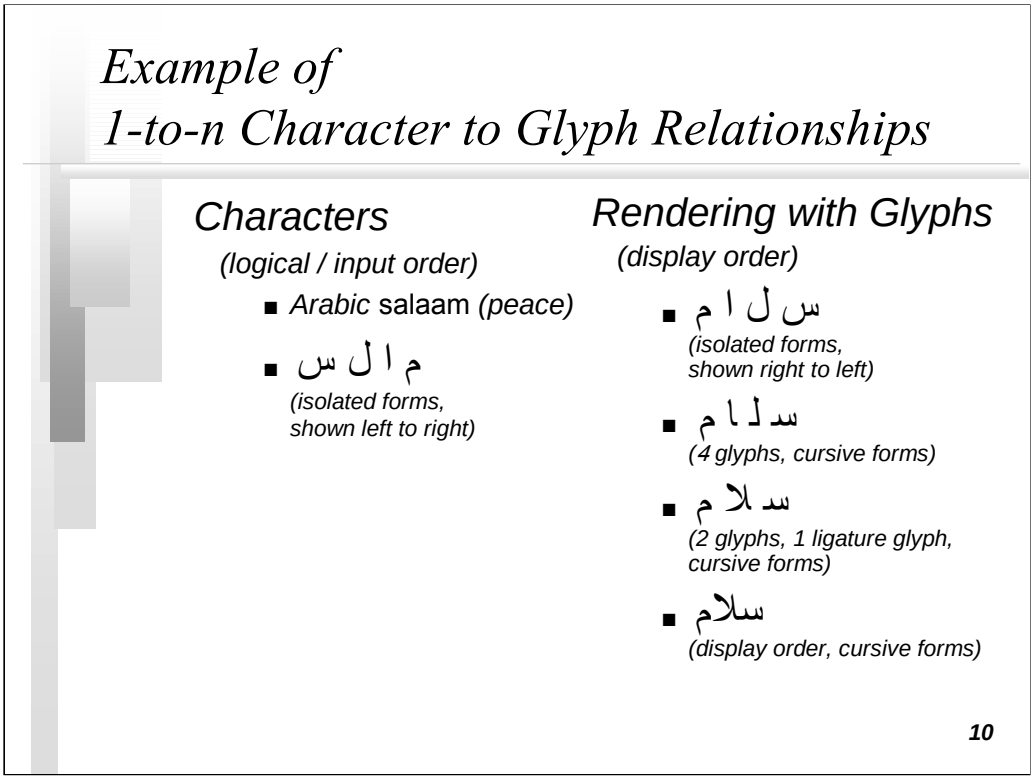
The figure has three columns. The first column contains the characters. The second column show the glyphs in a Serif font (Century Schoolbook). Like the first example on the previous page, the glyphs in the second column have a 1-to-1 relationship to the characters in the first column. Nothing new is in the second column.

However, focus on the third column. Here we have glyphs from a cursive Script font (Lucida Handwriting). The cursive glyphs are connected in words to mimic cursive handwriting. Note that in this font, the lengths of the tails of the "a", "e", and "o" glyphs vary depending on whether the glyph is isolated or at the end of a word versus preceding another letter. For these letters, we see a 1-to-2 character to glyph mapping. Now examine the "t" glyph in the words "host" and "haste". The "t" seems to have the same length tail regardless of what follows. For the "t", we see a 1-to-1 character to glyph mapping. Therefore, selecting the correct glyph for this cursive font depends on the character itself and sometimes on the context of the character. Rather than being strictly 1-to-1, sometimes, the mapping is 1-to-2. Now remember that we are still discussing rendering English text.



Let's continue with a third English example.

Good English typography, especially with serif fonts, makes use of ligatures. A ligature is a glyph formed by joining one or more other glyphs. In these examples, we map 2 or 3 characters into a single ligature. The first column represents the characters. The second column shows the letters in a roman (or upright) serif font (Times New Roman) and underlines the ligatures to identify them. The third column uses an italic serif font (again Times New Roman), where the ligatures seem to be more pronounced than the roman font. In this example, we see more complexity and we are still discussing rendering English text. Clearly, rendering even English text may not be as simple as we might have originally thought. You have just seen several examples where mapping the characters to glyphs is not as simple as a 1-to-1 mapping.

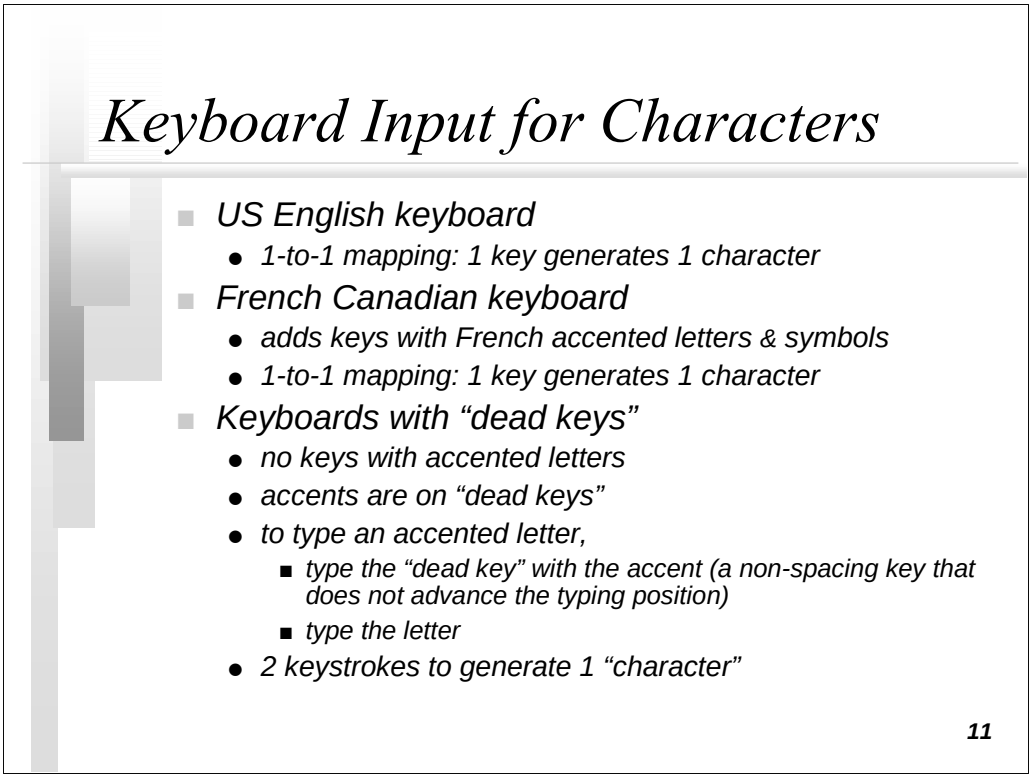


Now, let's turn to a more complex example. Once again, we have characters in the left column and glyphs in the right column. On the left are the letters for the Arabic word "salaam", which means "peace". Someone typing it, would type the Arabic letters "seen", "lam", "alef", and "meem". (The illustration has spaces between the glyphs so that you can more easily identify them.) The figure first shows the 4 letters in left-to-right *input* order.

Now look at the Glyph column on the right. Unlike English, Arabic letters are written from right to left. As part of the rendering process, we first order the letters in right-to-left *display* order. In addition, the Arabic script uses cursive (or joined) forms of the letters. Therefore, like the English example of the Script font, we select glyphs so that they properly connect together. However, each Arabic character can take one of up to 4 shapes depending on the context. The next bullet shows the 4 Arabic glyphs, in display order, with the proper contextual shape. However, we are not yet finished. Arabic typography also uses ligatures. Ligatures are single glyphs that represent multiple letters. In Arabic, the "lam-alef ligature" is mandatory. So we need to replace the "lam" glyph followed by the "alef" glyph with the "lam-alef ligature glyph". The next bullet shows 3 glyphs: the "seen, initial form", the mandatory "lam-alef ligature, final form", and the "meem, isolated form" glyphs. The final bullet shows a proper Arabic rendering of "salaam" without spaces.

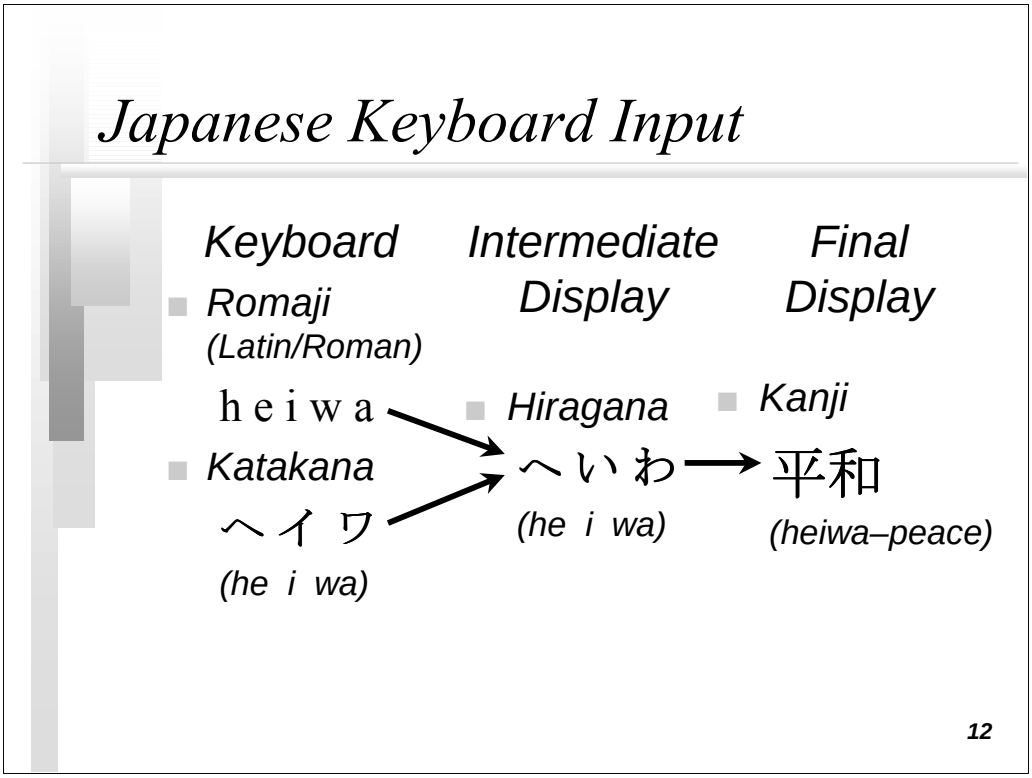
These examples illustrate some of the complexity of rendering characters into glyphs.

Kamal Mansour from Agfa Monotype kindly provided this Arabic example.



Because keyboards are based on typewriter technology, we see similar one-to-one relationships between keystrokes and characters. The input process for languages that use the Latin script are straightforward. The input process for the US English keyboard maps keystrokes one-to-one into characters. However, the US English keyboard has no accented characters. French Canadian keyboards add accented letters to the keyboard so people can type French words. However, the input process still maps keystrokes one-to-one into characters, although they are a different set of characters from the English ones.

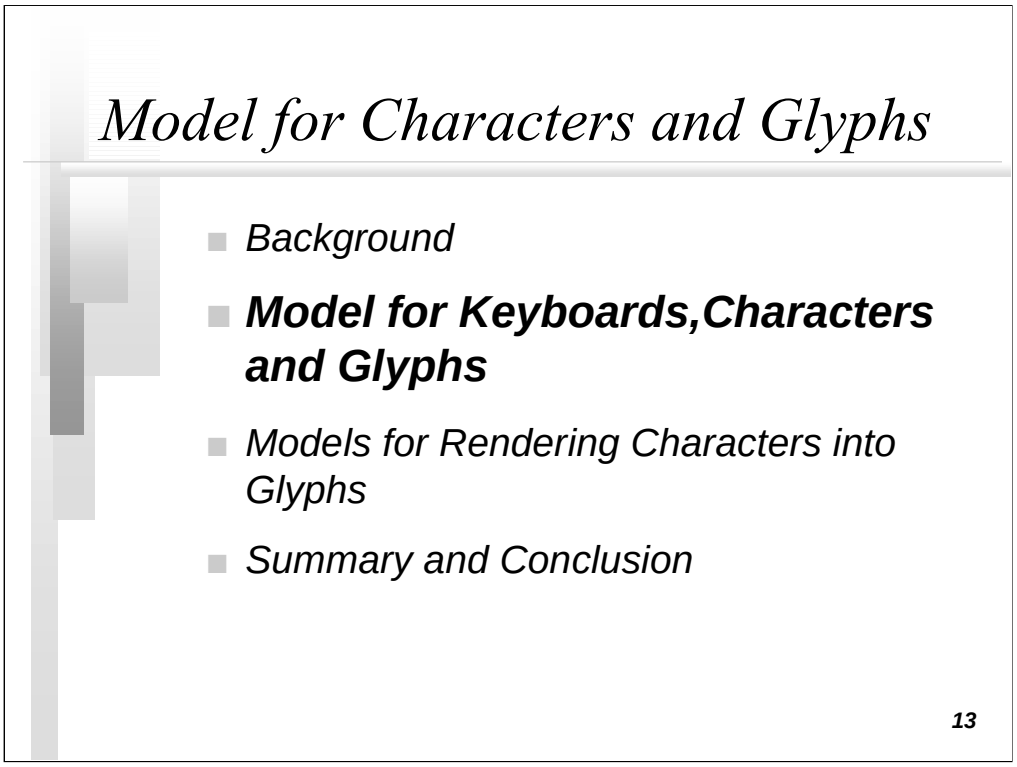
An alternative way to enter accented letters is to use keyboards with what are called "dead keys". These keyboards have no accented letters. However, they have the accents or diacritics located on the "dead keys". The way you enter an accented letter with one of these keyboards is first to type the "dead key" with the desired accent. This will display the accent, but not advance the typing or cursor position. Then you type the letter, which displays the letter at the same typing position as the accent. So on a French Canadian keyboard, you would use one keystroke to enter an accented letter, like LATIN SMALL LETTER A WITH GRAVE, "à". However, on a keyboard with dead keys, you would first type the grave accent and then the small letter A to display the accented letter. So, in contrast to using a French Canadian keyboard, using a keyboard with "dead keys" requires two keystrokes to form one accented letter.



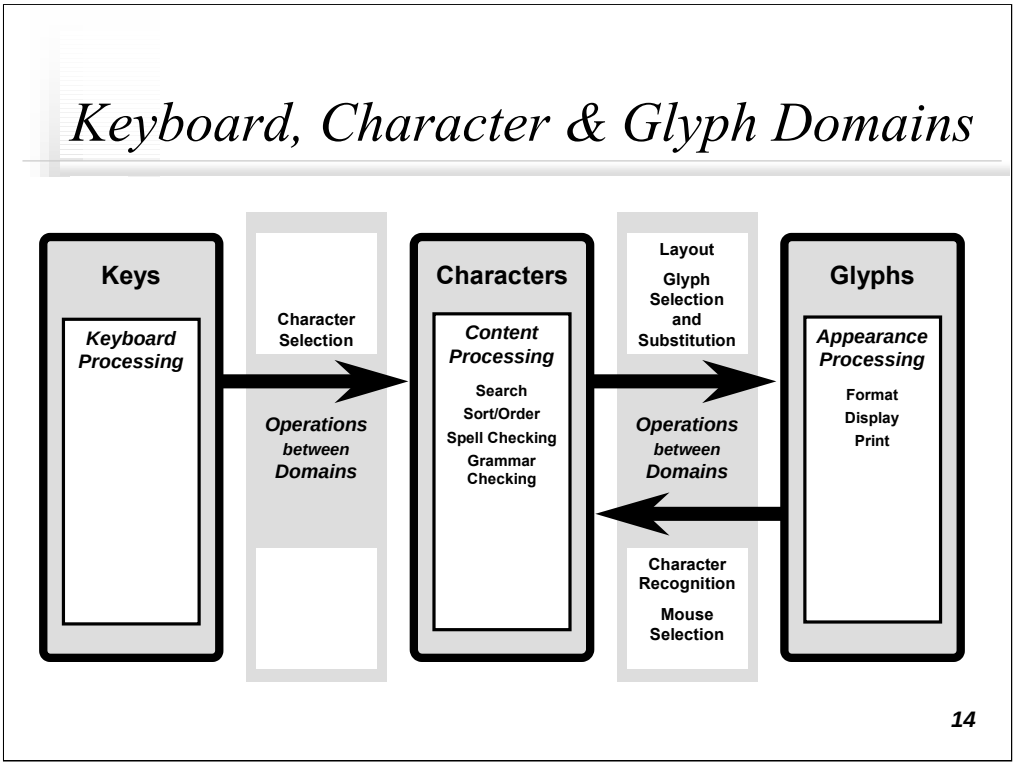
Japanese input is more complex. Japanese uses three scripts. Katakana is a phonetic script used for writing foreign words. Hiragana is another phonetic script but used for Japanese words. Kanji is the Japanese ideographic script.

The Japanese use an elaborate keyboard input system. Here is how it works. The user may use either the Japanese Romaji (or Romanji) keyboard with Latin (or Roman) letters or a Katakana keyboard. As the user types Latin letters or Katakana characters, the display shows the corresponding Hiragana characters. When the user hits the space bar, if the Hiragana sound corresponds to a unique Kanji word, the Kanji ideographs replaces the Hiragana in the display area. If, however, multiple Kanji ideographs have the same sound, another display windows shows alternate Kanji ideographs from which the user then selects the desired Kanji. After selection, the window with the Kanji disappears, and the selected Kanji ideographs then replace the Hiragana characters.

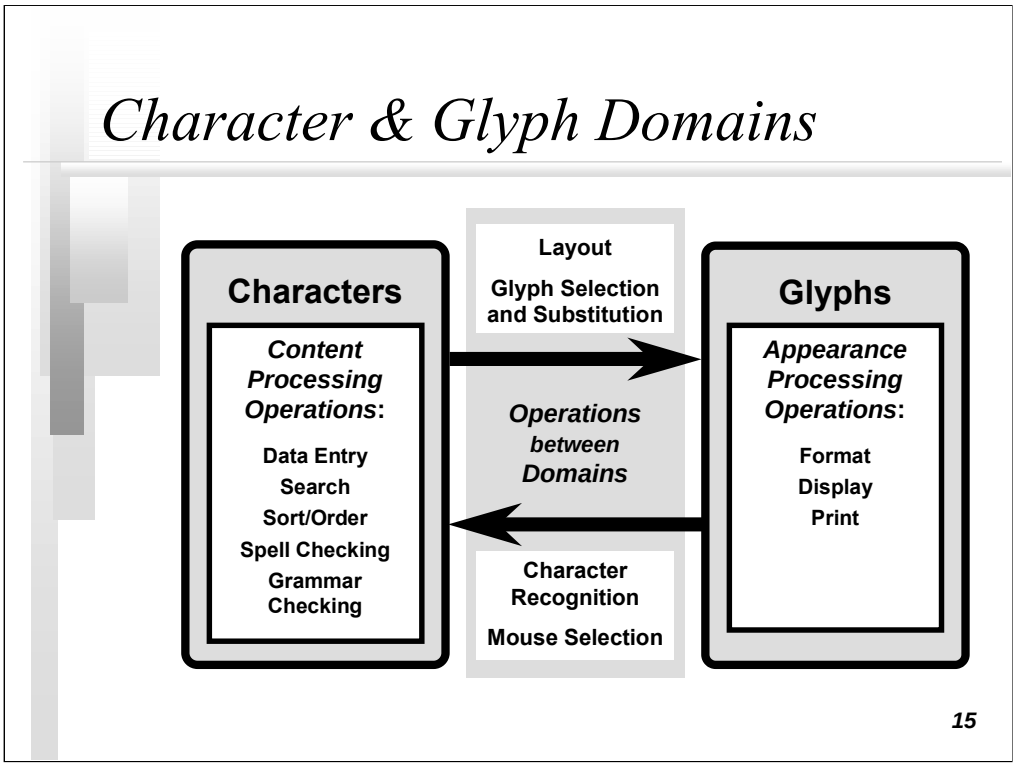
For example, to enter the Kanji for the Japanese word for “peace”, the user either types “h e i w a” on a Romaji keyboard or “ヘイワ” on a Katakana keyboard. The display will show the corresponding Hiragana characters, “へいわ”. When the user hits the space bar, the Kanji characters, “平和”, replace the Hiragana characters because only one pair of Kanji characters corresponds to the sound.



Having described the misunderstandings about characters and glyphs, and given examples to illustrate some of the complexity of the rendering process and the keyboard input process, let's continue by looking at a model for describing keyboards, characters and glyphs.



This figure shows three distinct domains, one for keys on keyboards on the left, one for characters in the middle, and another for glyphs on the right. It also uses arrows to show operations for going from one domain to the other. Note that the character domain provides the common mechanism for taking the keys on the keyboard and displaying the resulting glyphs.

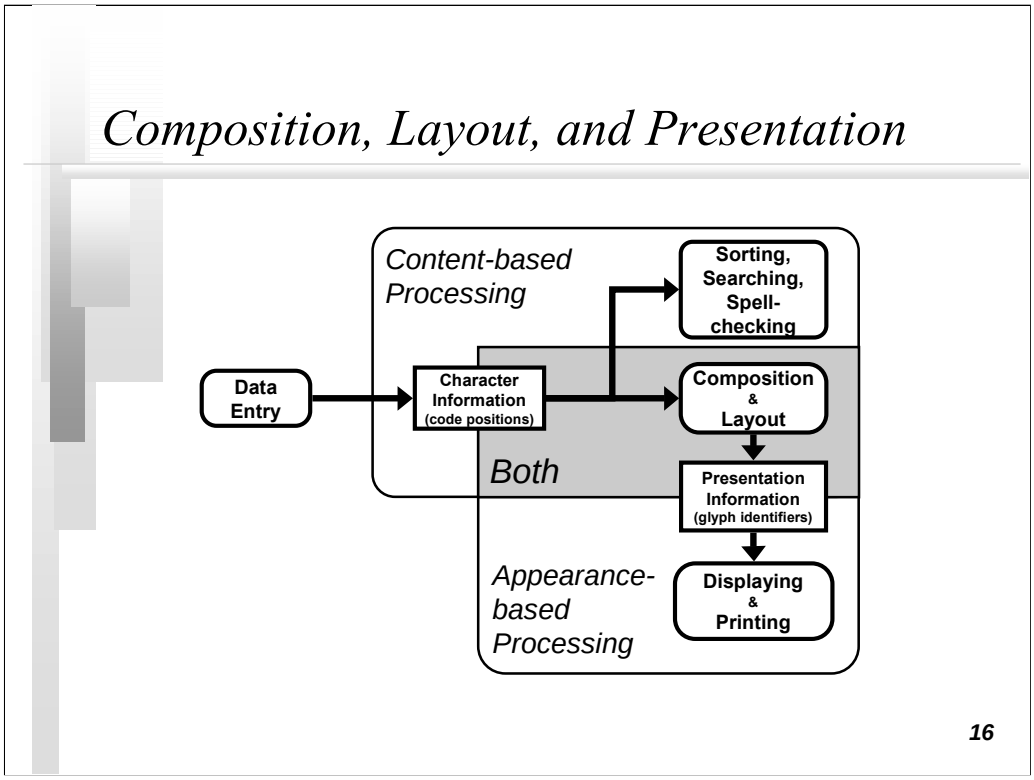


Let's first focus on the character and the glyph domains. This figure shows two separate domains, one for characters on the left and another for glyphs on the right. It also show operations for going from one domain to the other.

The *character domain* on the left concerns itself with processing the *information content* of text. In this domain, we see operations such as ordering and searching text, and checking the spelling and grammar. On the other hand, the *glyph domain* on the right concerns itself with processes related to the *appearance* of the text. In this domain, we see operations to format the text, and to display and print it.

In the middle, we see the operations of glyph selection and substitution to transform characters into glyphs for presentation, and the layout operation to decide, for example, where to split the text at the end of a line. We also see operations from the glyph domain into characters. Character-recognition processes transform glyphs into characters. Mouse selection identifies the character under the mouse cursor.

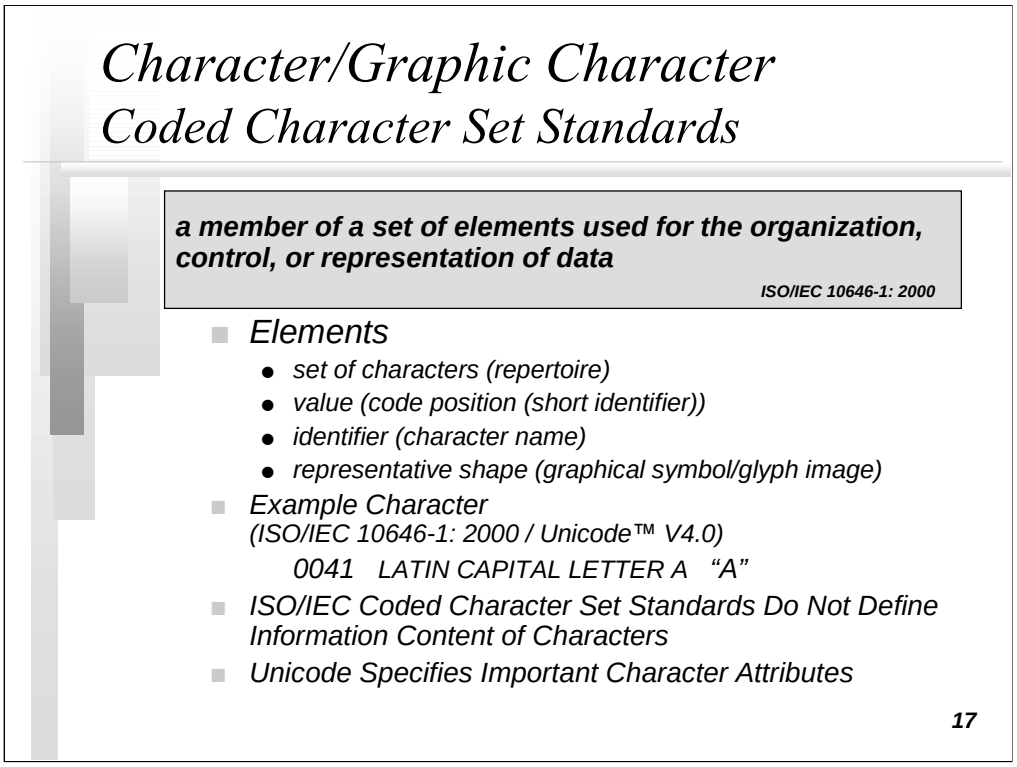
Notice that the two domains complement each other and the types of processes in one domain are distinct from those in the other domain.



Let's examine another model for the composition, layout, and presentation functions that you will find in a wordprocessor. This model, in some ways, is an overlay of the domains of previous figure. However, it is interesting because of an area of processing that is in both the character and the glyph domains.

First, lets examine the "Content-base Processing" area of the character domain. Here, we see processes concerned with the information content. The data entry process converts keystrokes into character codes. The character codes are, in turn, input to other information processes like sorting, searching, and spell checking and grammar checking. The character information is also the input to the composition and layout process that produces glyph identifiers as presentation information. Finally, the "Appearance-based Processing" area of the glyph domain takes the glyph identifiers and displays or prints them.

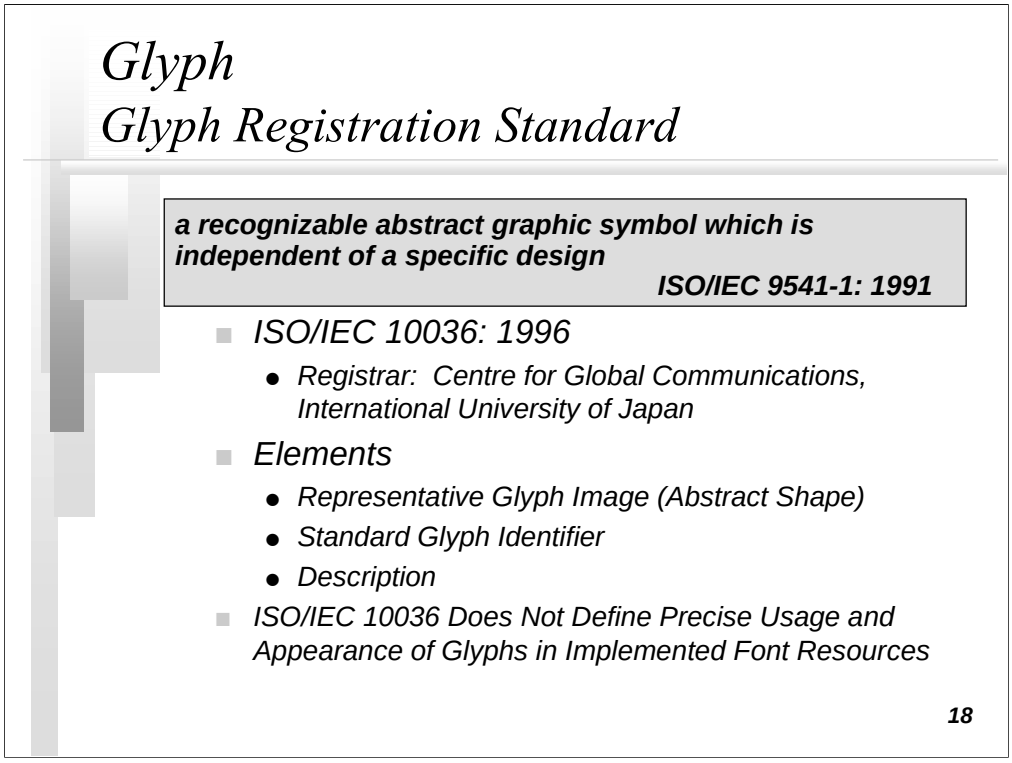
What is interesting is the area labeled "Both" because processes here must be aware of both domains. The composition and layout process uses glyph metrics to know the next position on the line (to know when it reaches the end of a line) and how far down to position the start of the next line from the previous line of text. However, a hyphenation or line-splitting subprocess uses the information domain to know where the word and syllable boundaries are.



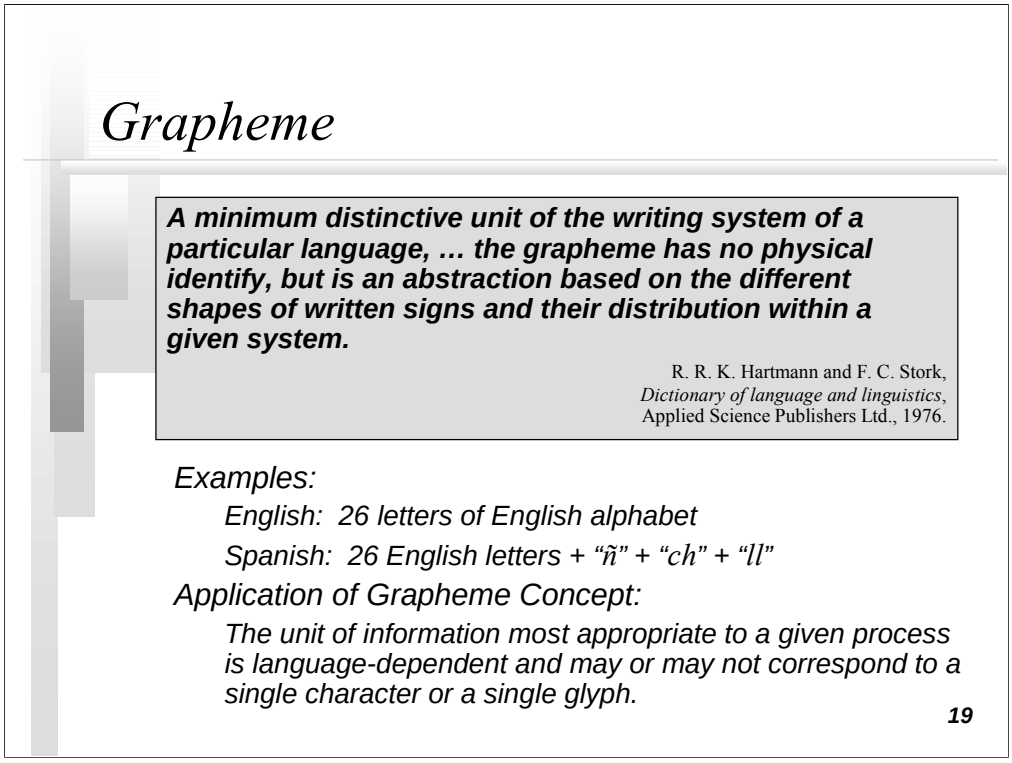
Having seen the model for the character and glyph domains, let's turn to the relevant standards in each domain.

In the character domain, we have coded-character-set standards. ISO defines a *character* as "a member of a set of elements used for the organization, control or representation of data". However, we are concerned with graphic characters rather than control characters. Coded-character-set standards have the following elements: a set of characters (the repertoire), and for each character in the set, code or value, a name, and a representative shape (glyph). For example, in Unicode and ISO/IEC 10646-1, the code 0041 represents LATIN CAPITAL LETTER A, which has a representative shape of "A".

ISO coded-character-set standards do not define the information content of a character. However, Unicode specified important character attributes to aid implementation.



For the glyph domain, ISO defines a *glyph* as "a recognizable abstract graphic symbol which is independent of a specific design". ISO has the 10036 standard that describes the ISO glyph registry. Glyphs registered in the 10036 registry have the following elements: a representative shape (which is an abstract shape), a standard glyph identifier, and a description of the glyph. In addition, the standard does not define the precise usage and appearance of glyphs that are implemented in a font (resource). AFII, the Association for Font Information Interchange, was the first Registrar for the glyph registry, but AFII ceased operations in September, 1999. Now, the Centre for Global Communications of the International University of Japan is the Registrar for the glyph registry.



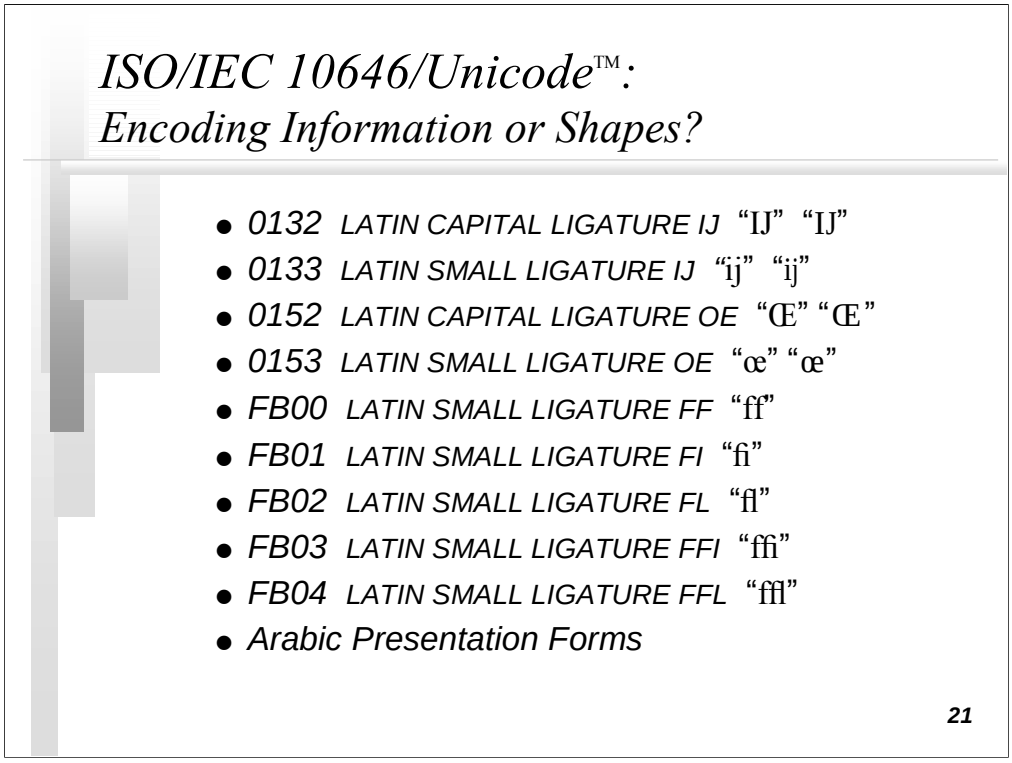
We need to introduce the linguistic concept of a grapheme. A *grapheme* is "the minimum distinctive unit of the writing system of a particular language". In English, the 26 letters of the English alphabet are graphemes. In Spanish, the 26 English letters plus the "ñ" plus the two digraphs, "ch" and "ll" are graphemes. This linguistic concept can be extended into information systems to say that the unit of information most appropriate to a given process is language-dependent and may or may not correspond to a single character or a single glyph. Applying this concept, an implementer may decide to transform character codes or glyph identifiers into a different encoding to optimize a particular process. An example of this is the transformation of a character code into weights for correctly sorting text data according to a particular culture. See *Unicode Technical Standard #10, Unicode Collation Algorithm* at www.unicode.org.

Use of the Character-Glyph Model in Standards

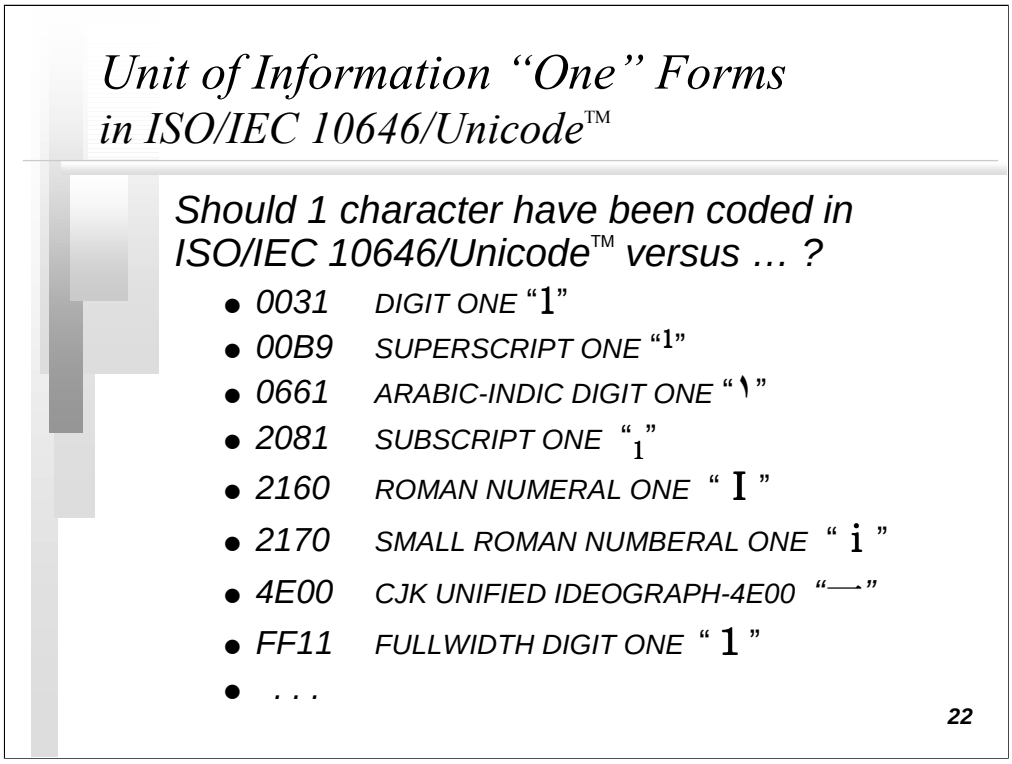
ISO/IEC 10646-1: 2000,
ISO/IEC 10646-2: 2001,
Unicode Version 4.0,
and the ISO/IEC 10036: 1996 Glyph Registry
generally follow the Character-Glyph model
—but not completely.

20

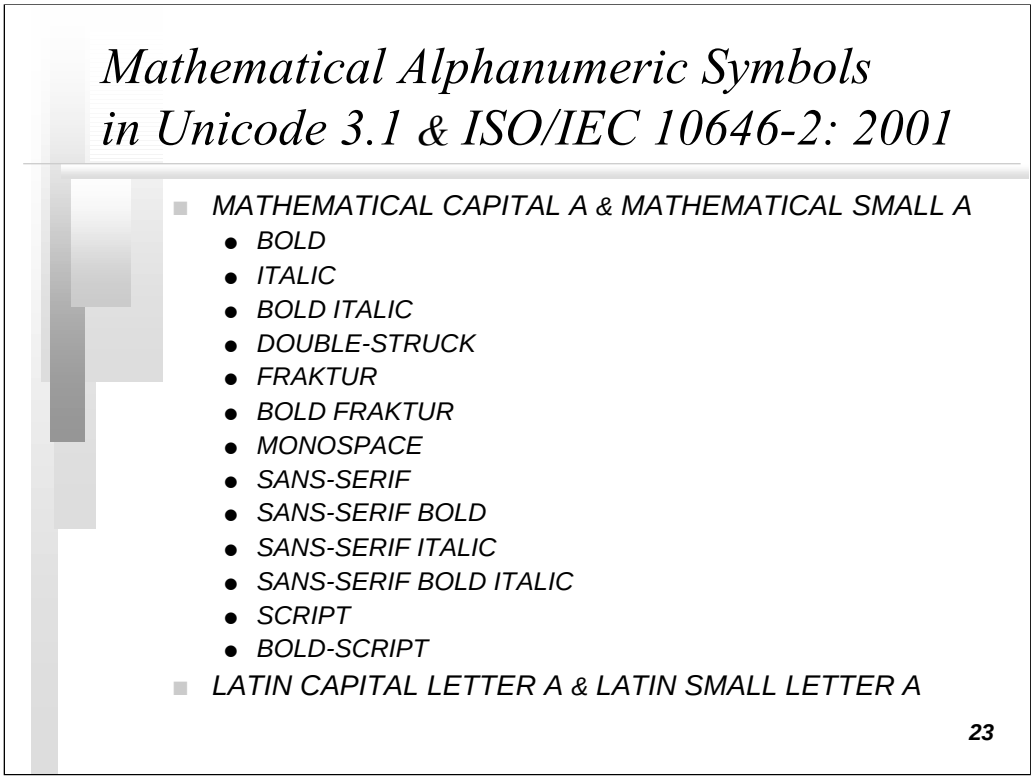
How well are the principles found in this Character-Glyph model followed in the ISO/IEC 10646-1, ISO/IEC 10646-2, and the Unicode Version 4.0 standards, and in the ISO/IEC 10036 Glyph Registry? For the most part, they follow the model. However, each has exceptions and the next few figures illustrate some of them.



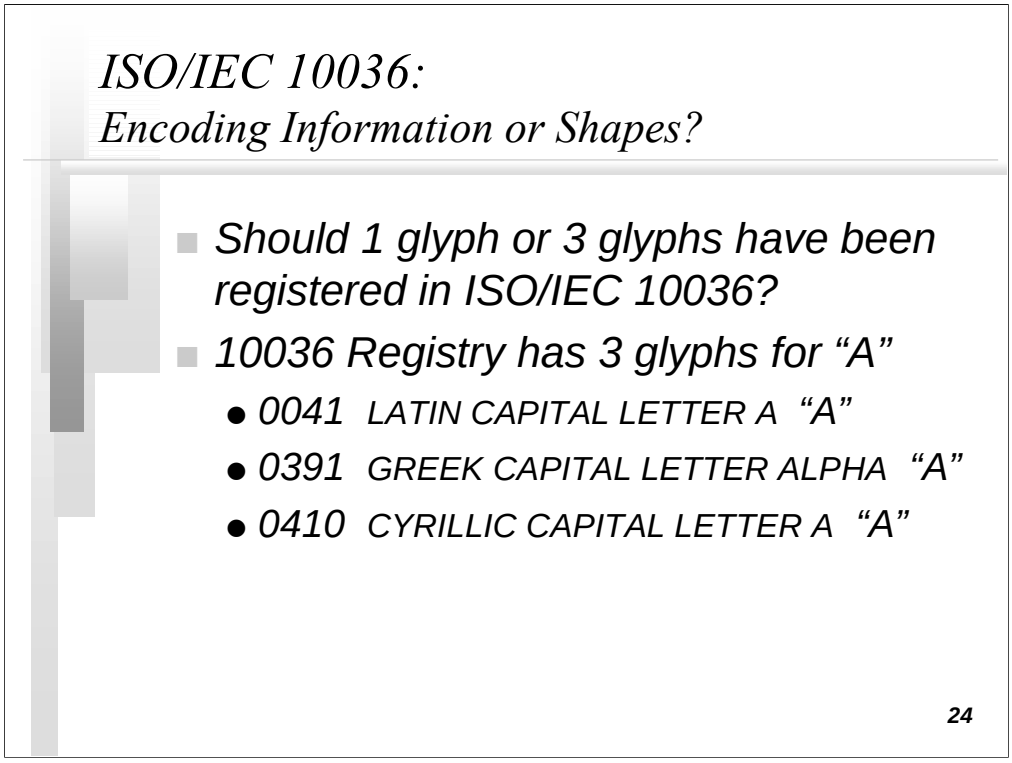
First, ISO/IEC 10646 and Unicode code several ligatures. Here, you can see ligatures for "IJ", "OE", and the various "FF", "FI", and "FFI" and "FFL" ligatures. In addition, ISO/IEC 10646 and Unicode code presentation forms and an extensive set of ligatures for Arabic letters. However, these were encoded as part of the merger of 10646 and Unicode for compatibility with existing coded-character-set standards. According to the principles of the Character-Glyph Model, characters for these ligatures and presentation forms should not have been encoded because they are needed only for rendering the characters.



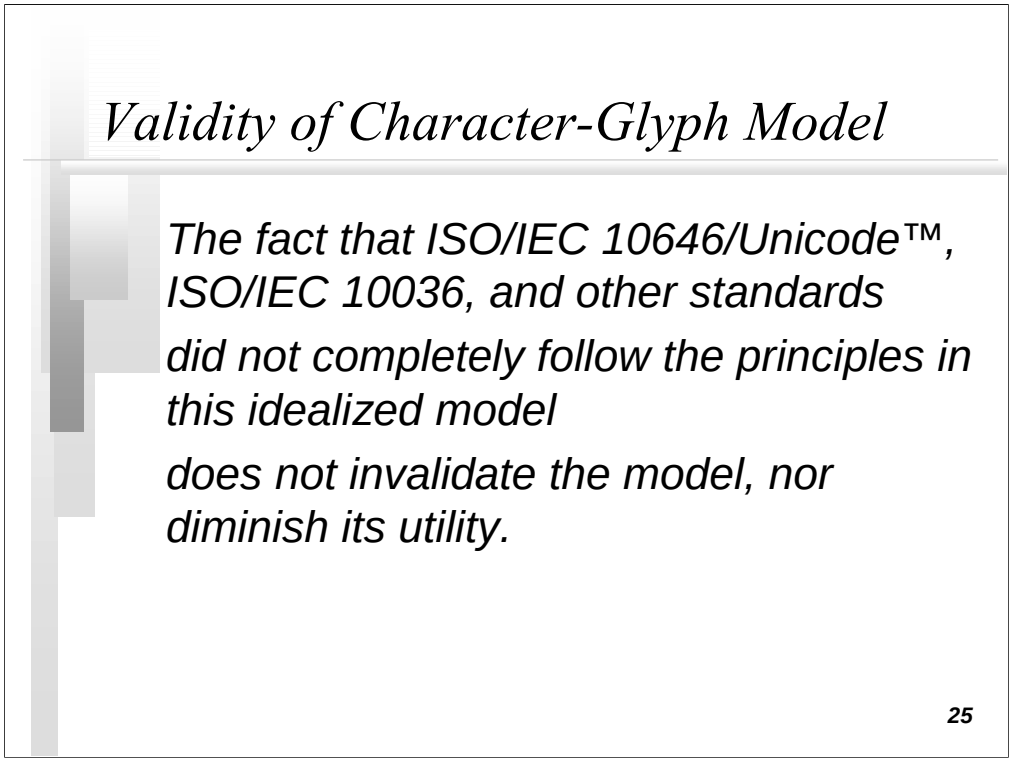
Similarly, could coding one character for the concept of "one" or "unity" been sufficient in 10646 and Unicode? The ISO/IEC Technical Report lists about 30 different characters that are encoded to represent this concept. Some of the characters are different forms for the same script (DIGIT ONE, SUPERSCRIPT ONE, SUBSCRIPT ONE, FULLWIDTH DIGIT ONE); others are different forms for the various scripts. Don't all of these forms encode the same information?



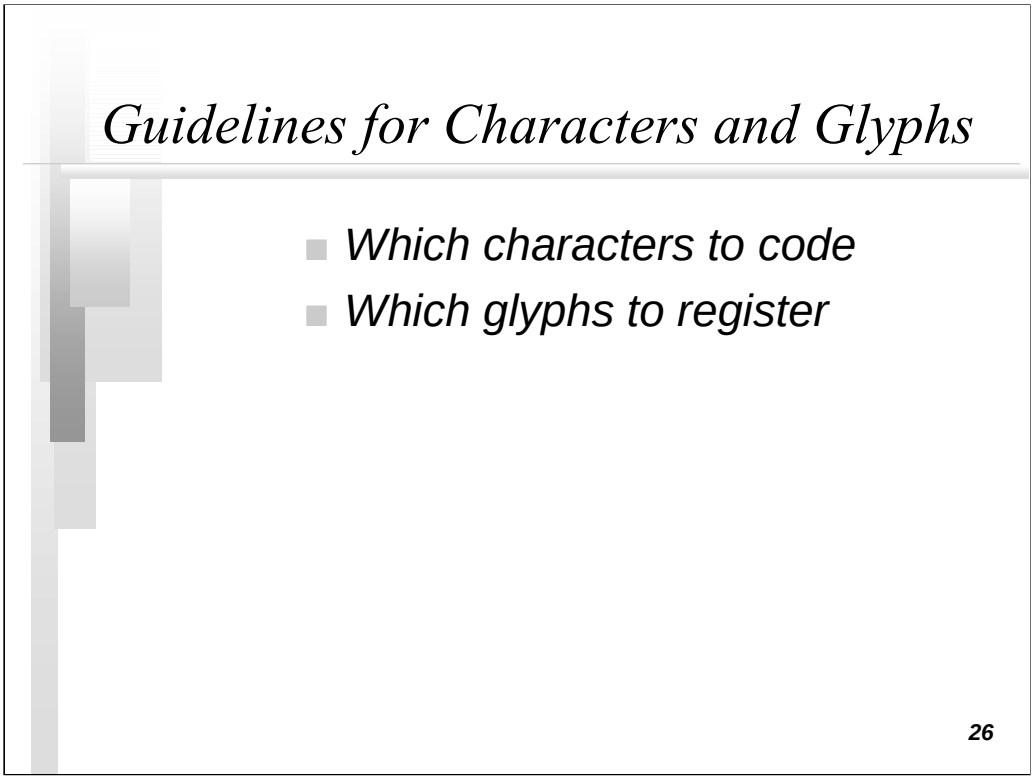
Unicode 3.1 and ISO/IEC 10646-2:2001 added a block of mathematical alphanumeric symbols. Mathematicians and scientists have the requirement to differentiate different forms of letters in plain text because such forms conveyed different meaning to them. For example, variables are shown in italic, matrix names are shown by bold capital letters. After considering several alternatives, ISO decided to code them as separate characters but to differentiate them as "Mathematical Alphanumeric *Symbols*" from the normal letters and digits. Unicode assigns different properties to the normal letters and digits and the Mathematical Alphanumeric Symbols.



Let's switch our focus to the Glyph Registry for ISO/IEC 10036. Should 3 glyphs have been registered for the shape needed for the LATIN CAPITAL LETTER A, and the GREEK CAPITAL LETTER ALPHA, and the CYRILLIC CAPITAL LETTER A? Are not all of these the same abstract shape?

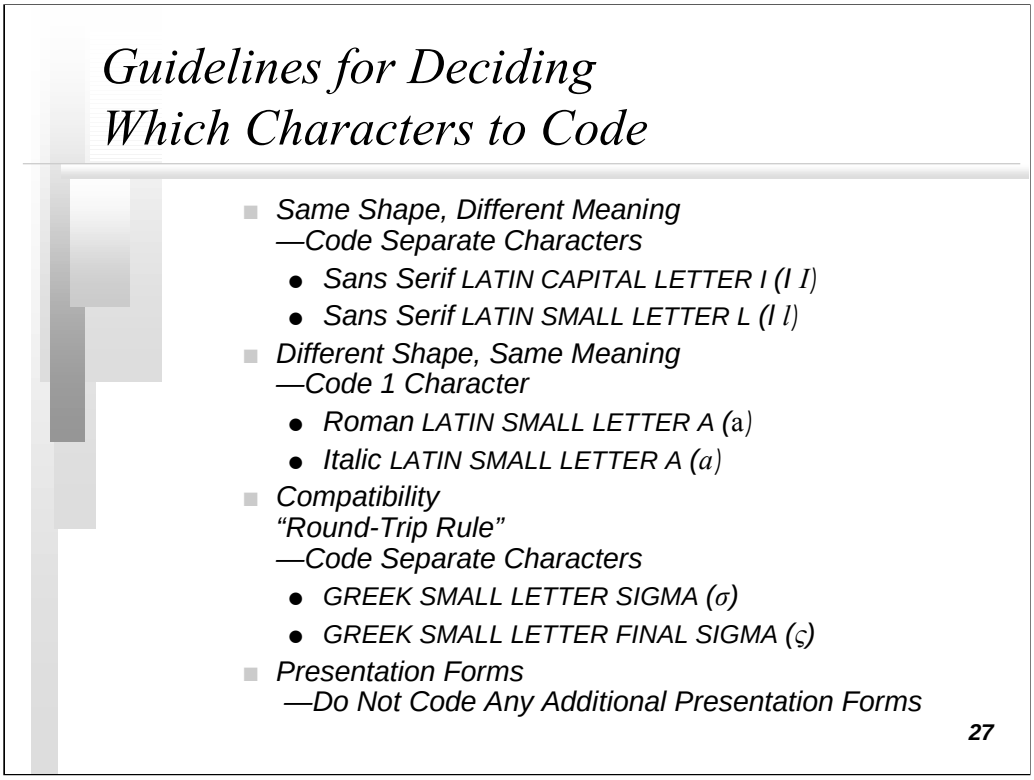


Even though the standards did not completely follow the principles in the idealized Character-Glyph Model, this does not invalidate the model, nor diminish its utility. The model is very important to understanding the concepts of a "character" and a "glyph", and how to use them to render characters into glyphs.



Given that ISO did not follow the model in developing ISO/IEC 10646 and the 10036 Registrar did not follow them for registering glyphs, can we derive any benefit from this model? In fairness to ISO, it could not be expected to follow the principles in this report when, in fact, the first draft of what was to become ISO/IEC Technical Report 15285 was not written until after the first edition of ISO/IEC 10646-1 and the Unicode Standard, Version 1.0 were published.

Given the existence of the model, does it provide ISO any guidelines for deciding which characters to encode, and which glyphs should be registered? The answer is that it does provide such guidance.



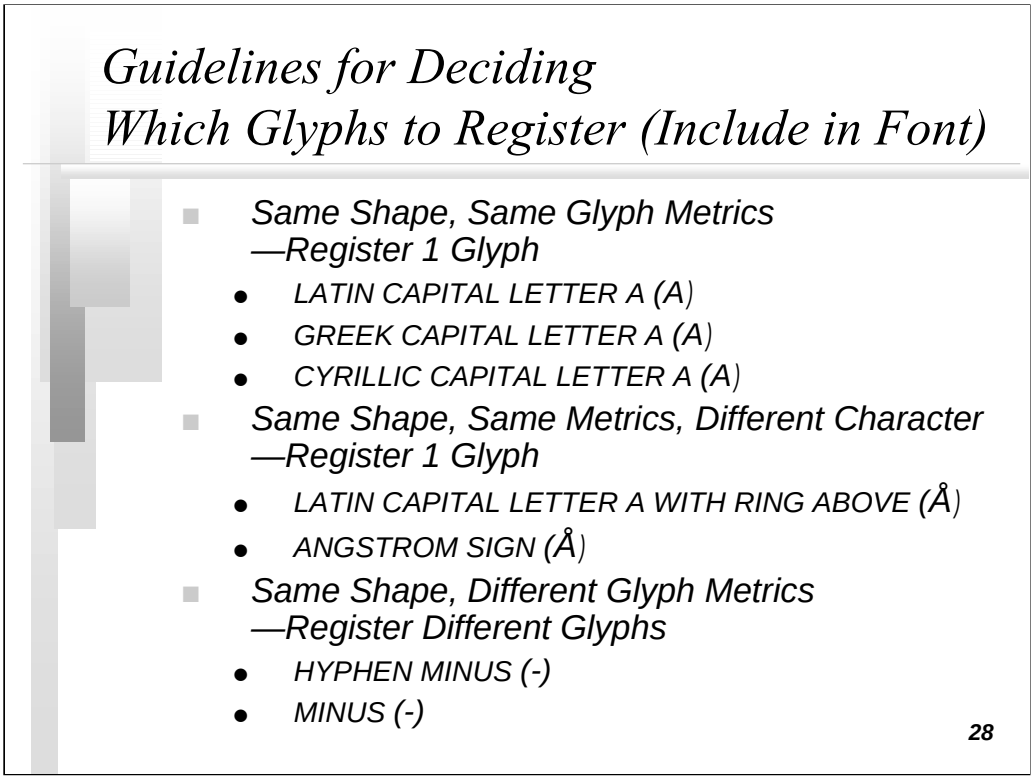
Based on the theory of the character-glyph model just described, here are some guidelines for deciding which characters to code.

If two glyphs have the same shape but different meaning, then code separate characters. An example of this is two characters that appear the same in a sans serif font, the LATIN CAPITAL LETTER I and the LATIN SMALL LETTER L.

If two glyphs have different shapes but the same meaning, then code one character. An example of this occurs in the Roman (upright) font and the Italic (slanted) font for the LATIN SMALL LETTER A.

When one of the source codes for Unicode and 10646 includes separate characters, encode separate characters for compatibility ("Round-Trip Rule: A character in a source code must map to a Unicode/10646 character and the Unicode/10646 character must map back to the character in the source code). ISO/IEC 8859-7, Latin/Greek is one of the source codes for Unicode and ISO/IEC 10646. Since this code has separate characters for GREEK SMALL LETTER SIGMA and GREEK SMALL LETTER FINAL SIGMA, Unicode and ISO/IEC 10646 have separate characters as well.

Do not code any additional presentation forms. Unicode and ISO/IEC 10464 do not need to code new characters for glyph variations and ligatures. These belong in the glyph domain in fonts.

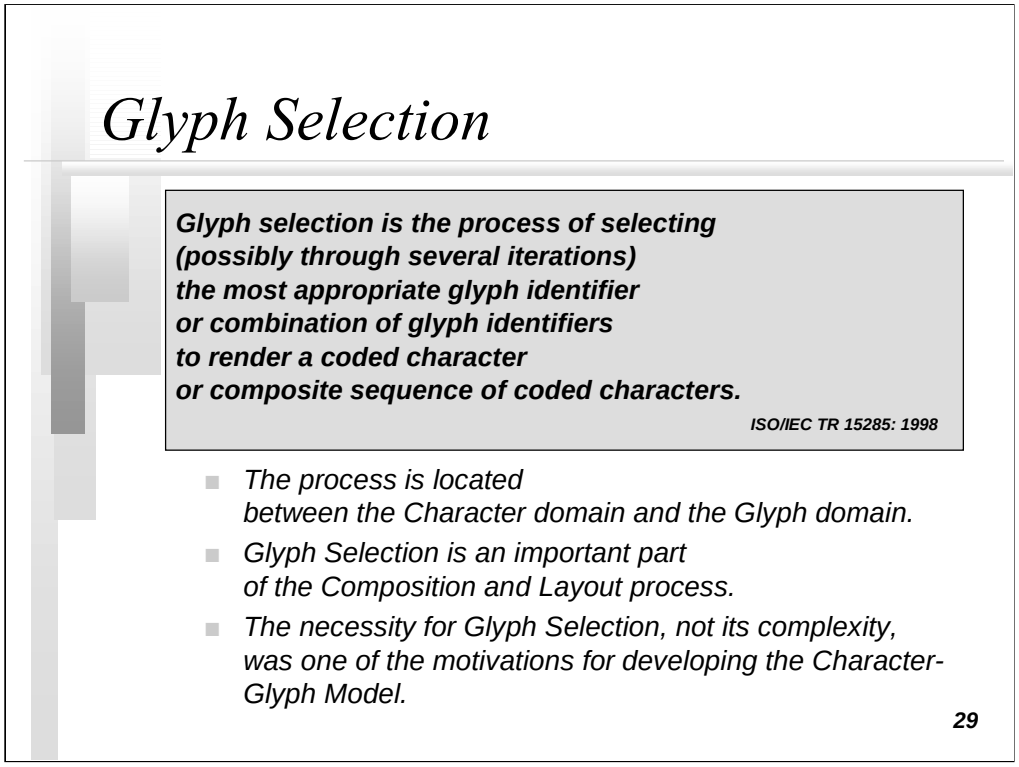


Based on the theory of the character-glyph model just described, here are some guidelines for deciding which glyphs to register or include in a font.

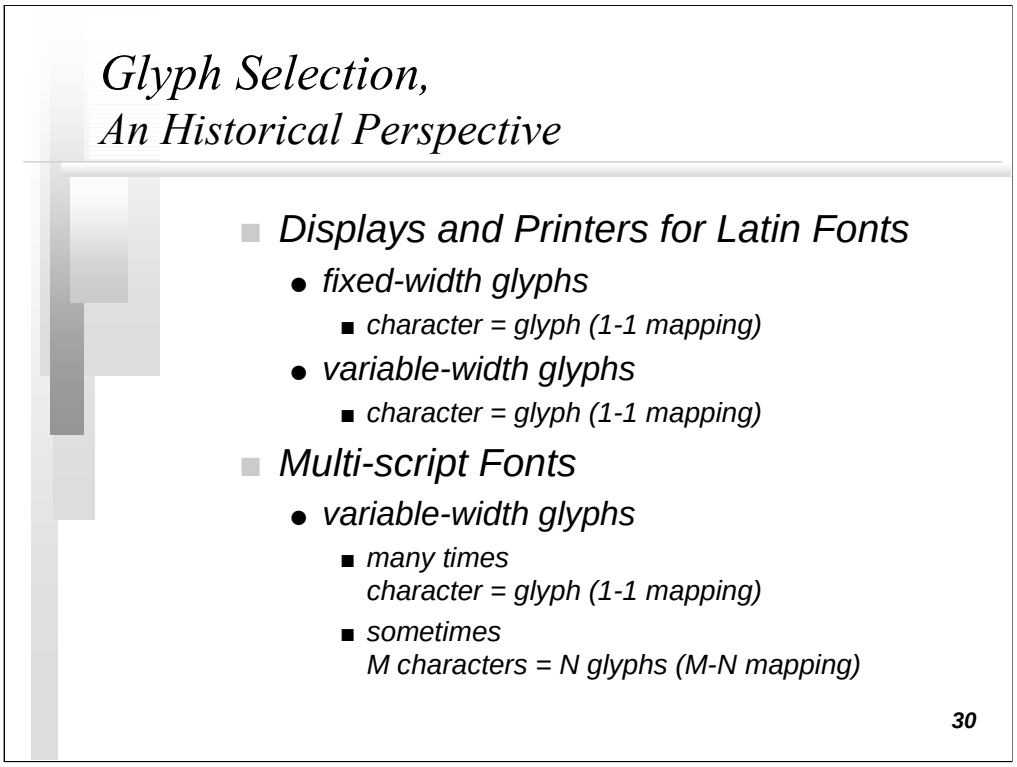
If two glyphs have the same shape and the same glyph metrics (e.g., spacing before or after), then include only one glyph.

If two glyphs have the same shape and the same glyph metrics, but are mapped from different characters, then still include only one glyph. The fact that two different characters map into the same shape with the same glyph metrics does not require a separate glyph in a font.

If two glyphs have the same shape, but have different glyph metrics, then include different glyphs in the font.



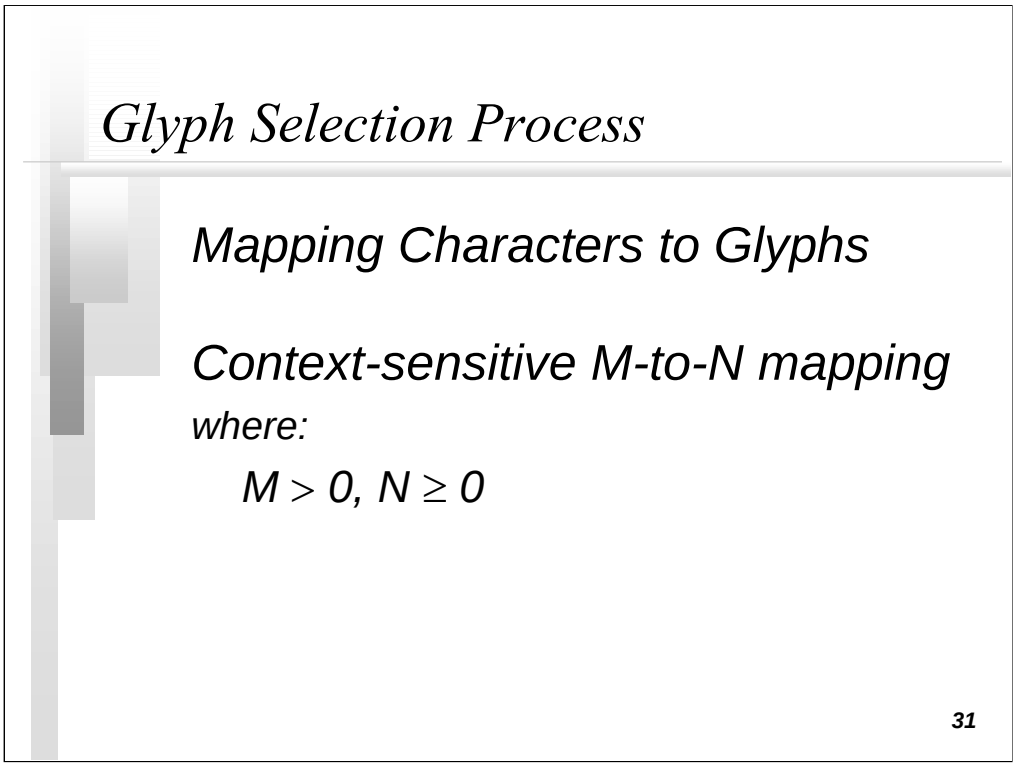
Quoting from the Technical Report (ISO/IEC TR 15285), "Glyph selection is the process of selecting (possibly through several iterations) the most appropriate glyph identifier or combination of glyph identifiers to render a coded character or composite sequence of coded characters." The process is located between the Character domain and the Glyph domain and converts coded character into glyph identifiers. This process is an important part of the composition and layout process. Moreover, the necessity for glyph-selection, not its complexity, was one of the motivations for developing the Character-Glyph Model.



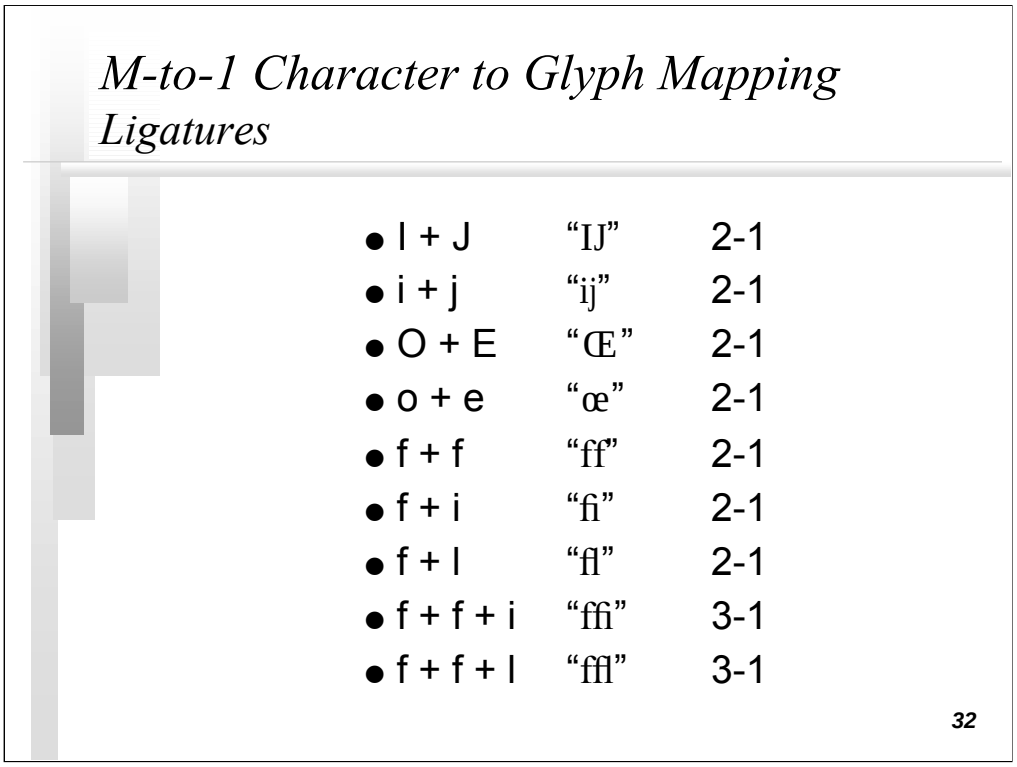
One of the reasons for the confusion is typewriter and the historical development of printing from computers. When typing on a typewriter, striking one key created one and only one glyph on the paper. Because computer input is based on typewriter technology, many of our misunderstandings of characters and glyphs derive from this origin.

In the early days of computing, no one distinguished a character from a glyph. Printers would only print digits, English letters and a few symbols. (In fact, it was rare to have a printer that would print both upper-case and lower-case characters.) At this time, the mapping between characters and glyphs was one-to-one and defined by the coding of the characters. At that time, no one conceived of any need to distinguish characters and glyphs—we thought that they were the same thing! This concept prevailed even as the vendors delivered more sophisticated printers that would print variable-width, proportionally-spaced "characters" (i.e., glyphs).

At this point, for the most part the one-to-one relationship between characters and glyphs still holds for the majority of characters in 10646 and Unicode. However, sometimes the relationship fails to render the characters correctly; sometimes something more sophisticated is needed to render the characters correctly.

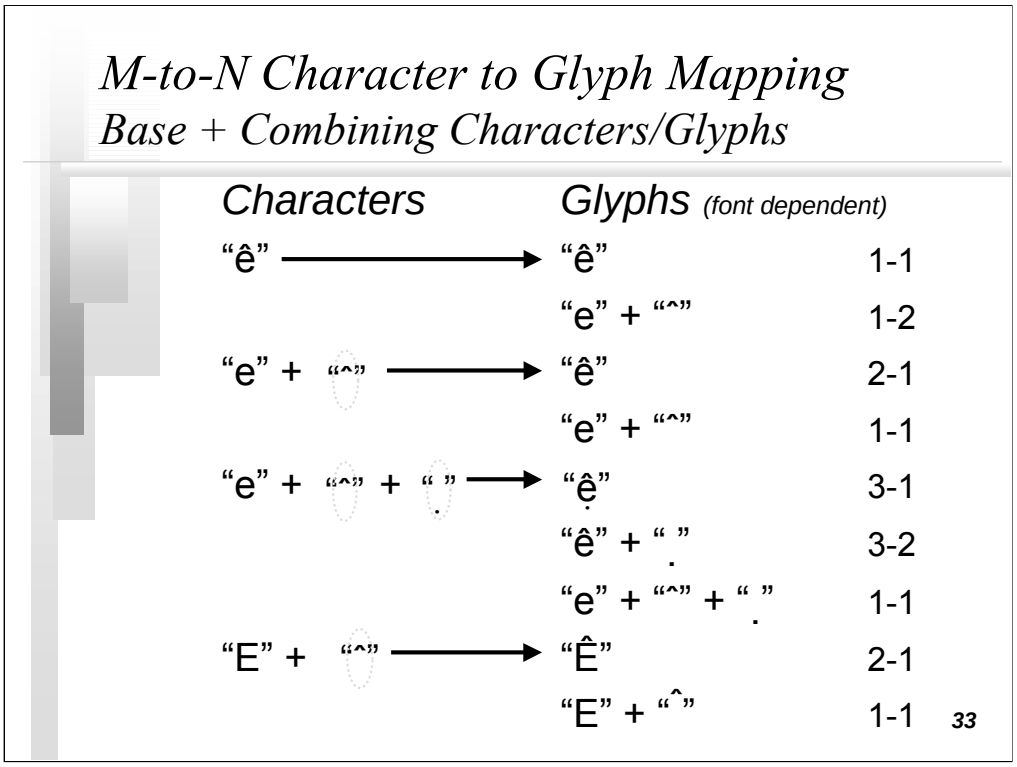


Sometimes the glyph-selection process for 10646 and Unicode characters needs to be extended from a one-to-one mapping. In the general case, glyph selection is a context-sensitive mapping of *M* characters into *N* glyphs. The next few figures will show examples of the general case.

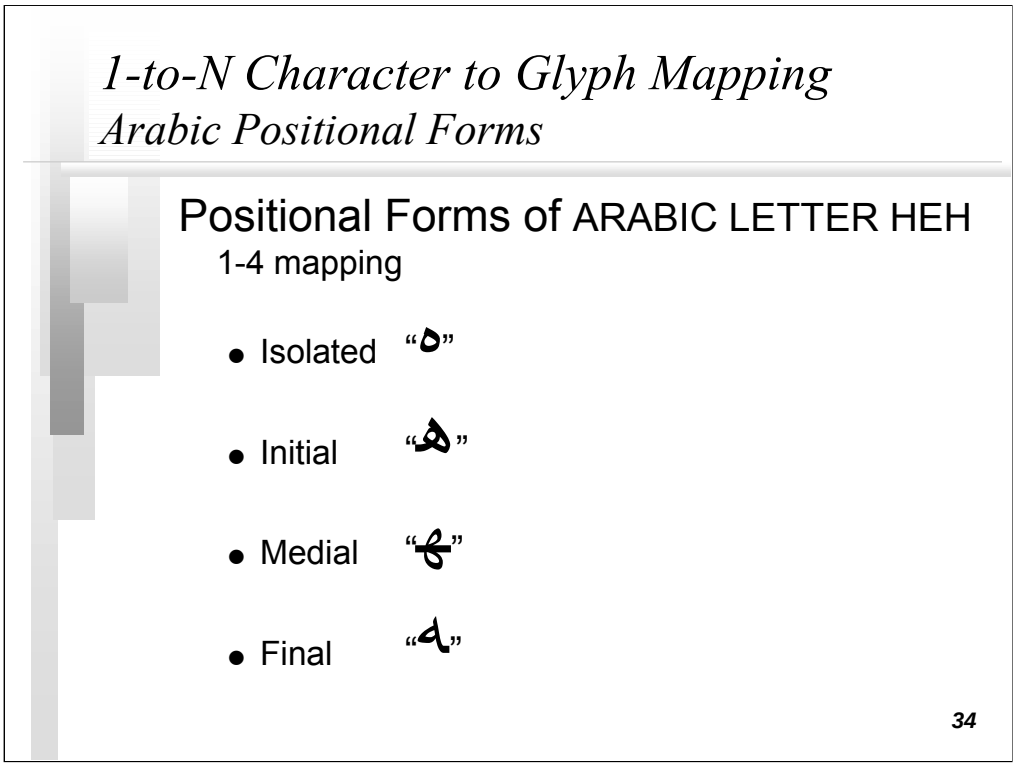


What you see here are several examples of an M-to-1 one glyph mapping. This occurs with ligatures used in the Latin script. The "IJ" ligatures could easily be implemented with a Kerning table. (A "Kerning table" in a "font" describes the spacing required between each pair of glyphs.) If fact, the illustrated glyphs in the figure are Kerned "I" and "J" glyphs from the Baskerville Old Face font rather than a true "IJ" ligature glyph.

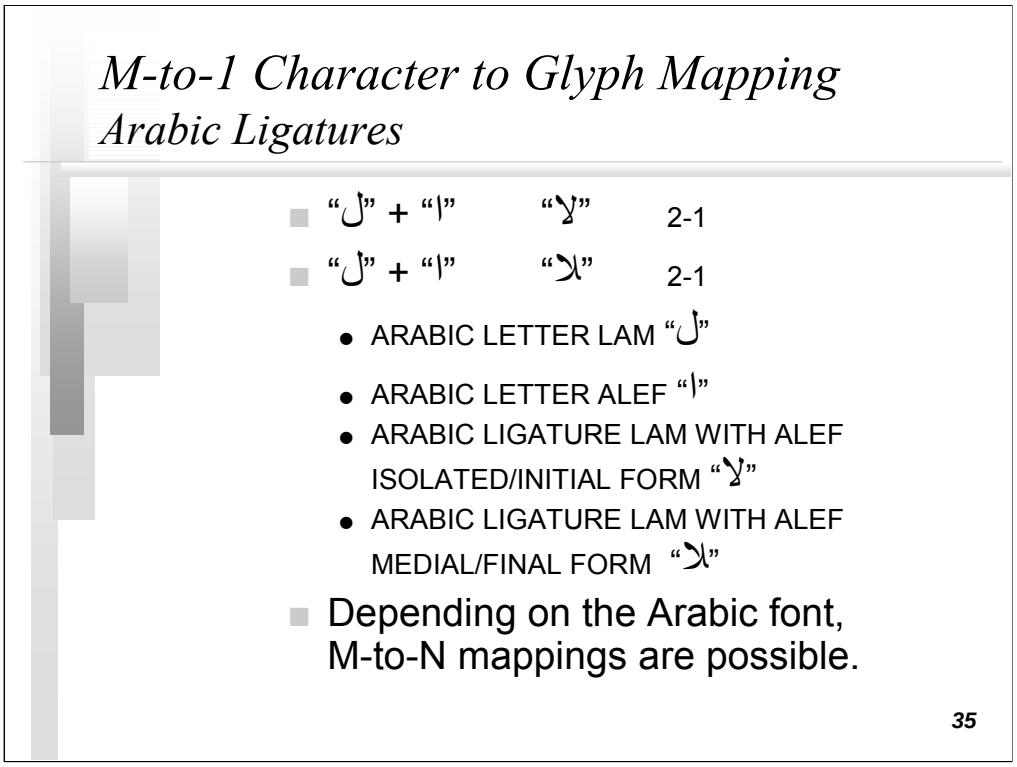
For mapping into the "OE" ligatures in French, the glyph selection process would need to know the context when the "Œ" ligature glyph is used instead of separate "O" and "E" glyphs. The last 5 ligatures illustrate various forms of the "FI" and "FL" ligatures. Note that the last two represent a 3-to-1 mapping. Also note that these ligatures occur more frequently in a serif font (as illustrated in Times New Roman) rather than a sans serif font.



Depending on the particular font implementation, the combination sequences of 10646 and Unicode may require more sophisticated mapping. Recall that a combining sequence consists of a base character followed by one or more combining characters. Combining sequences are frequently used to code accented characters. Recall also that many accented characters may be represented in 10646 and Unicode as either a single character or a combining sequence. The illustrated LATIN LETTER E WITH CIRCUMFLEX is one of these. Note that just as the accented characters may be encoded as a single character or a combining sequence in 10646 and Unicode, a font designer may choose to implement accented characters as either a single glyph or multiple glyphs that correspond to an encoded combining sequence. This figure illustrates several possible implementations and the corresponding mapping. Note that for the placement of the COMBINING CIRCUMFLEX ACCENT glyph needs to be higher for the LATIN CAPITAL LETTER E glyph than for the LATIN SMALL LETTER E glyph.

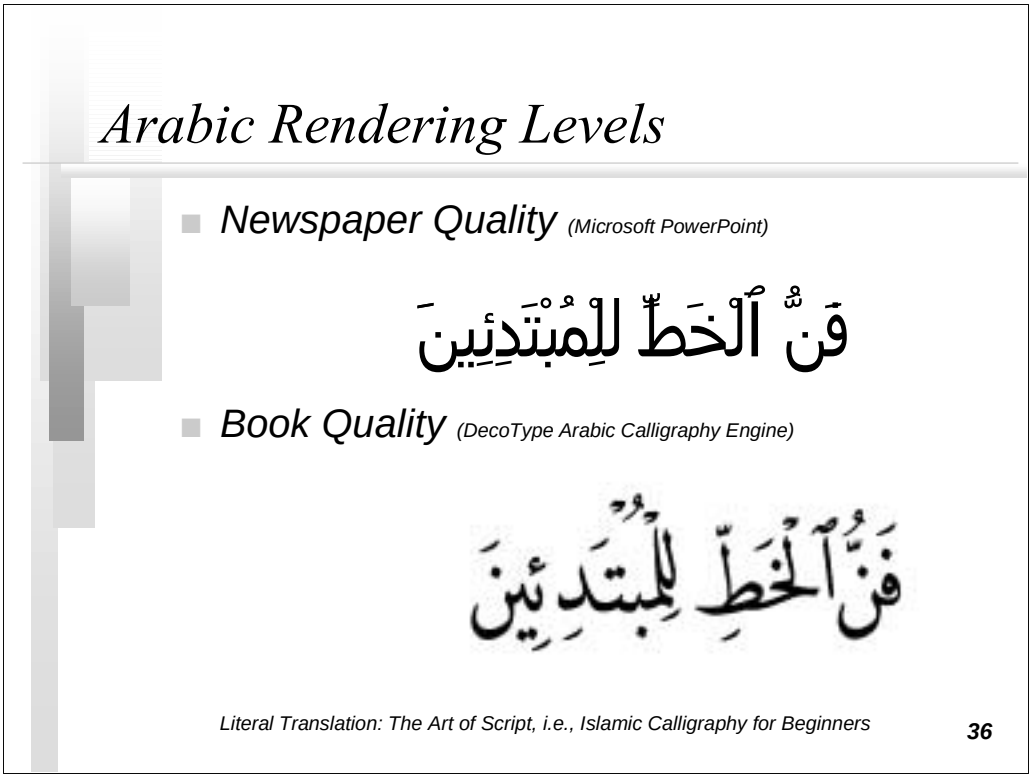


Recall that Arabic is a cursive script where the characters in words are connected. Minimum legibility of the Arabic script requires a 1-to-4 mapping to account for when the letter is alone (isolated form), at the beginning of a word (initial form), in the middle of a word (medial form), or at the end of a word (final form). This figure illustrates four forms of the ARABIC LETTER HEH. (However, Thomas Milo indicated that a fifth form, which looks like the initial form without the trailing connector, is required for enumeration to distinguish it from the ARABIC-INDIC DIGIT 5 "٥".)



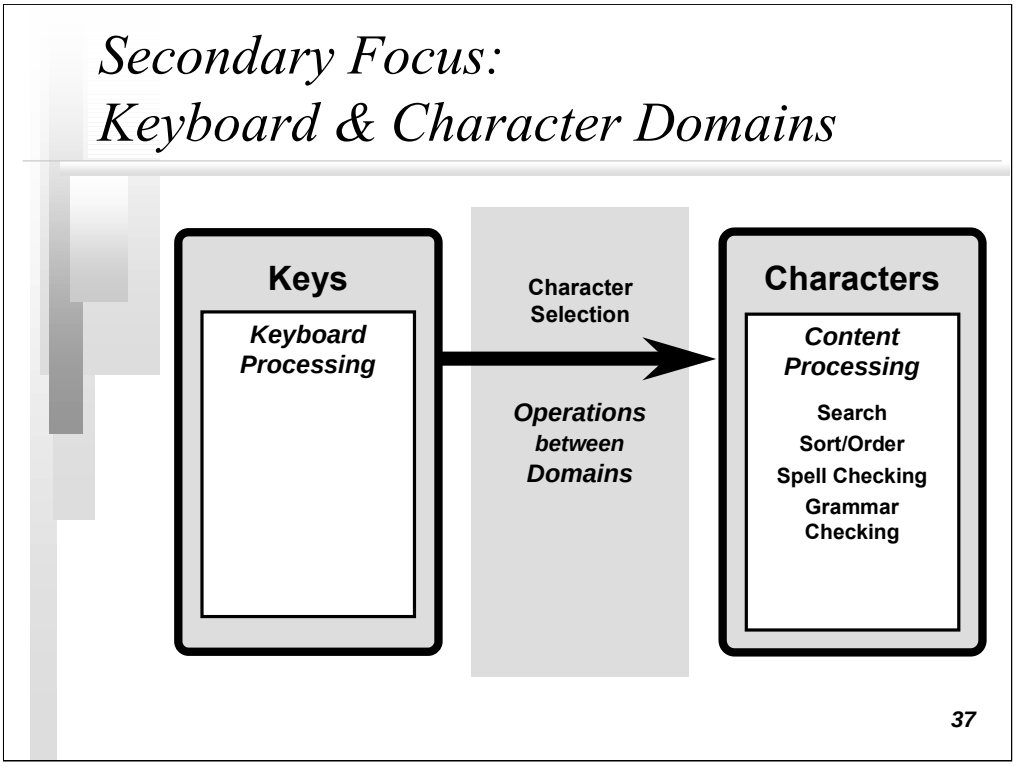
Arabic fonts also require glyphs for the LAM WITH ALEF ligature. Note that like the Arabic letters, this ligature has forms that depend on the position of the two Arabic letters in a word.

To obtain minimum legibility for the Arabic script, a font must contain all of the presentation forms for the Arabic letters plus the two LAM WITH ALEF ligature. However, this is the minimum that might be used, for example, in an Arabic newspaper. Rendering Arabic for books, requires that the fonts contain an extensive set of Arabic ligatures which, in turn, requires a much more sophisticated glyph selection process than the one required for minimum legibility of Arabic.



Rendering Arabic has an additional consideration, the level of quality of the rendering. Recall that Unicode is designed for plain text, that is text without formatting information. Implementations of Unicode need to legibly render the Unicode characters. Many implementations of Arabic using Unicode produce what Thomas Milo calls "newspaper" quality rendering. Such rendering provides a relatively low level of legibility to people familiar with Arabic. However, being a calligraphic script, Arabic readers expect a higher quality of rendering in books. In the *Koran*, they expect rendering of the highest quality. The highest quality Arabic rendering needs to approach the rendering level of hand-drawn calligraphy as would be done by a scribe. The ultimate goal of Thomas Milo is build the technology to deliver the highest level of rendering Arabic.

I thank Thomas Milo (DecoType) for providing these examples and the translation.



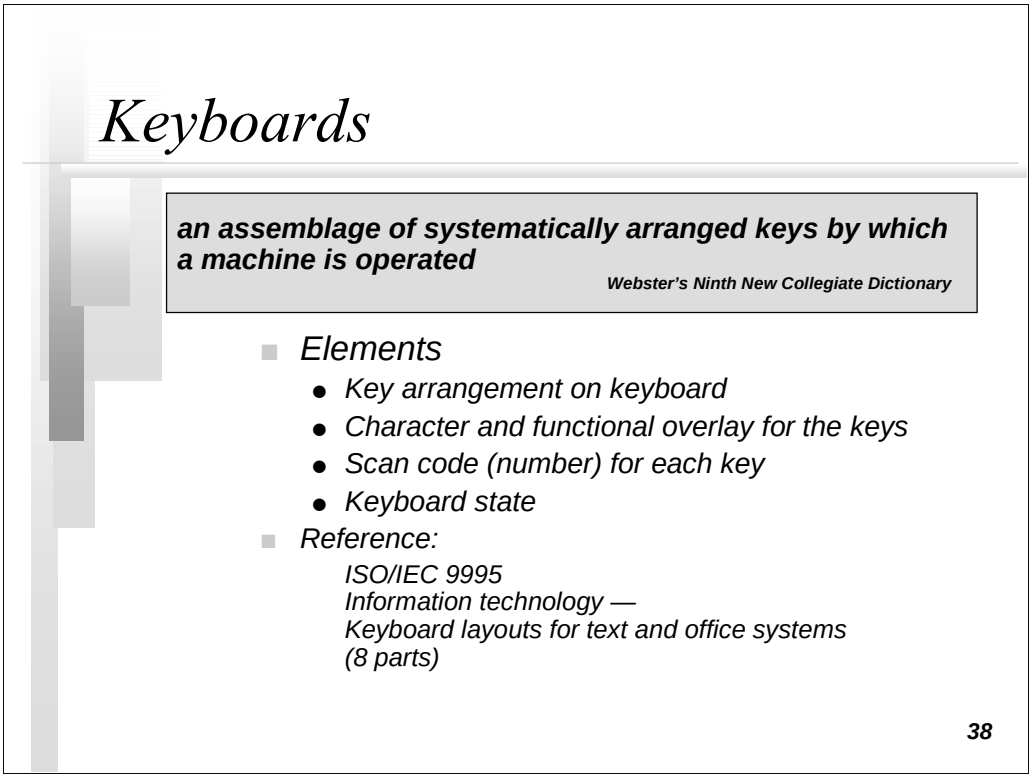
This figure shows two separate domains, the keyboard key domain on the left and the character domain on the right. It also uses an arrow for the character selection operation for going from the key domain to the character domain.

The *key domain* on the left concerns itself with processing keyboard and keyboard scan codes. In this domain, the operation is converting keystrokes into keyboard scan codes.

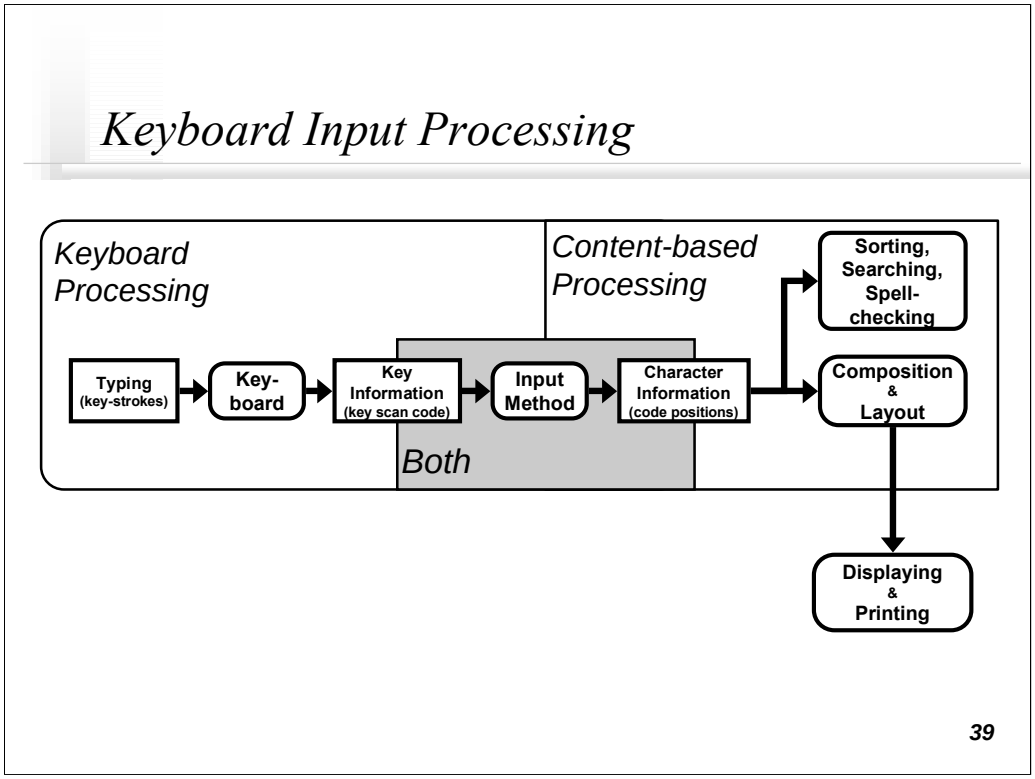
Earlier, we discussed the character domain with its operations on the character content.

In the middle, we see the character-selection operation that transforms keyboard scan codes into characters. An implementation of the transformation is frequently called an "input method". Unlike the character and glyph domains, these domains have no reverse operation to convert characters into scan codes.

Like the earlier discussion of the character and glyph domains, notice that the two domains complement each other and the types of processes in one domain are distinct from those in the other domain.



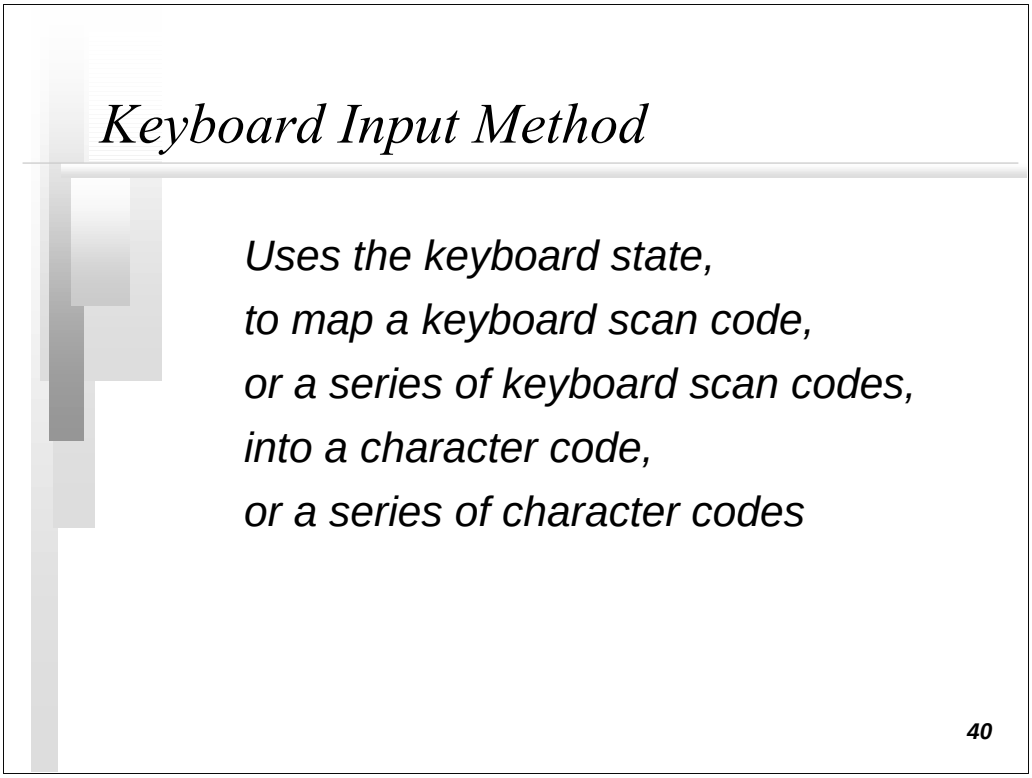
We are all intimately familiar with using keyboards, but do we know how they work? This information will be new to some of you and a review for the rest. The keyboard arrangements or keyboard layout is the geometry of how the keys are arranged on the keyboard. Over each key is some printed description of what the key is supposed to do, for example, to produce a character, control the keyboard state, or indicate a function. Keyboard layouts are fairly consistent now. Associated with each key is a scan code. The keyboard also has different shift states (e.g., due to the "Shift", "Ctrl", "Alt", "Caps Lock", "Num Lock" or "Scroll Lock" keys on a US English keyboard).



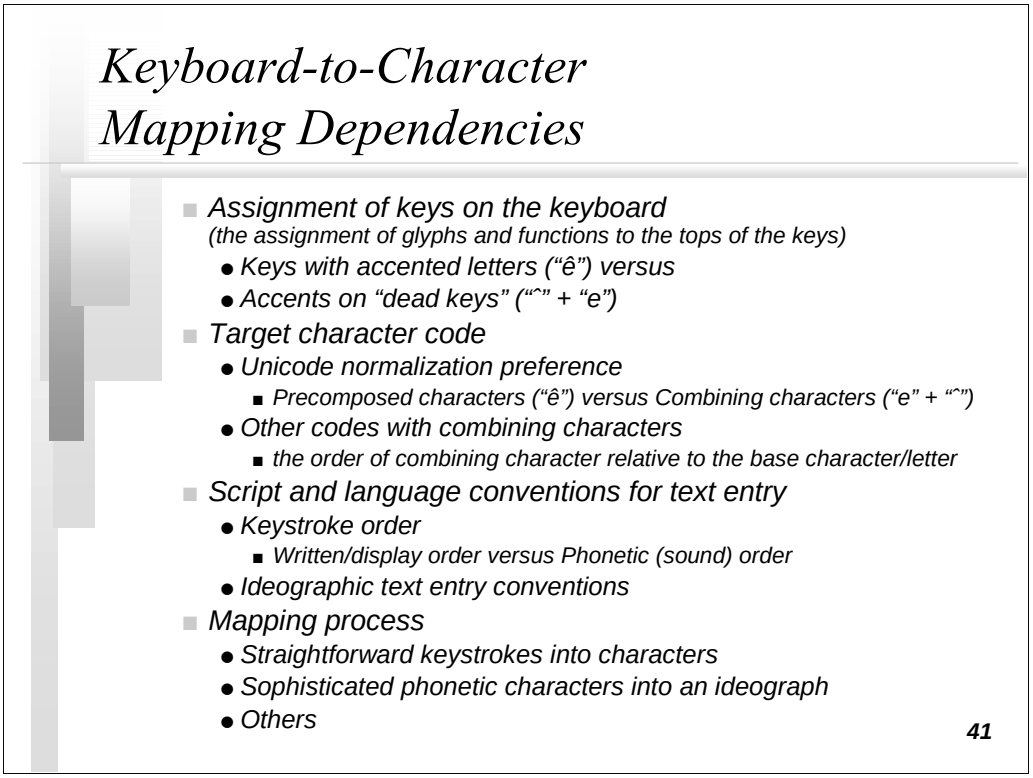
Let's examine a model for keyboard input. This model, in some ways, is an overlay of the Key and Character domains in the earlier figure. Note the input method area of processing is in both the key and the character domains.

Now, let's examine the keyboard input process. First, someone strikes a key on the keyboard. The keyboard transforms that action into a keyboard scan code or possibly a change of state for the keyboard (e.g., typing the "Caps Lock" key). In the "Keyboard Processing" area, this is all keyboard information. The input method process converts the keyboard scan codes into character codes (code positions). The character codes are, in turn, serve as input to other information processes in the "Content-based Processing" area.

Like the earlier text-processing model, the area labeled "Both" is interesting because the input method process here must be aware of both domains. The input method process must know both the keyboard layout (e.g., US English, French Canadian, Japanese) in the Key domain and the character coding (Unicode, ISO/IEC 8859-1, ASCII) used for the characters in the Character domain to properly convert the scan codes into character codes in the Character domain.

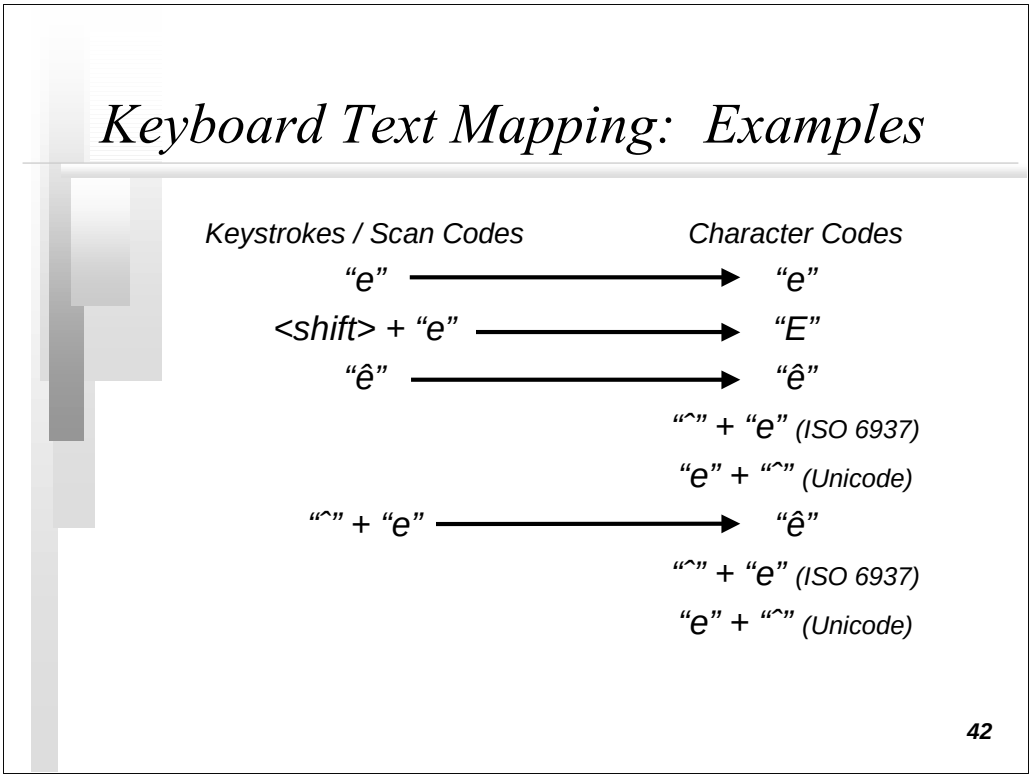


The input method process must know both the keyboard layout (e.g., US English, French Canadian, Japanese) in the Key domain and the character coding (Unicode, ISO/IEC 8859-1, ASCII) used for the characters in the Character domain to properly convert the scan codes into character codes in the Character domain. It then takes the state of the keyboard to map one or more keyboard scan codes into one or more character codes.



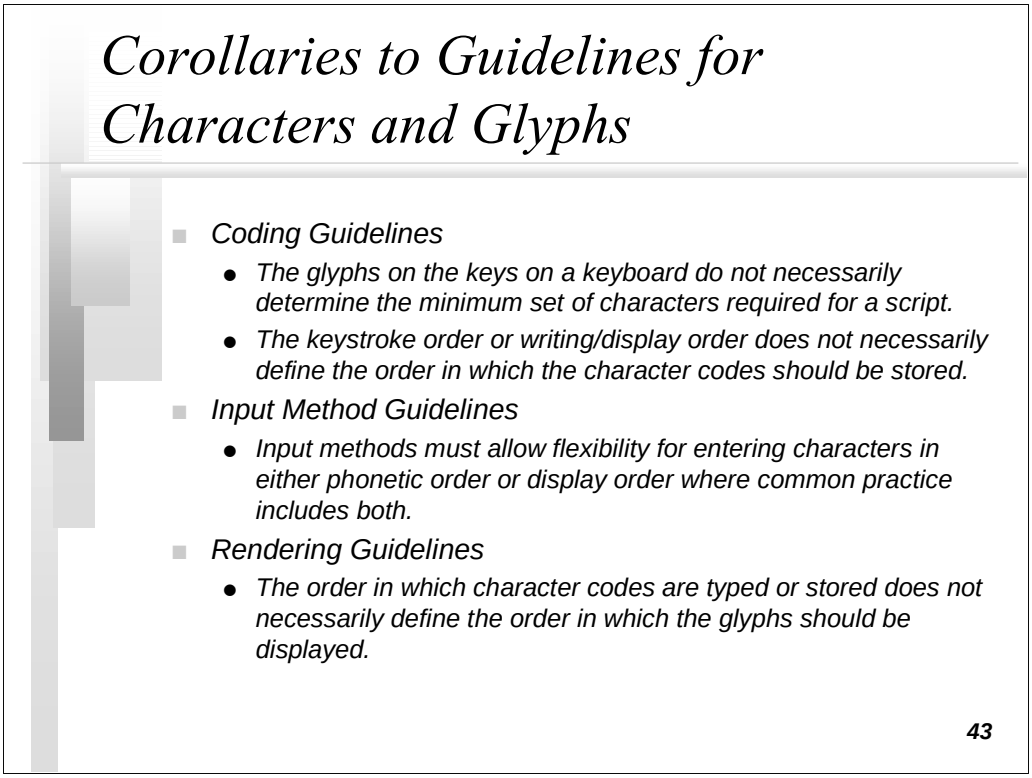
Several things determine the functions the input method must perform.

1. The first of these is how the assignment of keys to the keyboard and whether "dead keys" are used.
2. The second is the target character code. If it is Unicode, different processing is required if the preference is for precomposed characters versus using combining characters and a base character. If the target is another code that uses combining characters, processing will be required to place the combining characters in the proper order relative to the base character.
3. The third is the script or language conventions for typing the characters. This is an issue for some scripts where the display order is different from the phonetic order. Moreover, sometimes for the same script in the same area, the input method needs to allow the user to choose between display-order and phonetic-order data-entry. This is also a major issue for entering ideographic characters in East Asia.
4. Resolution of these types of issues determine the complexity of the keystroke to character mapping process.



Here are some examples of the types of processing done to map scan codes into character codes.

1. In the first example, the user types the "E" key to produce the LATIN SMALL LETTER E, "e" character.
2. In the second example, the user holds a shift key down and presses the "E" key to produce the LATIN CAPITAL LETTER E, "E" character.
3. In the third example, a user on a French keyboard types the "ê" key to produce the LATIN SMALL LETTER E WITH CIRCUMFLEX, "ê" character. However, if the code uses combining characters, the input method will need to produce both the COMBINING CIRCUMFLEX ACCENT and the LATIN CAPITAL LETTER E, and then place them in the proper order depending on the character code.
4. In the last example, a user on a keyboard with "dead keys" for the accents types the "dead key" with the circumflex accent followed by the "E" key to produce the LATIN SMALL LETTER E WITH CIRCUMFLEX, "ê" character. However, like the third example, the target character code could using combining characters and require that the input method produce two character codes in the correct order.



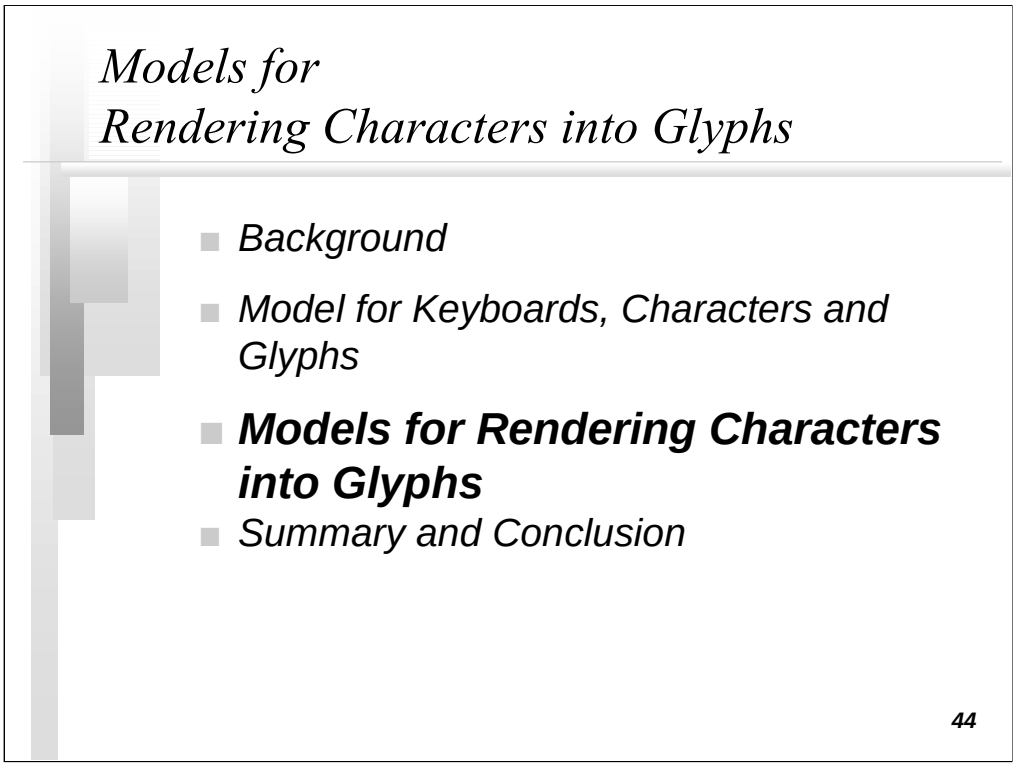
As a result of our discussion of keyboard entry, here are some additions to the guidelines for characters and glyphs presented earlier.

For deciding which characters to code in Unicode, the primary criteria is the information content. The coding decision should not depend solely on what is on the tops of the key of the keyboard.

In Unicode, the order of storing characters is the logical or phonetic order. The order in which characters are keyed on the keyboard does not necessarily determine the order in which they will be stored.

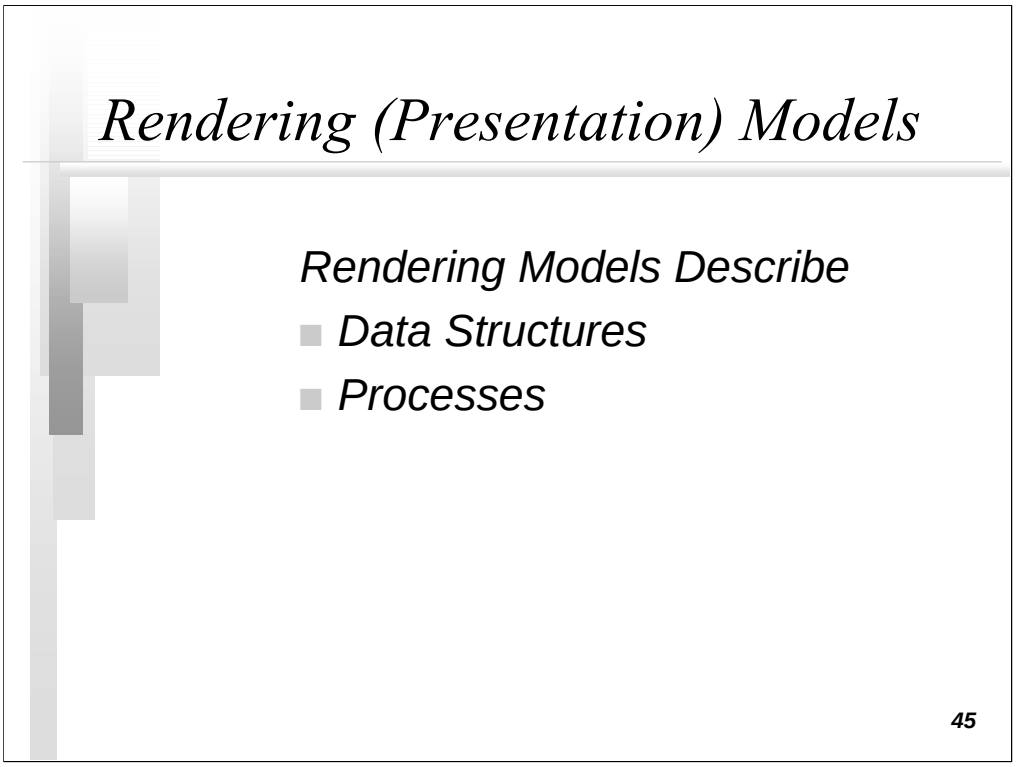
For those scripts where the phonetic order of the characters and their display order may be different, and where the culture allows for writing the characters in either phonetic order or display order, the input method must provide the user with the choice of the order of entering the text: display order or phonetic order.

Rendering characters into glyphs is a function of taking the logical (phonetic) ordering of characters in storage and displaying them in the proper order. The order of rendering characters is independent of the order in which the user typed them into the computer.

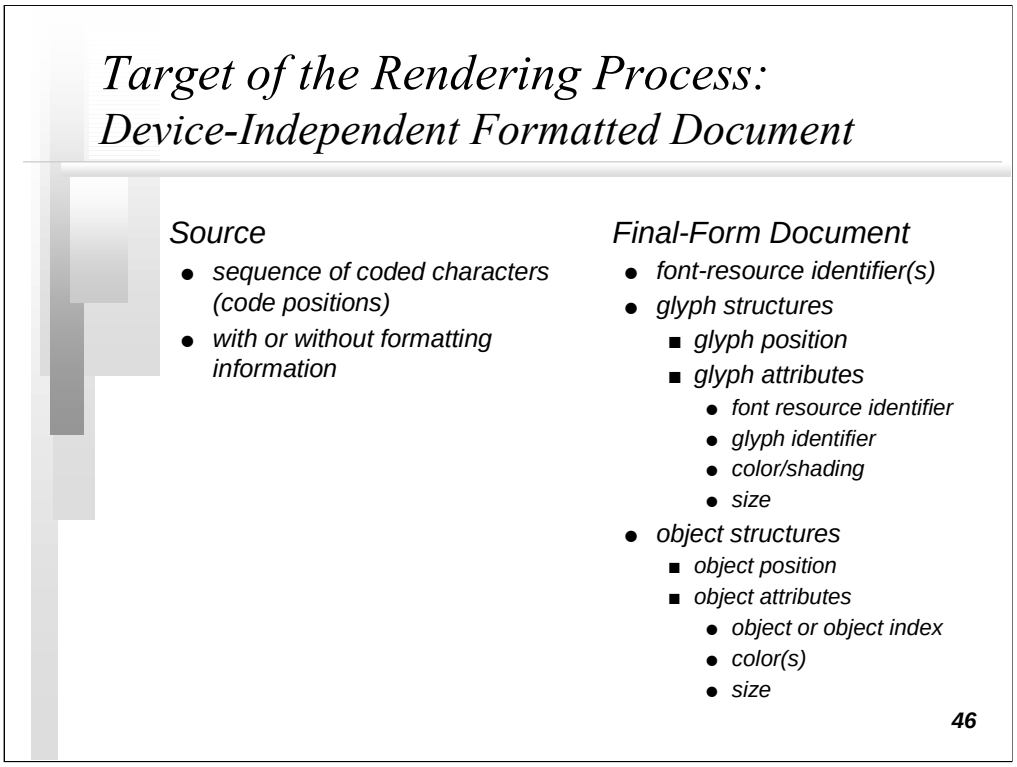


We are now at the next major section of this presentation.

In the previous section, we discussed the character domain and the glyph domain. We also discussed the glyph-selection process that maps characters into glyphs for displaying or printing. In this section, we will discuss three technologies for rendering characters into glyphs. At least two of these technologies are implemented now, and the third if it has not yet be delivered in products is not far from delivery.



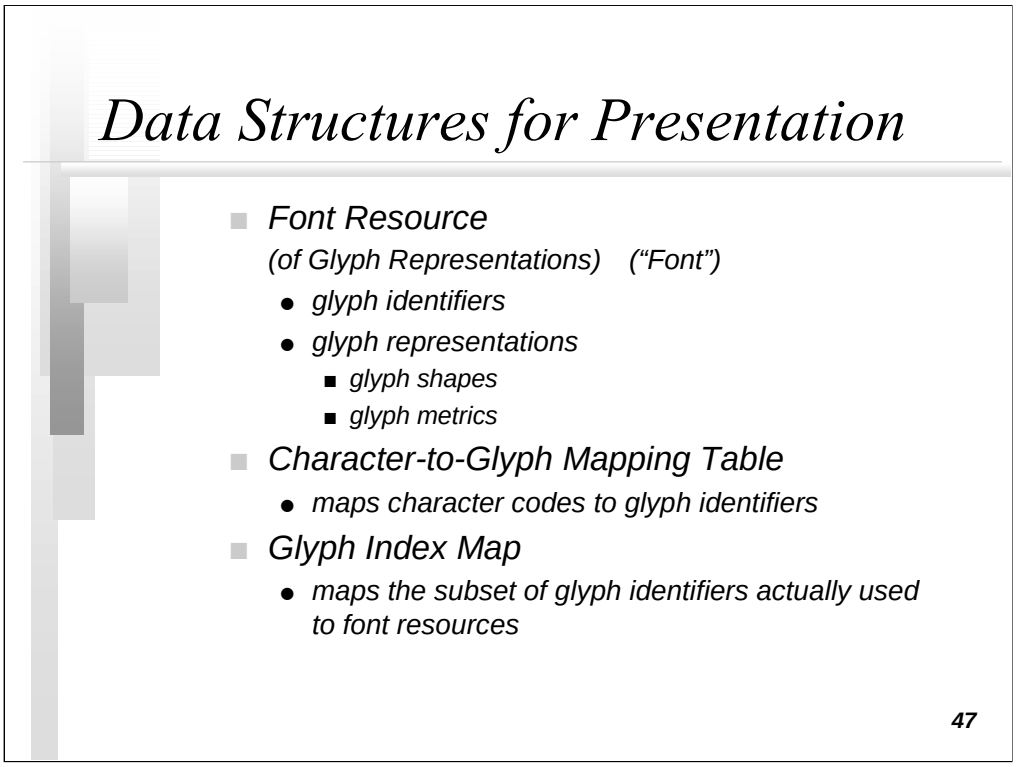
The three technologies rely on both data structures and processes to render characters.



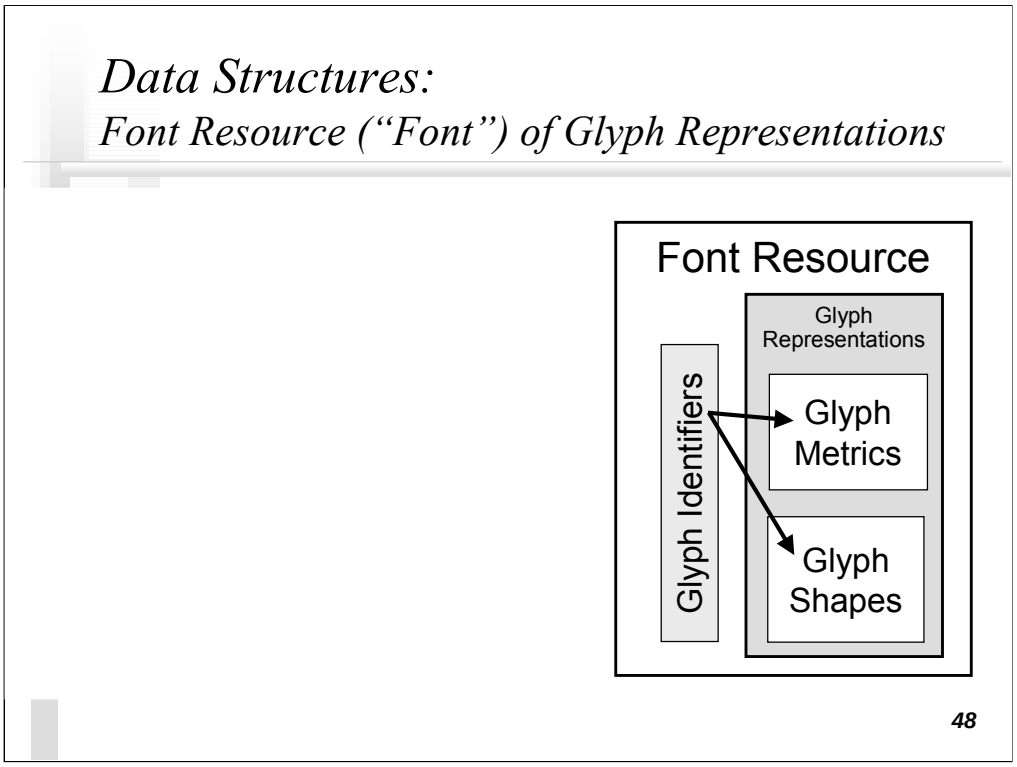
However, before we start discussing the data structures, let's discuss the inputs and outputs.

The input to the process is a sequence of coded characters (code positions). The input may or may not contain information for formatting the characters (bold, italic, underline, headers, paragraphs, lists, etc.).

The output of the process is what is called a "final-form document", which is ready for printing or displaying. In general, this output is independent of the particular output device, be it a display or a printer. The final form document consists of a list of fonts used in the document (font-resource identifiers), a list of glyph n-tupples (which typically contain the glyph position and glyph attributes such as the font resource, the glyph identifier, its color or shading, its size, etc.), and object structures (which contains the object's position and attributes).



We will be discussing three data structures: the "Font Resource" typically called a "Font", the Character-to-Glyph Mapping Table, and a Glyph Index Map. Note that these are not the only data structures used, they are merely the most common ones.



A Font Resource contains information to describe the set of glyphs in the font. It contains "Glyph Metrics" (which describe a glyph's height, width, placement relative to the baseline, and space from the previous character), and "Glyph Shapes" (which may be either a bitmap or the outline with some optional hinting). The Font Resource also includes a table to map each glyph identifier into its metric and shape information. This is the minimum information included in a Font Resource. However, the font designer may choose to include additional information.

Some Features of Font Resources

Characters	Glyph Shape	Glyph Metrics	Comments
LATIN CAPITAL LETTER A WITH RING, ANGSTROM SIGN	same	same	Arial Å Å Times New Roman Å Å
	different	same	Lucida Sans Unicode Å Å
LATIN CAPITAL LETTER A, GREEK CAPITAL LETTER ALPHA, CYRILLIC CAPITAL LETTER A	same	same (possible language differences)	Arial A Α А Times New Roman A Α А Lucida Sans Unicode A Α А
SPACE, NO-BREAK SPACE, EN-SPACE, EM-SPACE, ETC.	same (null)	different	
HYPHEN-MINUS, SOFT HYPHEN, HYPHEN, NON-BREAKING HYPHEN	same	different	Arial - - - - Times New Roman - - - -
	different	different	Lucida Sans Unicode ‒ - - ‒
PLUS SIGN, MINUS SIGN, MULTIPLICATION SIGN, DIVISION SIGN	different	same	Lucida Sans Unicode + − × ÷
CJK Ideographs	Includes glyphs for C_SC_TJK	horizontal and vertical layout	Font layering for multiple locales

49

This table illustrates some of the ways that the font resource designers implement fonts to reuse common data.

In the first row, the Arial and Times New Roman fonts use the same glyph shape for the LATIN CAPITAL LETTER A WITH RING, and the ANGSTROM SIGN. However, although it is not clear in this table, the Lucida San Unicode font includes different glyphs for each character.

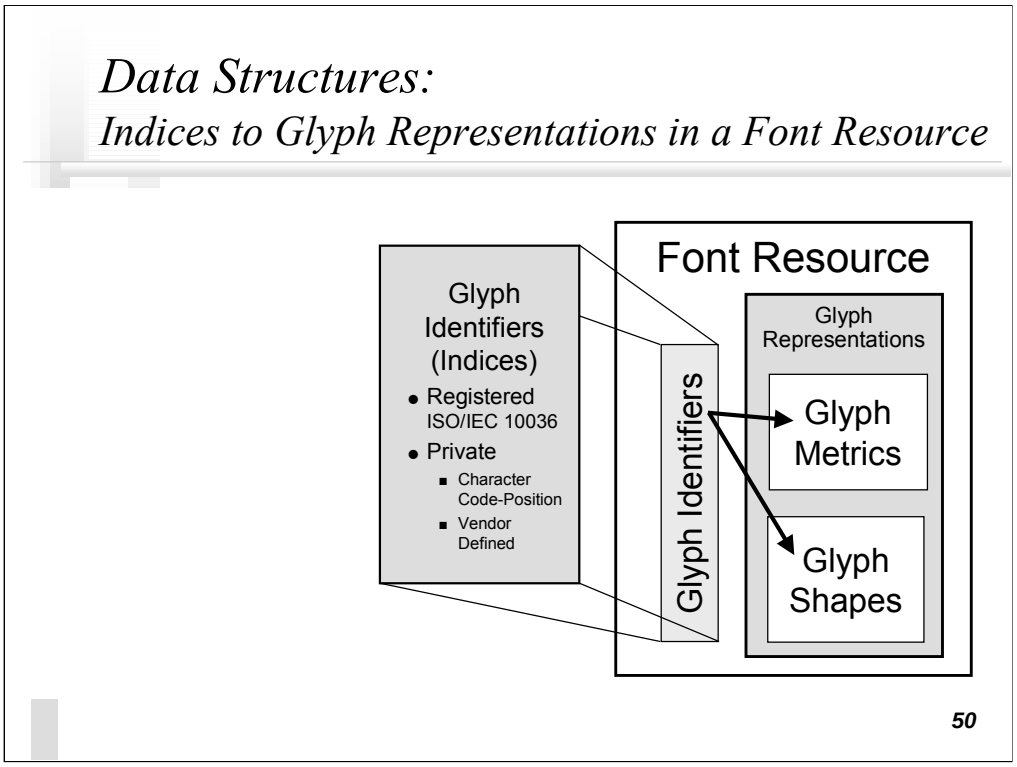
In the second row, the three fonts use the same shape and the same glyph metrics for the LATIN CAPITAL LETTER A, the GREEK CAPITAL LETTER ALPHA, and the CYRILLIC CAPITAL LETTER A. However, in the general case, the font may have different metrics that depend on the language or script.

For the third row, the various SPACE characters have the same (null) shape, but different metrics.

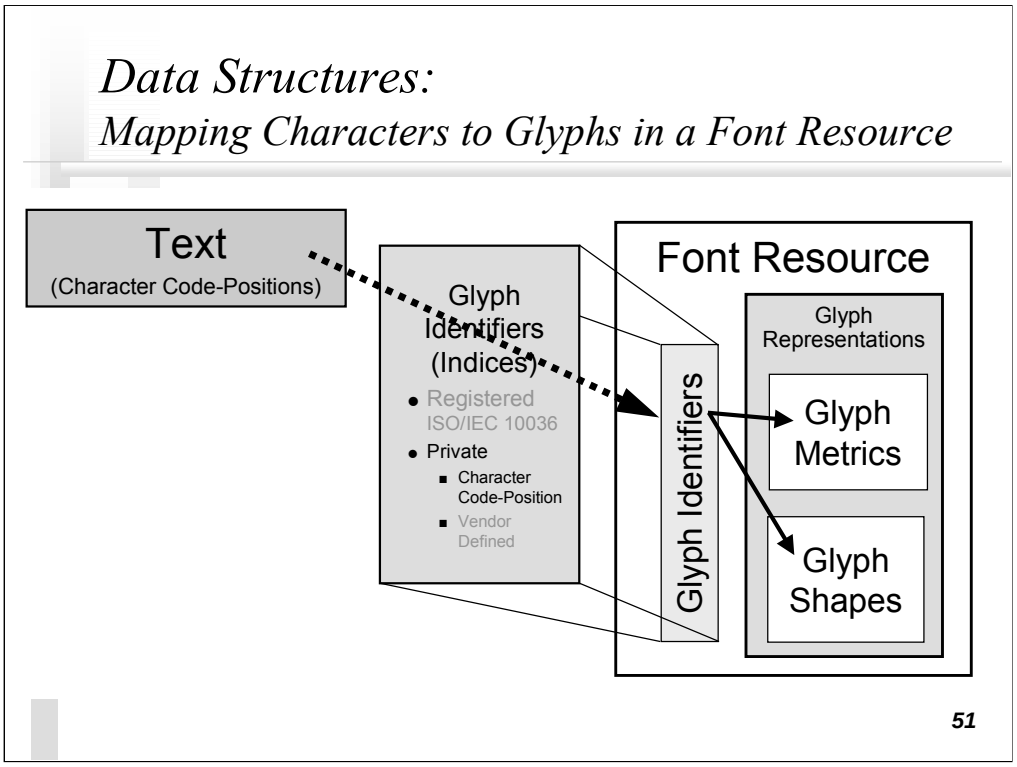
The fourth row illustrates different implementations of HYPHEN characters. Note that Lucida San Unicode uses a different shape.

The fifth row shows that spacing for the four basic arithmetic operators is the same.

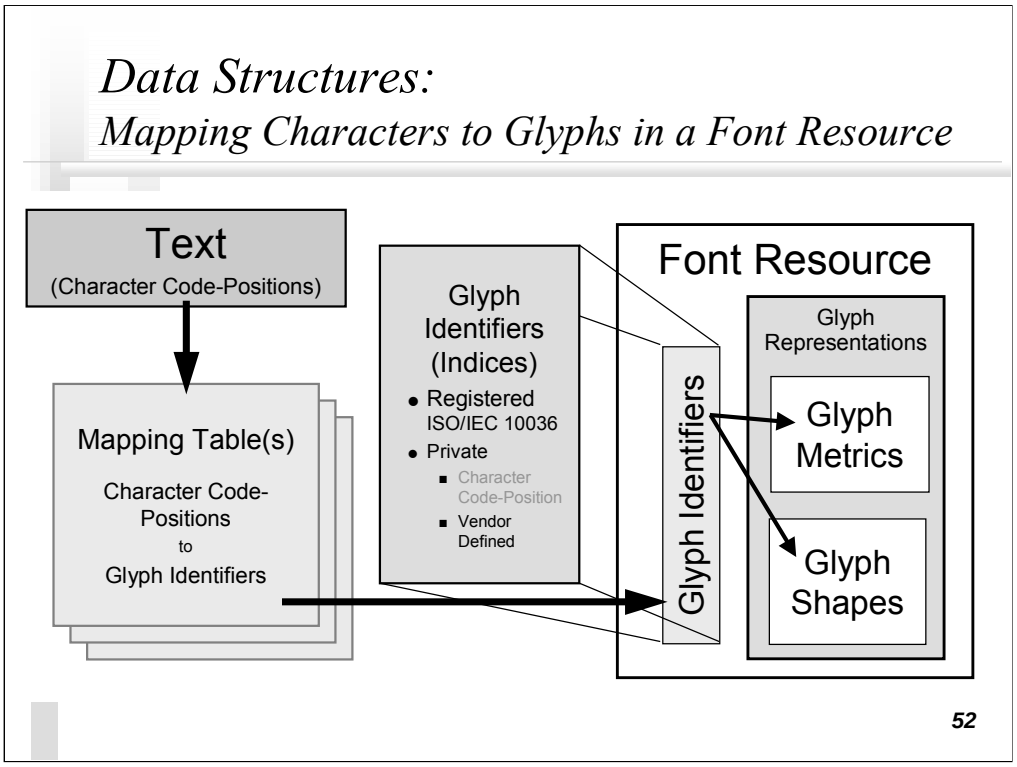
The sixth and last row is a relatively new development where a single font will contain traditional and simplified Chinese, Japanese and Korean glyphs for the unified ideographic characters plus metrics for both horizontal and vertical layout.



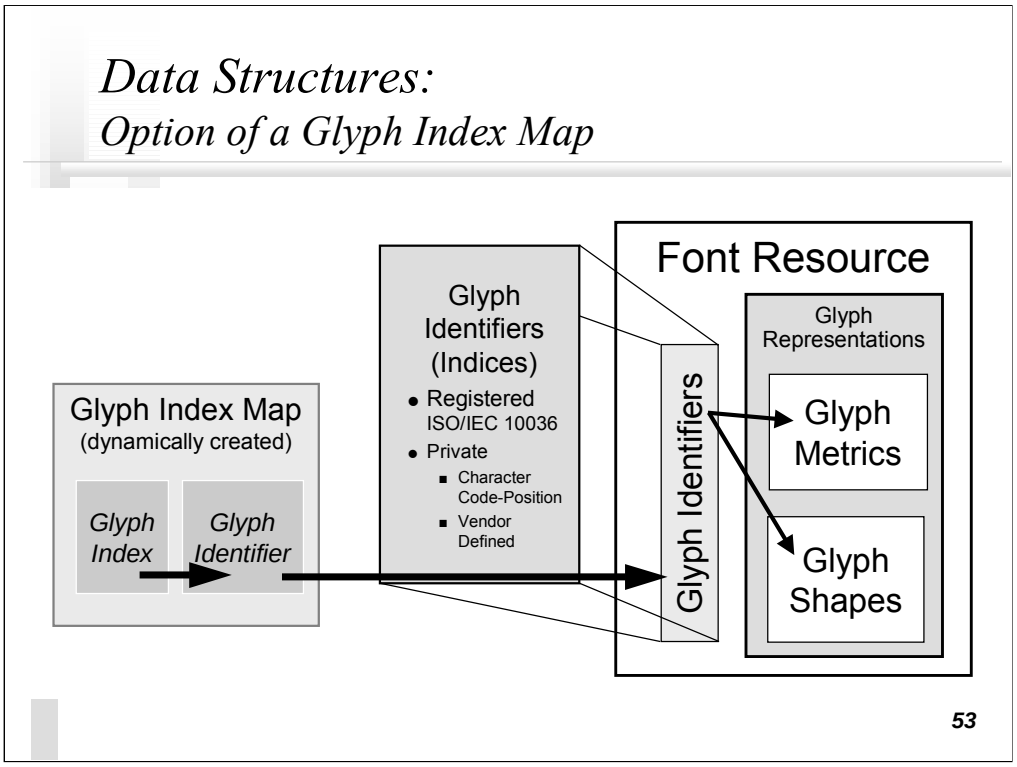
This figure illustrates different types of glyph identifiers used to index into the Font Resource. A font designer can use the registered glyph identifiers from the 10036 Registry or private glyph identifiers. Two types of private glyph identifiers are the character code position and vendor defined indices.



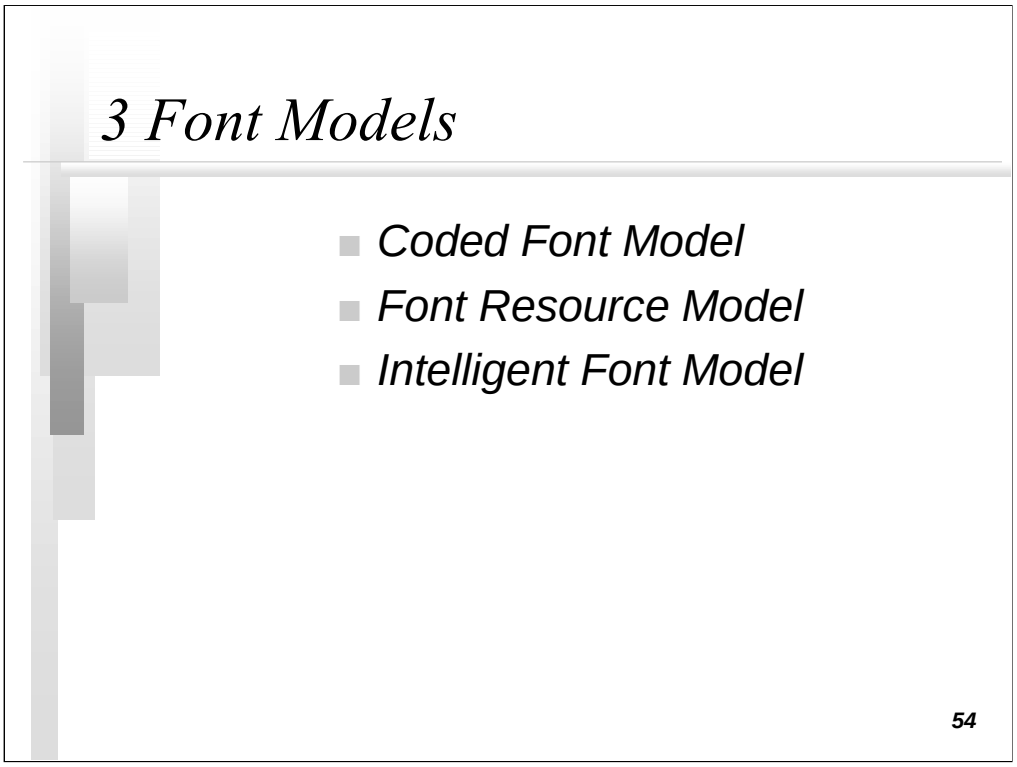
If the private, character code position is used for the glyph identifiers, glyph selection is one-to-one and consists of using the code position for a character to index into the font resource to obtain the glyph metrics and glyph shapes.



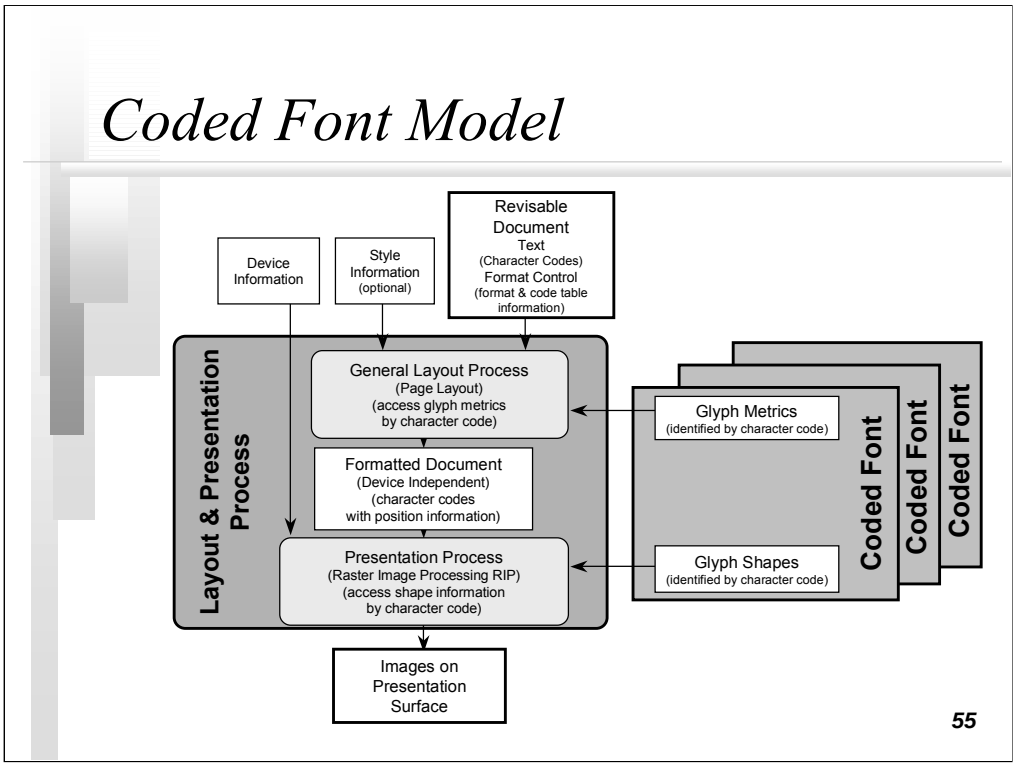
If glyph identifiers, either registered or vendor defined, are used, glyph selection requires tables to map from the character code position into the glyph identifier used to index into the font resource to obtain the glyph metrics and glyph shapes. The mapping tables may implement one-to-one mappings or more sophisticated *M*-to-*N* mappings.



An optional data structure is the "Glyph Index Map". This table is specific to a document and consists of entries for only those glyphs used in a particular document. The usefulness of this data structure depends on the assumption that rendering requires only a subset of the glyphs in a font rather than *all* of the glyphs in a font. The glyph index is used to obtain the glyph identifier (be it registered or private) to index into the Font Resource. The advantage of using this data structure is for large fonts, such as those that implement glyphs for large subsets of 10646 and Unicode. By using a Glyph Index Map, only the subset of glyph information for the glyphs actually used in the document need to be downloaded to render the document. This is valuable for printers with limited memory or for downloading pages from the WWW over slow telephone lines.



We will now turn to examine three font technologies to see how they use the various data structures to render characters into glyphs for display and printing.



A coded font (or a character-coded font) is a data structure in which character codes are used to index the glyph metric and glyph shape information contained in the font. The data structure depends on a one-to-one mapping of character codes to the glyphs in a font. Note that each code supported requires a separate coded font even if the glyphs are the same.

This font model is the historic presentation model for data processing. In this model, each character code encountered by the layout process is used to locate a corresponding glyph in the coded font. The glyph metric information for that character code is used to determine positioning of the glyph, along with line and page breaks. The formatted document may be interchanged to another location for presentation processing or transmitted to a local presentation process. The presentation process would use the character codes contained in the formatted document to locate a corresponding glyph in the coded font and use the associated glyph shape information to image the glyph on the presentation surface at the position indicated by the layout process.

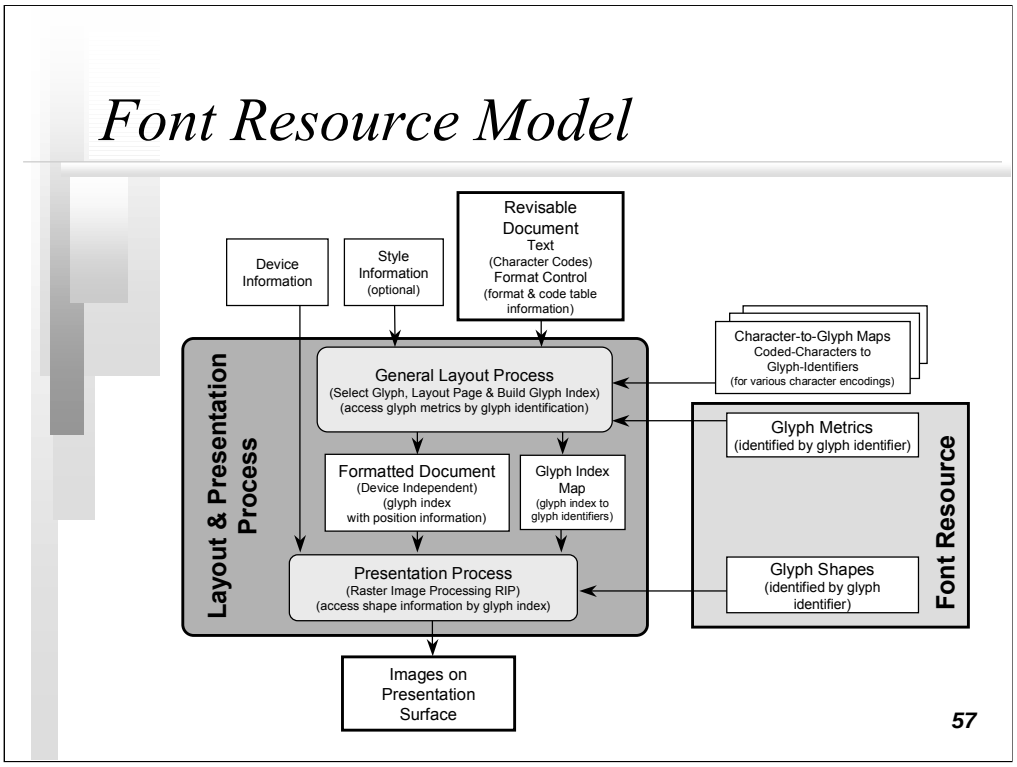
With the coded font model, if a desired glyph is not associated with a character in a coded character set, then the glyph cannot be displayed or printed. This fact and the widespread implementation of the coded font model have resulted in pressure to include some glyphs in coded character sets. The other two font models do not require that all the glyphs in a font resource be coded as characters in the coded character set to print or display the glyphs.

The coded font model is less suitable than the other two for the more complex glyph-selection requirements of printing and publishing. For example, the Arabic script requires special processing in the coded font model. If the input to the general layout process includes Arabic characters, the process also needs to convert the Arabic characters to the correct Arabic presentation forms, which is not normally part of this model.

Coded Font Model Characteristics

- *No Glyph-Selection Process*
  - *1-to-1 Mapping from Characters to Glyphs*
  - *Uses Character Code (Code-Position) to Index Font Resource*
- *Applicable to Most Characters in Unicode™ 4.0*
- *Widely Implemented*
- *Separate Font Resource for Each Code Table*
- *Inadequate for Multilingual Text and Advanced Typography for example, Processing the Arabic Script*
  - *Glyph Selection Process Required for Arabic Presentation Forms*
- *Each Glyph Must Be Coded as a Character*
  - *Glyphs Not in the Code Cannot Be Rendered*
  - *Pressure to Code Glyphs in 10646/Unicode™*
- *Optional Glyph-Index Map*

56



The font resource model permits definition of font resources that are less dependent on any single coded character set or document-processing model. This model is more suited to the document printing and publishing environment. Glyph identifiers index the glyph metrics and glyph shape representations in the font resource. In this model, the layout process uses predefined character-to-glyph maps to determine the mapping of character codes to presentation glyphs and replaces the character codes in the formatted document with glyph index values.

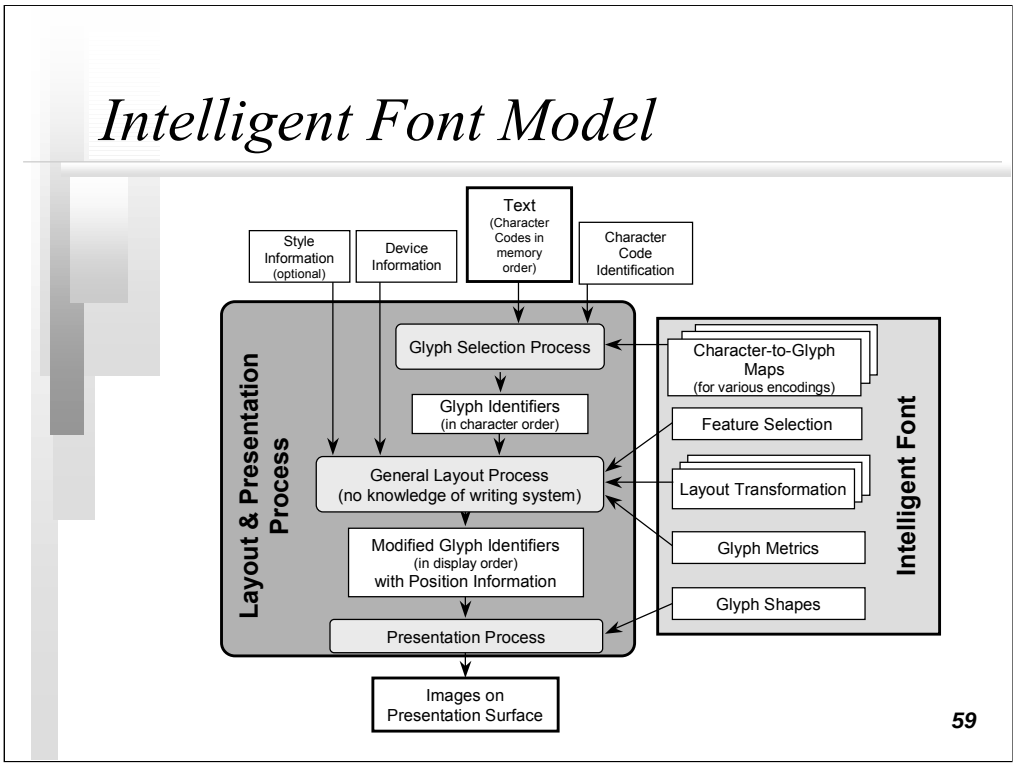
Optionally, a composition and layout process may generate a glyph index map that accesses only and exactly those glyphs of a large font resource that are needed to image the output of the process.

In the font resource model, the relationship between the character repertoire and the glyph collection may involve a one-to-one mapping but may also involve a one-to-many or many-to-one mapping. It is essential for successful presentation that the set of glyphs in the glyph collection be mappable to the repertoire of characters used in the text string.

Font Resource Model Characteristics

- *Adds Glyph-Selection Process*
  - *Could Be Sophisticated M-to-N or Merely 1-to-1 Mapping*
  - *Adequate for Multilingual Text and Advanced Typography*
- *Uses Glyph Identifiers (Different from Character Codes) to Index Font Resource*
  - *Less Dependent on Character Encoding*
  - *Private or Registered Glyph Identifiers*
  - *Font Resource May Cover Characters from Several Codes*
  - *Font Resource May Cover a Subset of 10646/Unicode*
- *Optional Glyph-Index Map*

58



An intelligent font is a data structure that augments a font resource with additional information describing

- how a sequence of coded characters is transformed into a sequence of glyph identifiers, with associated position information
- how the transformation of coded characters to glyph identifiers is affected by style information

The first set of additional information typically includes several mappings from various coded character sets to private (font-specific) glyph identifiers. Subsequent transformations use the glyph identifiers. The subsequent transformations may be complex and may result in changes to the number and ordering of the glyph identifiers. The second type of additional information permits, for example, substitution of glyph subsets (for example, swash variants, vertical substitution) based on style information.

Within the layout and presentation process of the intelligent font model, the glyph selection process transforms coded characters to glyph identifiers. This process requires

- information about how the characters are coded
- the map from coded characters to glyph identifiers for the specified character coding

The process takes coded characters in memory or logical order and produces glyph identifiers in character or logical order. Logical order is the order in which a person would normally read the characters regardless of the normal direction of the characters. For Latin text included in the middle of Arabic text, the logical order would be the rightmost Arabic character to the end of the Arabic text, then the leftmost Latin character to the end of the Latin text, and then the rightmost Arabic character of the second group of Arabic text to the end of the Arabic text.

Next, the general layout process transforms the glyph identifiers in logical order into (possibly modified) glyph identifiers in display order. Display order is the order in which the characters are to appear on paper or on a screen.

The presentation process is the final process. It takes the glyph identifiers in display order, the glyph positions, and the glyph shapes to produce the images on paper or a screen

Intelligent Font Model Characteristics

- *Font Resource with More Information*
  - *M-to-N Character-to-Glyph Maps with Position Information*
  - *Feature Selection Indicates Affects of Style Information*
  - *Layout Transformation Information Includes Provisions for Bi-Directional Text*
- *Potentially, More-Sophisticated M-to-N Glyph-Selection Process than Font Resource Model*
  - *Adequate for Multilingual Text and Advanced Typography*
- *Works with "Plain Text", Character Stream without Formatting Information*
- *Uses Private Glyph Identifiers to Index Font Resource*
- *Optional Glyph-Index Map*

60

Comparison of Font Models

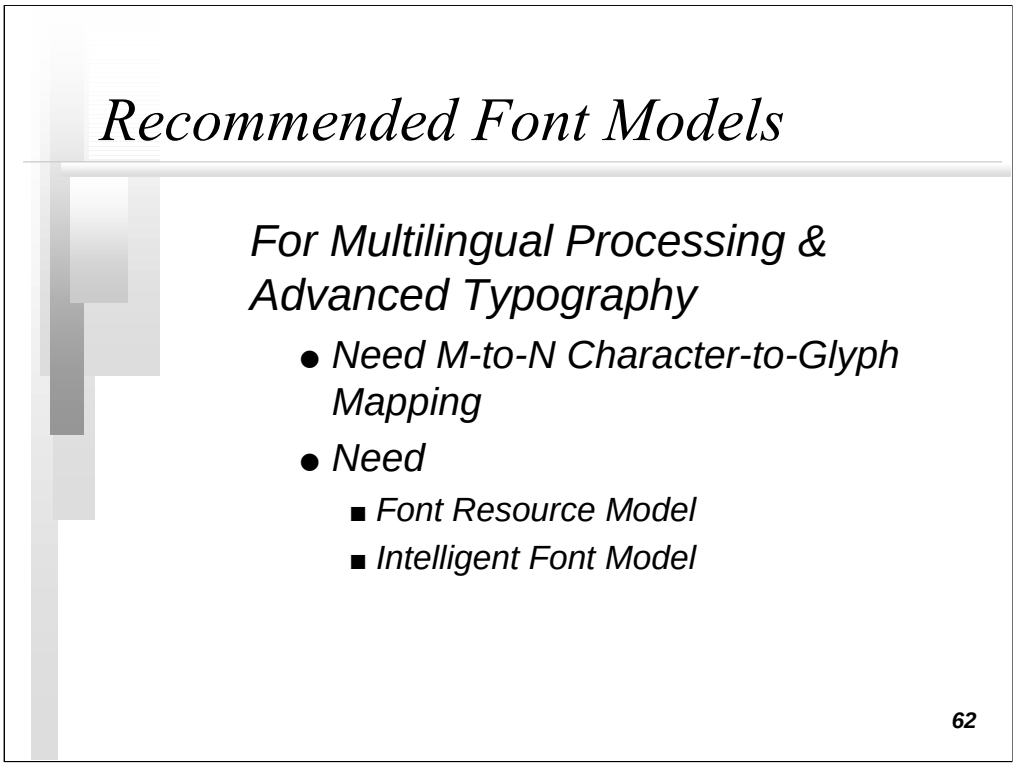
Characteristic	Coded Font	Font Resource	Intelligent Font
Glyph Selection Process (character-to-glyph mapping)	None (1-to-1)	Yes (1 Process) (1-to-1 or M-to-N)	Yes (2 Processes) (1-to-1 or M-to-N)
Font Structure			
Character-to-Glyph Mapping	No (implied by character code position)	Yes (external to font resource)	Yes (in font resource)
Index to Glyphs	Code Position in Character Code Table	Glyph Identifier (private or registered)	Glyph Identifier (private)
Glyph Metrics and Shapes	Yes	Yes	Yes
Additional Data	No	No	Feature Selection, Layout Transformation
Examples	Adobe PostScript Type1 Fonts Classic Printing in Data-Processing	Adobe CID Fonts Apple/Microsoft TrueType Fonts IBM Advanced Function Presentation (AFP)	Apple Advanced Typography Adobe/Microsoft OpenType

61

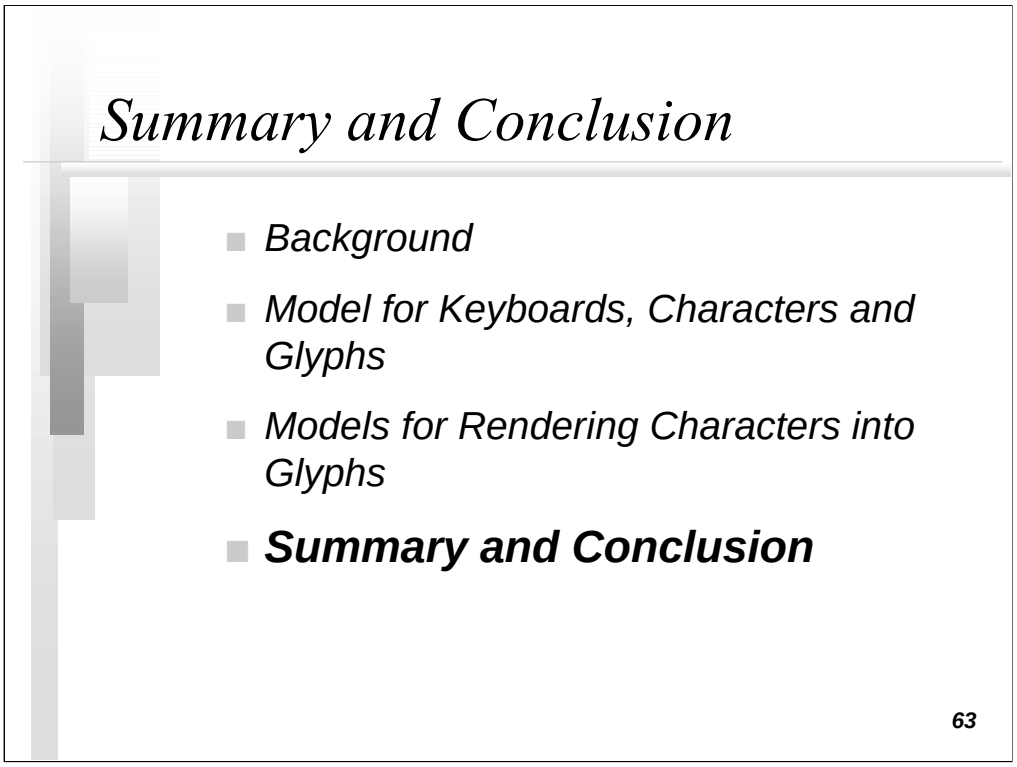
The principal differences between the font models are:

- the presence of a glyph selection process and the potential sophistication of that process
- how the glyphs are indexed in the font resource
- the presence of additional data in the font resource

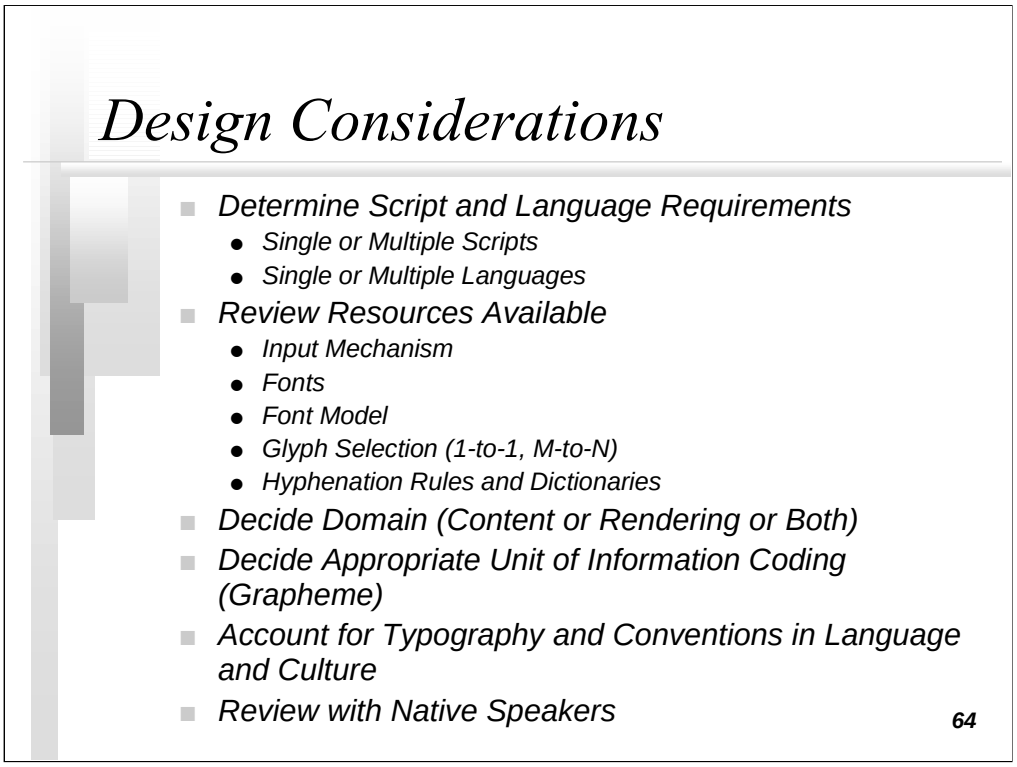
Examples of the Coded Font are Adobe PostScript Type 1 fonts and classic printing in data processing. Examples of the Font Resource are Adobe CID fonts, Apple/Microsoft TrueType fonts, and IBM Advanced Function Presentation (APF). Examples of Intelligent fonts are Apple Advanced Typography, and Adobe/Microsoft OpenType.



For multilingual and advanced typography, the more sophisticated *M*-to-*N* glyph selection is required. Potentially, both the Font Resource Model and the Intelligent Font have this capability. However, frequently this potential is unrealized with the Font Resource Model because only a one-to-one glyph selection is implemented.



At this point, let me try to draw some conclusions and summarize the presentation.



Let me first attempt to summarize how a developer might use the information presented by listing some design considerations.

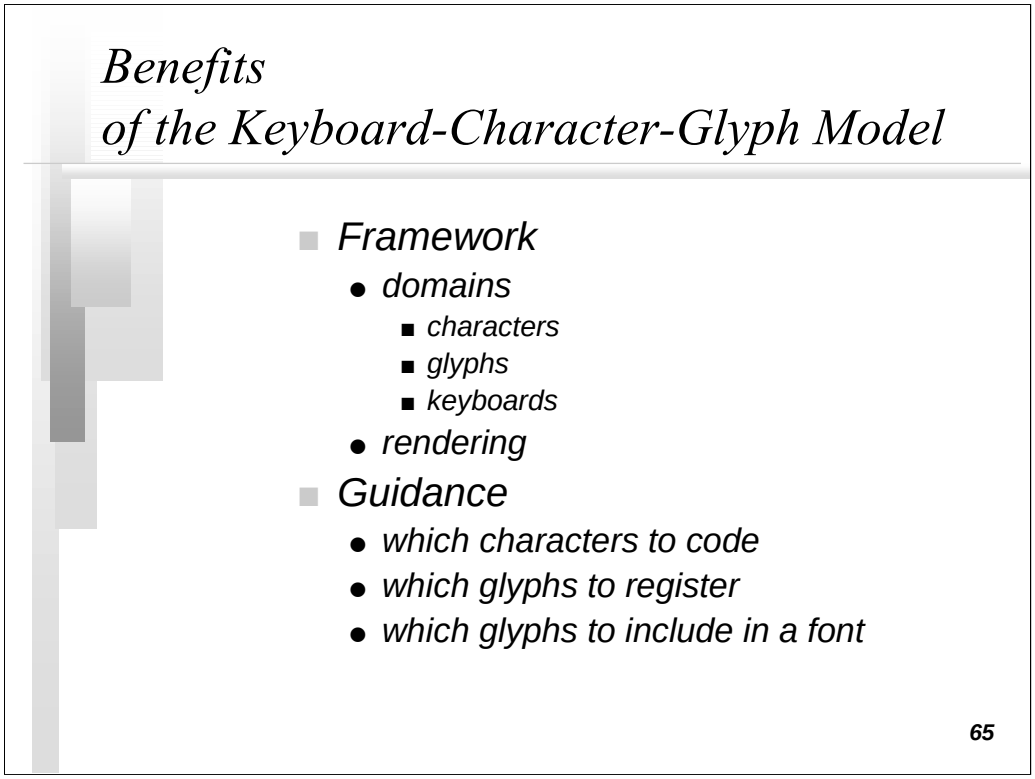
First, determine your script and language requirements. Are you working with one or multiple scripts, or one or multiple languages in your product?

Second, review the resources available on the platforms on which you are building your products. Does it have an input mechanism for the script/languages of your product? Are fonts available? Which font model is used for rendering? Does it provide one-to-one or *M*-to-*N* glyph selection? Are dictionaries with hyphenation rules available?

Third, decide the domain or domains required for your product. Is it in the content domain, or the presentation domain, or both? Based on your answer, consider if you should recode the information to optimize it to your processing.

Fourth, consider the typography and conventions used in the language and culture. This will have sometime subtle but import differences between cultures.

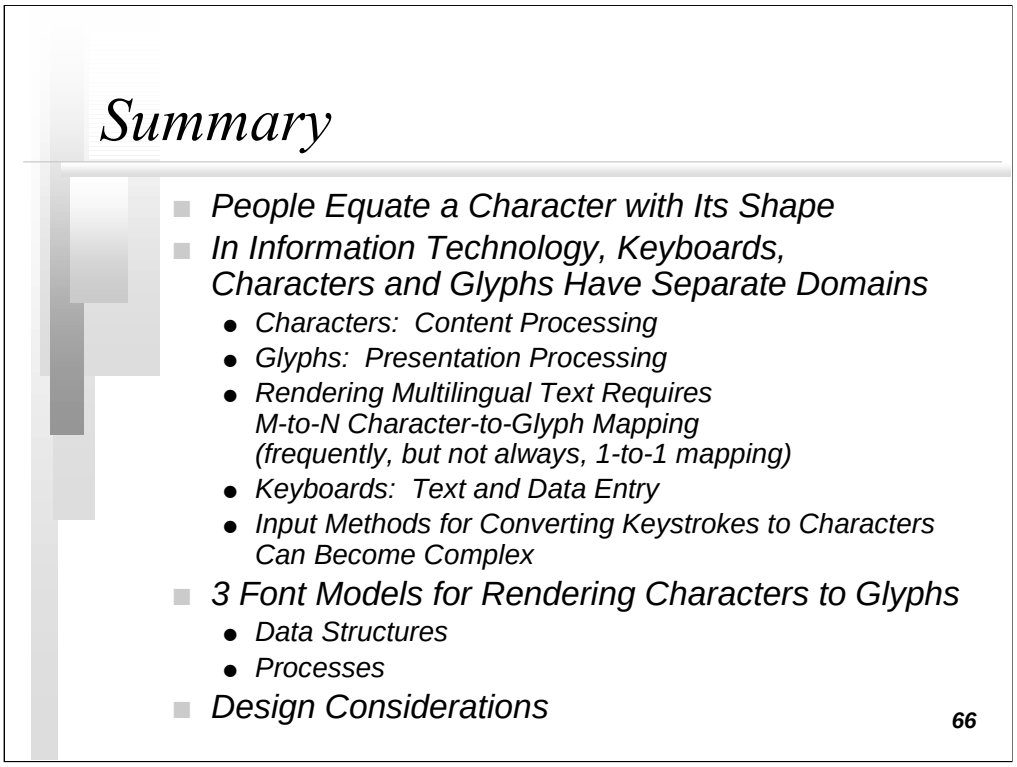
Finally, have native speakers review your product before you ship it. They will find problems that the "experts" miss and which may turn your product into a failure.



What are the benefits of the Character-Glyph model that was just described?

First, it provides a framework to developers and the standards committees to characterize the character domain and the glyph domain and to describe the glyph-selection process and models for rendering characters into glyphs for presentation. It also characterizes keyboards and input method processing to convert keystrokes into characters.

Second, it provides guidance to the standards committees about whether to code an entity as a character or to register it as a glyph. Note that I said "guidance" because frequently other considerations besides the idealized principles presented here become important in the decision process. Finally, it provides guidance for which additional presentation glyphs should be included in a font.



Let me conclude the presentation with a summary.

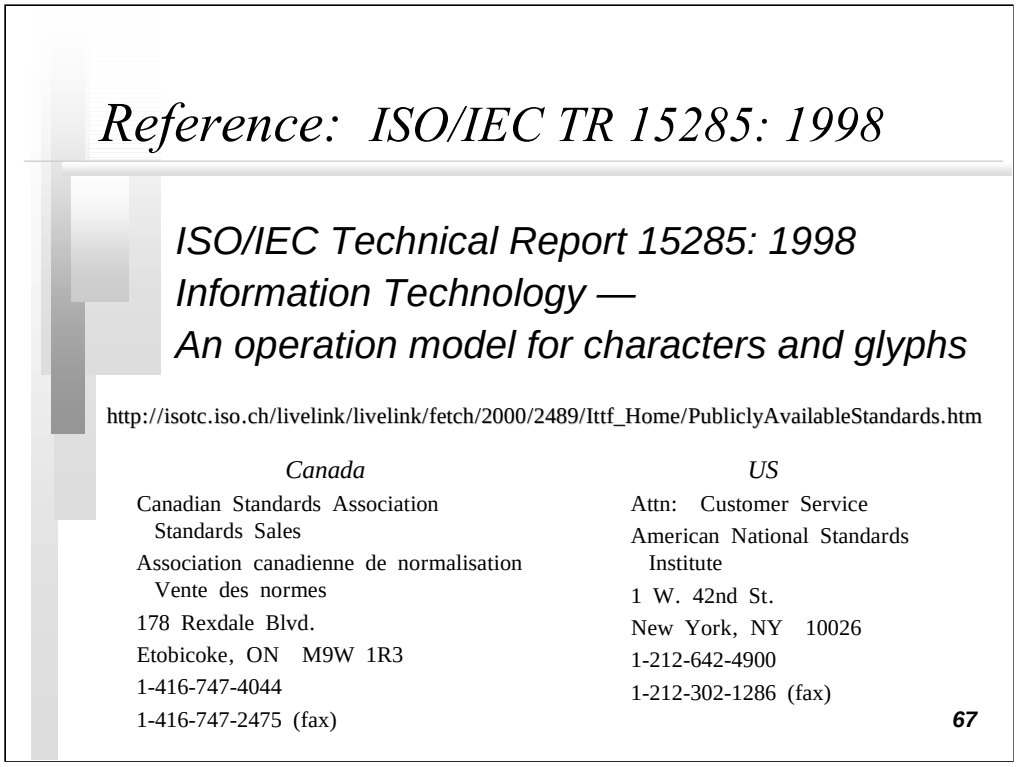
People equate a character with its shape. To a person, they are inseparable.

However, in the world of Information Technology, we have decided to distinguish between the character attribute of its information content, which we code as characters, and the attribute of its shape, which we call a glyph. We divide these into a character domain and a glyph domain. Converting from characters in the character domain to glyphs in the glyph domain requires a glyph selection process. The glyph selection process is frequently a one-to-one mapping but sometimes requires a more complex *M*-to-*N* mapping. In general, rendering multilingual text requires an *M*-to-*N* mapping.

Information Technology has a third domain for keyboards for text and data entry. The process for mapping keystrokes into characters, while simple for US English keyboards, can become quite complex.

Three technologies, or three font models, are available for rendering characters into glyphs. Each uses similar data structures and processes to render text. The presentation compared the font models and recommended two to use for multilingual rendering.

Finally, we listed several considerations for the developer of multilingual products.



You can obtain a copy of the ISO Technical Report on which this presentation is based from the ISO web site, www.iso.ch, or from your national standards organization. Since at times, ISO has redesigned its web site, you may need to go to www.iso.org (the new ISO URL in 2002) and search for "publicly available standards".