I

Unicode®

I.1 Introduction

The use of inconsistent **character encodings** (i.e., numeric values associated with characters) when developing global software products causes serious problems because computers process information using numbers. For example, the character "a" is converted to a numeric value so that a computer can manipulate that piece of data. Many countries and corporations have developed encoding systems that are incompatible with the encoding systems of other countries and corporations. For example, the Microsoft Windows operating system assigns the value `0xC0` to the character "A with a grave accent," while the Apple Macintosh operating system assigns the same value to an upside-down question mark. This results in the misrepresentation and possible corruption of data.

In the absence of a universal character encoding standard, global software developers had to localize their products extensively before distribution. **Localization** includes the language translation and cultural adaptation of content. The process of localization usually includes significant modifications to the source code (e.g., the conversion of numeric values and the underlying assumptions made by programmers), which results in increased costs and delays in releasing the software. For example, an English-speaking programmer might design a global software product assuming that a single character can be represented by one byte. However, when those products are localized in Asian markets, the programmer's assumptions are no longer valid because there are many more Asian characters, and therefore most, if not all, of the code needs to be rewritten. Localization is necessary with each release of a version. By the time a software product is localized for a particular market, a newer version, which needs to be localized as well, can be ready for distribution. As a result, it is cumbersome and costly to produce and distribute global software products in a market where there is no universal character encoding standard.

In response to this situation, the **Unicode Standard**, an encoding standard that facilitates the production and distribution of software, was created. The Unicode Standard outlines a specification to produce consistent encoding of the world's characters and symbols. Software products which handle text encoded in the Unicode Standard need to be

localized, but the localization process is simpler and more efficient because the numeric values need not be converted and the assumptions made by programmers about the character encoding are universal. The Unicode Standard is maintained by a non-profit organization called the **Unicode Consortium**, whose members include Apple, IBM, Microsoft, Oracle, Sun Microsystems, Sybase and many others.

When the Consortium envisioned and developed the Unicode Standard, it wanted an encoding system that was **universal**, **efficient**, **uniform** and **unambiguous**. A universal encoding system encompasses all commonly used characters. An efficient encoding system allows text files to be parsed quickly. A uniform encoding system assigns fixed values to all characters. An unambiguous encoding system represents a given character in a consistent manner. These four terms are referred to as the Unicode Standard design basis.

I.2 Unicode Transformation Formats

Although Unicode incorporates the limited ASCII character set (i.e., a collection of characters), it encompasses a more comprehensive character set. In ASCII each character is represented by a byte containing 0s and 1s. One byte is capable of storing the binary numbers from 0 to 255. Each character is assigned a number between 0 and 255, thus ASCII-based systems can support only 256 characters, a tiny fraction of the world's characters. Unicode extends the ASCII character set by encoding the vast majority of the world's characters. The Unicode Standard encodes characters in a uniform numerical space from 0 to 10FFFF hexadecimal. An implementation will express these numbers in one of several transformation formats, choosing the one that best fits the particular application at hand.

Three such formats are in use, called **UTF-8**, **UTF-16** and **UTF-32**. UTF-8, a variable-width encoding form, requires one to four bytes to express each Unicode character. UTF-8 data consists of 8-bit bytes (sequences of one, two, three or four bytes depending on the character being encoded) and is well suited for ASCII-based systems when there is a predominance of one-byte characters (ASCII represents characters as one-byte). Currently, UTF-8 is widely implemented in UNIX systems and in databases.

The variable-width UTF-16 encoding form expresses Unicode characters in units of 16-bits (i.e., as two adjacent bytes, or a short integer in many machines). Most characters of Unicode are expressed in a single 16-bit unit. However, characters with values above FFFF hexadecimal are expressed with an ordered pair of 16-bit units called **surrogates**. Surrogates are 16-bit integers in the range D800 through DFFF, which are used solely for the purpose of "escaping" into higher numbered characters. Approximately one million characters can be expressed in this manner. Although a surrogate pair requires 32 bits to represent characters, it is space-efficient to use these 16-bit units. Surrogates are rare characters in current implementations. Many string-handling implementations are written in terms of UTF-16. [*Note:* Details and sample-code for UTF-16 handling are available on the Unicode Consortium Web site at `www.unicode.org`.]

Implementations that require significant use of rare characters or entire scripts encoded above FFFF hexadecimal, should use UTF-32, a 32-bit fixed-width encoding form that usually requires twice as much memory as UTF-16 encoded characters. The major advantage of the fixed-width UTF-32 encoding form is that it expresses all characters uniformly, so it is easy to handle in arrays.

There are few guidelines that state when to use a particular encoding form. The best encoding form to use depends on the computer system and business protocol, not on the

data itself. Typically, the UTF-8 encoding form should be used where computer systems and business protocols require data to be handled in 8-bit units, particularly in legacy systems being upgraded, because it often simplifies changes to existing programs. For this reason, UTF-8 has become the encoding form of choice on the Internet. Likewise, UTF-16 is the encoding form of choice on Microsoft Windows applications. UTF-32 is likely to become more widely used in the future as more characters are encoded with values above FFFF hexadecimal. UTF-32 requires less sophisticated handling than UTF-16 in the presence of surrogate pairs. Figure I.1 shows the different ways in which the three encoding forms handle character encoding.

Character	UTF-8	UTF-16	UTF-32
LATIN CAPITAL LETTER A	0x41	0x0041	0x00000041
GREEK CAPITAL LETTER ALPHA	0xCD 0x91	0x0391	0x00000391
CJK UNIFIED IDEOGRAPH-4E95	0xE4 0xBA 0x95	0x4E95	0x00004E95
OLD ITALIC LETTER A	0xF0 0x80 0x83 0x80	0xDC00 0xDF00	0x00010300

Fig. I.1 | Correlation between the three encoding forms.

I.3 Characters and Glyphs

The Unicode Standard consists of characters—written components (i.e., alphabets, numbers, punctuation marks, accent marks, etc.) that can be represented by numeric values. An example of such a character is U+0041 LATIN CAPITAL LETTER A. In the first character representation, **U+*yyyy*** is a **code value**, in which U+ refers to Unicode code values, as opposed to other hexadecimal values. The *yyyy* represents a four-digit hexadecimal number of an encoded character. Code values are bit combinations that represent encoded characters. Characters are represented using **glyphs**—various shapes, fonts and sizes for displaying characters. There are no code values for glyphs in the Unicode Standard. Examples of glyphs are shown in Fig. I.2.

The Unicode Standard encompasses the alphabets, ideographs, syllabaries, punctuation marks, **diacritics**, mathematical operators and other features. that comprise the written languages and scripts of the world. A diacritic is a special mark added to a character to distinguish it from another letter or to indicate an accent (e.g., in Spanish, the tilde "~" above the character "n"). Currently, Unicode provides code values for 96,382 character representations, with more than 878,000 code values reserved for future expansion.

Fig. I.2 | Various glyphs of the character A.

I.4 Advantages/Disadvantages of Unicode

The Unicode Standard has several significant advantages that promote its use. One is the impact it has on the performance of the international economy. Unicode standardizes the characters for the world's writing systems to a uniform model that promotes transferring and sharing data. Programs developed using such a schema maintain their accuracy because each character has a single definition (i.e., *a* is always U+0061, *%* is always U+0025). This enables corporations to manage the high demands of international markets by processing different writing systems at the same time. All characters can be managed in an identical manner, thus avoiding any confusion caused by different character-code architectures. Moreover, managing data in a consistent manner eliminates data corruption, because data can be sorted, searched and manipulated using a consistent process.

Another advantage of the Unicode Standard is portability (i.e., software that can execute on disparate computers or with disparate operating systems). Most operating systems, databases, programming languages (including Java and Microsoft's .NET languages) and Web browsers currently support, or are planning to support, Unicode.

A disadvantage of the Unicode Standard is the amount of memory required by UTF-16 and UTF-32. ASCII character sets are 8-bits in length, so they require less storage than the default 16-bit Unicode character set. The **double-byte character set** (DBCS) encodes Asian characters with one or two bytes per character. The **multibyte character set** (MBCS) encodes characters with a variable number of bytes per character. In such instances, the UTF-16 or UTF-32 encoding forms may be used with little hindrance on memory and performance.

Another disadvantage of Unicode is that although it includes more characters than any other character set in common use, it does not yet encode all of the world's written characters. Also, UTF-8 and UTF-16 are variable-width encoding forms, so characters occupy different amounts of memory.

I.5 Unicode Consortium's Web Site

If you would like to learn more about the Unicode Standard, visit `www.unicode.org`. This site provides a wealth of information about the Unicode Standard. Currently, the home page is organized into various sections: New to Unicode, General Information, The Consortium, The Unicode Standard, Work in Progress and For Members, Key Specifications and Technical Publications.

The New to Unicode section consists of four subsections: **What is Unicode?**, **How to Use this Site**, **FAQ** and **Glossary of Unicode Terms**. The first subsection provides a technical introduction to Unicode by describing design principles, character interpretations and assignments, text processing and Unicode conformance. This subsection is recommended reading for anyone new to Unicode. Also, this subsection provides a list of related links that provide the reader with additional information about Unicode. The **How to Use this Site** subsection contains information about using and navigating the site as well hyperlinks to additional resources.

The General Information section contains five subsections: **Where is my Character?**, **Display Problems?**, **Useful Resources**, **Unicode Enabled Products** and **Mail Lists**. The main areas covered in this section include a link to the Unicode code charts (a complete listing of code values) assembled by the Unicode Consortium as well as a detailed outline on how to locate an encoded character in the code chart. Also, the section contains advice on how

to configure different operating systems and Web browsers so that the Unicode characters can be viewed properly. Moreover, from this section, the user can navigate to other sites that provide information on various topics, such as fonts, linguistics and such other standards as the Armenian Standards Page and the Chinese GB 18030 Encoding Standard.

The Consortium section consists of eight subsections: Who We Are, Our Members, How to Join, Conferences, Job Postings, Press Info, Policies & Positions and Contact Us. This section provides a list of the current Unicode Consortium members as well as information on how to become a member. Privileges for each member type—full, associate, specialist and individual—and the fees assessed to each member are listed here.

The For Members section consists of two subsections that are available only to consortium members: Member Resources and Working Documents.

The Unicode Standard section consists of five subsections: Start Here, Latest Version, Code Charts, Unicode Character Database and Unihan Database. This section describes the updates applied to the latest version of the Unicode Standard and categorizes all defined encoding. The user can learn how the latest version has been modified to encompass more features and capabilities. For instance, one enhancement of Version 3.1 is that it contains additional encoded characters.

The Key Specifications and Technical Publications sections provide all the Unicode technical documentation.

The Work in Progress section consists of seven subsections: Calendar of Meetings, Proposals for Public Review, Unicode Technical Committee, UTC Meeting Minutes, Proposed Characters, Submitting Proposals and CLDR Technical Committee. This section presents the user with a catalog of the recent characters included into the Unicode Standard scheme as well as those characters being considered for inclusion. If users determine that a character has been overlooked, then they can submit a written proposal for the inclusion of that character. The Submitting Proposals subsection contains strict guidelines that must be adhered to when submitting written proposals. In addition, this section provides information about upcoming and past technical committee meetings.

I.6 Using Unicode

Numerous programming languages (e.g., C, Java, JavaScript, Perl, Visual Basic) provide some level of support for the Unicode Standard. The application shown in Fig. I.3–Fig. I.4 prints the text "Welcome to Unicode!" in eight different languages: English, Russian, French, German, Japanese, Portuguese, Spanish and Traditional Chinese.

Class `UnicodeJFrame` (Fig. I.3) uses escape sequences to represent characters. An escape sequence is in the form `\u`*yyyy*, where *yyyy* represents the four-digit hexadecimal

```
1  // Fig. F.3: UnicodeJFrame.java
2  // Demonstrating how to use Unicode in Java programs.
3  import java.awt.GridLayout;
4  import javax.swing.JFrame;
5  import javax.swing.JLabel;
6
7  public class UnicodeJFrame extends JFrame
8  {
```

Fig. I.3 | Java application that uses Unicode encoding (Part 1 of 2.).

```
9     // constructor creates JLabels to display Unicode
10    public UnicodeJFrame()
11    {
12       super( "Demonstrating Unicode" );
13
14       setLayout( new GridLayout( 8, 1 ) ); // set frame layout
15
16       // create JLabels using Unicode
17       JLabel englishJLabel = new JLabel( "\u0057\u0065\u006C\u0063" +
18          "\u006F\u006D\u0065\u0020\u0074\u006F\u0020Unicode\u0021" );
19       englishJLabel.setToolTipText( "This is English" );
20       add( englishJLabel );
21
22       JLabel chineseJLabel = new JLabel( "\u6B22\u8FCE\u4F7F\u7528" +
23          "\u0020\u0020Unicode\u0021" );
24       chineseJLabel.setToolTipText( "This is Traditional Chinese" );
25       add( chineseJLabel );
26
27       JLabel cyrillicJLabel = new JLabel( "\u0414\u043E\u0431\u0440" +
28          "\u043E\u0020\u043F\u043E\u0436\u0430\u043B\u043E\u0432" +
29          "\u0430\u0442\u044A\u0020\u0432\u0020Unicode\u0021" );
30       cyrillicJLabel.setToolTipText( "This is Russian" );
31       add( cyrillicJLabel );
32
33       JLabel frenchJLabel = new JLabel( "\u0042\u0069\u0065\u006E\u0076" +
34          "\u0065\u006E\u0075\u0065\u0020\u0061\u0075\u0020Unicode\u0021");
35       frenchJLabel.setToolTipText( "This is French" );
36       add( frenchJLabel );
37
38       JLabel germanJLabel = new JLabel( "\u0057\u0069\u006C\u006B\u006F" +
39          "\u006D\u006D\u0065\u006E\u0020\u007A\u0075\u0020Unicode\u0021");
40       germanJLabel.setToolTipText( "This is German" );
41       add( germanJLabel );
42
43       JLabel japaneseJLabel = new JLabel( "Unicode\u3078\u3087\u3045" +
44          "\u3053\u305D\u0021" );
45       japaneseJLabel.setToolTipText( "This is Japanese" );
46       add( japaneseJLabel );
47
48       JLabel portugueseJLabel = new JLabel( "\u0053\u00E9\u006A\u0061" +
49          "\u0020\u0042\u0065\u006D\u0076\u0069\u006E\u0064\u006F\u0020" +
50          "Unicode\u0021" );
51       portugueseJLabel.setToolTipText( "This is Portuguese" );
52       add( portugueseJLabel );
53
54       JLabel spanishJLabel = new JLabel( "\u0042\u0069\u0065\u006E" +
55          "\u0076\u0065\u006E\u0069\u0064\u0061\u0020\u0061\u0020" +
56          "Unicode\u0021" );
57       spanishJLabel.setToolTipText( "This is Spanish" );
58       add( spanishJLabel );
59    } // end UnicodeJFrame constructor
60 } // end class UnicodeJFrame
```

Fig. I.3 | Java application that uses Unicode encoding (Part 2 of 2.).

```
1  // Fig. F.4: Unicode.java
2  // Display Unicode.
3  import javax.swing.JFrame;
4
5  public class Unicode
6  {
7     public static void main( String args[] )
8     {
9        UnicodeJFrame unicodeJFrame = new UnicodeJFrame();
10       unicodeJFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       unicodeJFrame.setSize( 350, 250 );
12       unicodeJFrame.setVisible( true );
13    } // end method main
14 } // end class Unicode
```

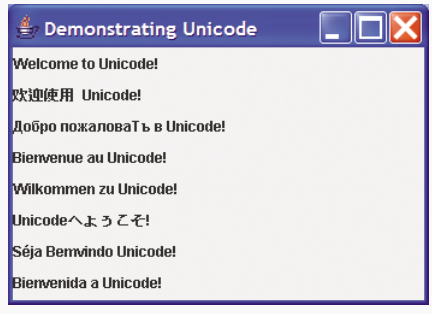

Fig. I.4 | Displaying Unicode.

code value. Lines 17–18 contain the series of escape sequences necessary to display "Welcome to Unicode!" in English. The first escape sequence (`\u0057`) equates to the character "W," the second escape sequence (`\u0065`) equates to the character "e," and so on. The `\u0020` escape sequence (line 18) is the encoding for the space character. The `\u0074` and `\u006F` escape sequences equate to the word "to." Note that "Unicode" is not encoded because it is a registered trademark and has no equivalent translation in most languages. Line 18 also contains the `\u0021` escape sequence for the exclamation point (`!`).

Lines 22–56 contain the escape sequences for the other seven languages. The Unicode Consortium's Web site contains a link to code charts that lists the 16-bit Unicode code values. The English, French, German, Portuguese and Spanish characters are located in the **Basic Latin** block, the Japanese characters are located in the **Hiragana** block, the Russian characters are located in the **Cyrillic** block and the Traditional Chinese characters are located in the **CJK Unified Ideographs** block. The next section discusses these blocks.

I.7 Character Ranges

The Unicode Standard assigns code values, which range from 0000 (**Basic Latin**) to E007F (**Tags**), to the written characters of the world. Currently, there are code values for 96,382 characters. To simplify the search for a character and its associated code value, the Unicode Standard generally groups code values by script and function (i.e., Latin characters are grouped in a block, mathematical operators are grouped in another block, etc.). As a rule, a script is a single writing system that is used for multiple languages (e.g., the Latin script

is used for English, French, Spanish, etc.). The **Code Charts** page on the Unicode Consortium Web site lists all the defined blocks and their respective code values. Figure I.5 lists some blocks (scripts) from the Web site and their range of code values.

Script	Range of code values
Arabic	U+0600–U+06FF
Basic Latin	U+0000–U+007F
Bengali (India)	U+0980–U+09FF
Cherokee (Native America)	U+13A0–U+13FF
CJK Unified Ideographs (East Asia)	U+4E00–U+9FFF
Cyrillic (Russia and Eastern Europe)	U+0400–U+04FF
Ethiopic	U+1200–U+137F
Greek	U+0370–U+03FF
Hangul Jamo (Korea)	U+1100–U+11FF
Hebrew	U+0590–U+05FF
Hiragana (Japan)	U+3040–U+309F
Khmer (Cambodia)	U+1780–U+17FF
Lao (Laos)	U+0E80–U+0EFF
Mongolian	U+1800–U+18AF
Myanmar	U+1000–U+109F
Ogham (Ireland)	U+1680–U+169F
Runic (Germany and Scandinavia)	U+16A0–U+16FF
Sinhala (Sri Lanka)	U+0D80–U+0DFF
Telugu (India)	U+0C00–U+0C7F
Thai	U+0E00–U+0E7F

Fig. I.5 | Some character ranges.

Self-Review Exercises

I.1 Fill in the blanks in each of the following.

a) Global software developers had to _____ their products to a specific market before distribution.

b) The Unicode Standard is a(n) _____ standard that facilitates the uniform production and distribution of software products.

c) The four design basis that comprises the Unicode Standard are: _____, _____, _____ and _____.

d) Characters are represented using _____.

e) Software that can execute on different operating systems is said to be _____.

I.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) The Unicode Standard encompasses all the world's characters.
b) A Unicode code value is represented as U+*yyyy*, where *yyyy* represents a number in binary notation.
c) A diacritic is a character with a special mark that emphasizes an accent.
d) Unicode is portable.
e) When designing Java programs, a Unicode escape sequence is denoted by /u*yyyy*.

Answers to Self-Review Exercises

I.1 a) localize. b) encoding. c) universal, efficient, uniform, unambiguous. d) glyphs. e) portable.

I.2 a) False. It encompasses the majority of the world's characters. b) False. The *yyyy* represents a hexadecimal number. c) False. A diacritic is a special mark added to a character to distinguish it from another letter or to indicate an accent. d) True. e) False. A Unicode escape sequence is denoted by \u*yyyy*.

Exercises

I.3 Navigate to the Unicode Consortium Web site (`www.unicode.org`) and write the hexadecimal code values for the following characters. In which block were they located?

a) Latin letter "Z."
b) Latin letter "n" with the "tilde (~)."
c) Greek letter "delta."
d) Mathematical operator "less than or equal to."
e) Punctuation symbol "open quote (")."

I.4 Describe the Unicode Standard design basis.

I.5 Define the following terms:

a) code value.
b) surrogates.
c) Unicode Standard.

I.6 Define the following terms:

a) UTF-8.
b) UTF-16.
c) UTF-32.

I.7 Describe a scenario where it is optimal to store your data in UTF-16 format.

I.8 Using the Unicode Standard code values, write a Java program that prints your first and last name. The program should print your name in all uppercase letters and in all lowercase letters. If you know other languages, print your first and last name in those languages as well.