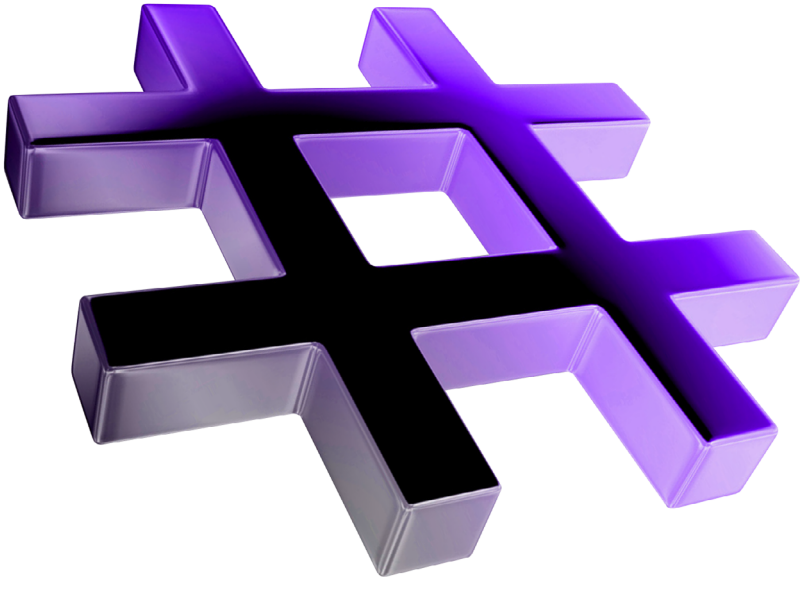
F

Unicode®

Objectives

In this appendix you'll learn:

- The mission of the Unicode Consortium.
- The design basis of Unicode.
- The three Unicode encoding forms: UTF-8, UTF-16 and UTF-32.
- Characters and glyphs.
- The advantages and disadvantages of using Unicode.



Outline

F.1 Introduction

The use of inconsistent character **encodings** (i.e., numeric values associated with characters) in the developing of global software products causes serious problems, because computers process information as numbers. For instance, 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 their own 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"; the Apple Macintosh operating system assigns that same value to an upside-down question mark. This results in the misrepresentation and possible corruption of data when it is not processed as intended.

In the absence of a widely implemented 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 (such as the conversion of numeric values and the underlying assumptions made by programmers), which results in increased costs and delays releasing the software. For example, some English-speaking programmers might design global software products assuming that a single character can be represented by one byte. However, when those products are localized for Asian markets, the programmer's assumptions are no longer valid; thus, the majority, if not the entirety, of the code needs to be rewritten.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, may 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 that 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 nonprofit organization called the **Unicode Consortium**, whose members include Apple, IBM, Microsoft, Oracle, Sybase and many others.

When the Consortium envisioned and developed the Unicode Standard, they 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 easily. 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**.

F.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 world's characters. Unicode extends the ASCII character set by encoding the vast majority of the world's characters. The Unicode Standard encodes all of those 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**, depending on the size of the units—in bits—being used. 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, where 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 website 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 computer systems and business protocols, 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. Also, UTF-32 requires less sophisticated handling than UTF-

16 in the presence of surrogate pairs. Figure F.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. F.1 | Correlation between the three encoding forms.

F.3 Characters and Glyphs

The Unicode Standard consists of characters, written components (i.e., alphabetic letters, numerals, punctuation marks, accent marks, and so on) that can be represented by numeric values. Examples of characters include: 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 with **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. F.2.

The Unicode Standard encompasses the alphabets, ideographs, syllabaries, punctuation marks, **diacritics**, mathematical operators and so on 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 94,140 character representations, with more than 880,000 code values reserved for future expansion.

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

F.4 Advantages/Disadvantages of Unicode

The Unicode Standard has several significant advantages that promote its use. One is its impact 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 all characters 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 via a consistent process.

Another advantage of the Unicode Standard is portability (i.e., the ability to execute software on disparate computers or with disparate operating systems). Most operating systems, databases, programming languages and web browsers currently support, or are planning to support, Unicode. Additionally, Unicode includes more characters than any other character set in common use (although it does not yet include all of the world's characters).

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. However, the **double-byte character set (DBCS)** and the **multibyte character set (MBCS)** that encode Asian characters (ideographs) require two to four bytes, respectively. In such instances, the UTF-16 or the UTF-32 encoding forms may be used with little hindrance to memory and performance.

F.5 Using Unicode

Visual Studio uses Unicode UTF-16 encoding to represent all characters. Figure F.3 uses C# to display the text "Welcome to Unicode!" in eight different languages: English, French, German, Japanese, Portuguese, Russian, Spanish and Traditional Chinese.

The first welcome message (lines 19–23) contains the hexadecimal codes for the English text. The **Code Charts** page on the Unicode Consortium website contains a document that lists the code values for the **Basic Latin** block (or category), which includes the English alphabet. The hexadecimal codes in lines 19–21 equate to "Welcome." When using Unicode characters in C#, the format `\u`*yyyy* is used, where *yyyy* represents the hexadecimal Unicode encoding. For example, the letter "W" (in "Welcome") is denoted by `\u0057`.

```
1  // Fig. F.3: UnicodeForm.cs
2  // Unicode enconding demonstration.
3  using System;
4  using System.Windows.Forms;
5
6  namespace UnicodeDemo
7  {
8     public partial class UnicodeForm : Form
9     {
10       public UnicodeForm()
11       {
12          InitializeComponent();
13       }
14
15       // assign Unicode strings to each Label
16       private void UnicodeForm_Load(object sender, EventArgs e)
17       {
18          // English
19          char[] english = { '\u0057', '\u0065', '\u006C',
```

Fig. F.3 | Unicode enconding demonstration. (Part 1 of 3.)

```
            '\u0063', '\u006F', '\u006D', '\u0065', '\u0020',
            '\u0074', '\u006F', '\u0020' };
         englishLabel.Text = new string(english) +
            "Unicode" + '\u0021';


         // French
         char[] french = { '\u0042', '\u0069', '\u0065',
            '\u006E', '\u0076', '\u0065', '\u006E', '\u0075',
            '\u0065', '\u0020', '\u0061', '\u0075', '\u0020' };
         frenchLabel.Text = new string(french) +
            "Unicode" + '\u0021';

         // German
         char[] german = { '\u0057', '\u0069', '\u006C',
            '\u006B', '\u006F', '\u006D', '\u006D', '\u0065',
            '\u006E', '\u0020', '\u007A', '\u0075', '\u0020' };
         germanLabel.Text = new string(german) +
            "Unicode" + '\u0021';

         // Japanese
         char[] japanese = { '\u3078',  '\u3087', '\u3045',
            '\u3053', '\u305D', '\u0021' };
         japaneseLabel.Text = "Unicode" + new string(japanese);

         // Portuguese
         char[] portuguese = { '\u0053', '\u0065', '\u006A',
            '\u0061', '\u0020', '\u0062', '\u0065', '\u006D',
            '\u0020', '\u0076', '\u0069', '\u006E', '\u0064',
            '\u006F', '\u0020', '\u0061', '\u0020' };
         portugueseLabel.Text = new string(portuguese) +
            "Unicode" + '\u0021';

         // Russian
         char[] russian = { '\u0414', '\u043E', '\u0431',
            '\u0440', '\u043E', '\u0020', '\u043F', '\u043E',
            '\u0436', '\u0430', '\u043B', '\u043E', '\u0432',
            '\u0430', '\u0442', '\u044A', '\u0020', '\u0432', '\u0020' };
         russianLabel.Text = new string(russian) +
            "Unicode" + '\u0021';

         // Spanish
         char[] spanish = { '\u0042', '\u0069', '\u0065',
            '\u006E', '\u0076', '\u0065', '\u006E', '\u0069',
            '\u0064', '\u006F', '\u0020', '\u0061', '\u0020' };
         spanishLabel.Text = new string(spanish) +
            "Unicode" + '\u0021';

         // Simplified Chinese
         char[] chinese = { '\u6B22', '\u8FCE', '\u4F7F',
            '\u7528', '\u0020' };
         chineseLabel.Text = new string(chinese) +
            "Unicode" + '\u0021';
      } // end method UnicodeForm_Load
```

Fig. F.3 | Unicode enconding demonstration. (Part 2 of 3.)

```
73      } // end class UnicodeForm
74  } // end namespace UnicodeDemo
```

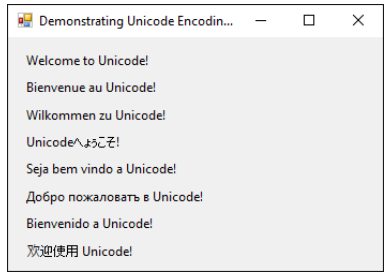


Fig. F.3 | Unicode enconding demonstration. (Part 3 of 3.)

Line 9 contains the hexadecimal for the *space* character (`\u0020`). The hexadecimal value for the word "to" is on line 21, and the word "Unicode" is on line 23. "Unicode" is not encoded because it is a registered trademark and has no equivalent translation in most languages. Line 23 also contains the `\u0021` notation for the exclamation mark (`!`).

The remaining welcome messages (lines 26–71) contain the hexadecimal codes for the other seven languages. The code values used for the French, German, Portuguese and Spanish text are located in the **Basic Latin** block, the code values used for the Traditional Chinese text are located in the **CJK Unified Ideographs** block, the code values used for the Russian text are located in the **Cyrillic** block and the code values used for the Japanese text are located in the **Hiragana** block.

[*Note:* To render the Asian characters in an application, you need to install the proper language files on your Windows computer. To do this, open the **Regional Options** dialog from the **Control Panel** (**Start > Settings > Control Panel**). At the bottom of the **General** tab is a list of languages. Check the **Japanese** and the **Traditional Chinese** checkboxes and press **Apply**. Follow the directions of the install wizard to install the languages. For more information, visit `www.unicode.org/help/display_problems.html`.]

F.6 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 94,140 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, and so on). 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, and so on). The **Code Charts** page on the Uni-

code Consortium website lists all the defined blocks and their respective code values. Figure F.4 lists some blocks (scripts) from the website 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+9FAF
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

Fig. F.4 | Some character ranges. (Part 1 of 2.)

Script	Range of code values
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. F.4 | Some character ranges. (Part 2 of 2.)

Summary

- Before Unicode, software developers were plagued by the use of inconsistent character encoding. Most countries and organizations had their own encoding systems, which were incompatible. A good example is the individual encoding systems on the Windows and Macintosh platforms.
- Computers process data by converting characters to numeric values. For instance, the character "a" is converted to a numeric value so that a computer can manipulate that piece of data.
- Without Unicode, localization of global software requires significant modifications to the source code, which results in increased cost and delays in releasing the product.

- 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, is 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.
- The Unicode Consortium developed the Unicode Standard in response to the serious problems created by multiple character encodings and the use of those encodings.
- The Unicode Standard facilitates the production and distribution of localized software. It outlines a specification for the consistent encoding of the world's characters and symbols.
- Software products that 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.
- The Unicode Standard is designed to be universal, efficient, uniform and unambiguous.
- A universal encoding system encompasses all commonly used characters; an efficient encoding system parses text files easily; a uniform encoding system assigns fixed values to all characters; and an unambiguous encoding system represents the same character for any given value.
- Unicode extends the limited ASCII character set to include all the major characters of the world.
- Unicode makes use of three Unicode Transformation Formats (UTF): UTF-8, UTF-16 and UTF-32, each of which may be appropriate for use in different contexts.
- 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, where there is a predominance of one-byte characters (ASCII represents characters as one byte).
- UTF-8 is a variable-width encoding form that is more compact for text involving mostly Latin characters and ASCII punctuation.
- UTF-16 is the default encoding form of the Unicode Standard. It is a variable-width encoding form that uses 16-bit code units instead of bytes. Most characters are represented by a single unit, but some characters require surrogate pairs.
- Surrogates are 16-bit integers in the range D800 through DFFF, which are used solely for the purpose of "escaping" into higher-numbered characters.
- Without surrogate pairs, the UTF-16 encoding form can encompass only 65,000 characters, but with the surrogate pairs, this is expanded to include over a million characters.
- UTF-32 is a 32-bit encoding form. The major advantage of the fixed-width encoding form is that it uniformly expresses all characters, so that they're easy to handle in arrays and so forth.
- The Unicode Standard consists of characters. A character is any written component that can be represented by a numeric value.
- Characters are represented with glyphs (various shapes, fonts and sizes for displaying characters).
- Code values are bit combinations that represent encoded characters. The Unicode notation for a code value is U+*yyyy*, in which U+ refers to the Unicode code values, as opposed to other hexadecimal values. The *yyyy* represents a four-digit hexadecimal number.
- Currently, the Unicode Standard provides code values for 94,140 character representations.
- An advantage of the Unicode Standard is its impact on the overall performance of the international economy. Applications that conform to an encoding standard can be processed easily by computers anywhere.
- Another advantage of the Unicode Standard is its portability. Applications written in Unicode can be easily transferred to different operating systems, databases, web browsers and so on. Most companies currently support, or are planning to support, Unicode.

- To obtain more information about the Unicode Standard and the Unicode Consortium, visit `www.unicode.org`. It contains a link to the code charts, which contain the 16-bit code values for the currently encoded characters.
- The Unicode Standard has become the default encoding system for XML and any language derived from XML, such as XHTML.
- The C# IDE uses Unicode UTF-16 encoding to represent all characters.
- In the marking up of C# documents, the entity reference \u*yyyy* is used, where *yyyy* represents the hexadecimal code value.

Terminology

\u*yyyy* notation
ASCII
block
character
character set
code value
diacritic
double-byte character set (DBCS)
efficient (Unicode design basis)
encode
entity reference
glyph
hexadecimal notation
localization
multibyte character set (MBCS)
portability
script
surrogate
symbol
unambiguous (Unicode design basis)
Unicode Consortium
Unicode design basis
Unicode Standard
Unicode Transformation Format (UTF)
uniform (Unicode design basis)
universal (Unicode design basis)
UTF-8
UTF-16
UTF-32

Self-Review Exercises

F.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 bases that constitute the Unicode Standard are: _____, _____, _____ and _____.
d) A(n) _____ is the smallest written component that can be represented with a numeric value.
e) Software that can execute on different operating systems is said to be _____.

F.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 C# programs, the entity reference is denoted by `#u`*yyyy*.

Answers to Self-Review exercises

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

F.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. The entity reference is denoted by `\u`*yyyy*.

Exercises

F.3 Navigate to the Unicode Consortium website (`www.unicode.org`) and write the hexadecimal code values for the following characters. In which block are 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 (").'

F.4 Describe the Unicode Standard design basis.

F.5 Define the following terms:

a) code value.
b) surrogates.
c) Unicode Standard.
d) UTF-8.
e) UTF-16.
f) UTF-32.

F.6 Describe a scenario where it is optimal to store your data in UTF-16 format.

F.7 Using the Unicode Standard code values, create a program that prints your first and last name. If you know other writing systems, print your first and last name in those as well. Use a `Label` to display your name.

F.8 Write an ASP.NET program that prints "Welcome to Unicode!" in English, French, German, Japanese, Portuguese, Russian, Spanish and Traditional Chinese. Use the code values provided in Fig. F.3. In ASP.NET, a code value is represented the same way as in a Windows application (`\u`*yyyy*, where *yyyy* is a four-digit hexadecimal number).