
Worldwide Software Development

Macintosh worldwide system software is designed to address the complex problems you'll encounter when you design your applications to be compatible with regional, linguistic, and script differences around the globe. Worldwide system software consists of the Macintosh Script Management System (which is one or more script systems and the **Script Manager**) and related components that include the **International Utilities Package Data**, the international resources, and keyboard resources.

The Macintosh computer has always presented one of the most flexible architectures for developing worldwide software. Because of the enhanced support for script systems in system 7.0, it's easier for users to add one or more non-Roman script systems to their Macintosh computers. With system 7.0, software can be localized with greater ease. Now it's even more advantageous for you to create applications that can be used worldwide.

It's much easier to design software with worldwide support from the beginning of your development process. This may mean that you create your application so that it is easy to localize, or that you adapt it for use in a specific area. Localizing software involves translating an application's menus, dialog boxes, alert boxes, and content areas into a language or regional dialect.

You can also make your application **Script Manager**-compatible. The **Script Manager** routines and the **International Utilities Package** handle text issues for all script systems. If your application is not text-oriented but does simple text processing, using **TextEdit** provides adequate support.

If your application does moderate text processing, such as that accomplished by a simple word processor, you probably want to incorporate **Script Manager** capabilities. If it does intensive text processing, such as page layout, you can build in support beyond the **Script Manager** routines to handle text for a specific script system.

The following sections outline the major issues you need to consider when you develop software for local or worldwide use.

For a complete description of the issues and a discussion of technical implementation, see the **TextEdit** and **Worldwide Software Overview** sections. These sections discuss the routines that assist you in developing your application for worldwide use. See *Macintosh Worldwide Development: Guide to System Software* for a complete discussion of developing worldwide software. This book is available from APDA.

Cultural Values

Whenever you design a user interface, consider that differences exist in the use of color, graphics, calendars, text, and the representation of time in various regions around the world. It's important that you be able to localize your user interface elements with ease. As an example, consider how different cultures assign different meanings to colors. The color white represents purity in one culture and death in another. Therefore you may want to localize elements of the user interface, such as the colors of text or graphics, in versions of your application designed for different regions.

Graphics have the potential to enhance your application, but they can also be offensive. In addition to colors, many cultures assign varying values and characteristics to living creatures, plants, and inanimate objects. In the United States the owl is a symbol of wisdom and knowledge, whereas in Central America the owl represents witchcraft and black magic. Some cultures forbid the depiction of uncovered bodies and body parts, while other cultures enhance marketing materials with pictures of scantily clad people. It's a good idea to avoid the use of seasons, holidays, or calendar events in software that you expect to distribute worldwide. Also avoid using graphics that represent holidays or seasons, such as Christmas trees, pumpkins, or snow-or be sure that the symbols can be localized. You can influence your audience in simple but profound ways by carefully selecting elements of your application's interface. Make sure that visible interface elements can be localized for other regions around the world.

Different calendars are used to mark time around the world. The United States and most of Europe observe time according to the Gregorian calendar. The traditional Arabic calendar, the Jewish calendar, and the Chinese calendar are lunar rather than solar. Often time is marked one way for business and government purposes while religious events are dated according to a different calendar. Therefore your application should be flexible in handling dates, and you may want to provide the user with a way to change the representation of time. To handle numbers, dates, and sorting, use the

International Utilities Package

Resources

It's essential to store region-dependent information in resources so that text the user sees can be translated (during localization) without modifying your application's code. When you create resources, consider text size, location, and direction. Remember that text size varies in different languages. Also, depending on the script system, the direction of text may change. Most Middle Eastern languages read from right to left instead of left to right, the direction of Roman script. Text location within a window should be easy to change.

Use the Macintosh Script Management System to handle these situations. See the **Worldwide Software Overview**, **Compatibility Guidelines**, and **Resource Manager** for more information on using resources to store data the user sees. Also consult *Macintosh Worldwide Development: Guide to System Software* for more information.

Language Differences

Languages differ in grammar, structure, meaning, and nuance. Translating languages is a delicate task and often can cause confusion, so be wary of using colloquial phrases or nonstandard usage and syntax. Choose your words carefully for command names in menus and for messages in dialog boxes, alert boxes, and help balloons. When translated, text can become up to 50 percent larger than U.S. English text, so you cannot rely on string length. Text needs room to grow up, down, and sideways.

Potential grammar problems may arise with error messages and the so-called user programming structure of languages like HyperTalk. The word order of messages may be completely different in translation, thus rendering a message nonsense when translated. Simple concatenation of strings generally

does not work when an application is translated. For example, word order in German usually places the verb at the end of a sentence. Suppose a German developer built an application that concatenated two strings to create an error message. When localized for the United States, the application might produce a sentence like "The file with the long name move." Instead of concatenating strings, use the **ReplaceText** function, which correctly assists with the syntactic ordering of elements. See the **Worldwide Software Overview** for information on technical implementation.

Text Display and Text Editing

System 7.0 allows users to display different scripts at the same time. A script is a writing system for a human language. Scripts may differ in the direction in which their characters and lines run, the size of the character set used to represent the script, and context dependence. Whenever a user installs a non-Roman script system, at least two scripts are available, the Roman script that is present on all Macintosh computers and the non-Roman script. If you use the **TextEdit** routines and the **Dialog Manager** routines, you can correctly handle most text in different scripts. For moderate text processing, the routines provided by the **Script Manager** can assist you in implementing these guidelines. The **TextEdit** and **Worldwide Software Overview** discuss all of these issues thoroughly.

No matter what level of worldwide support you provide, it's important to avoid two common assumptions. Characters are not necessarily 1 byte; they can be 2 bytes. You also should not assume that text is always left-aligned and read from left to right.

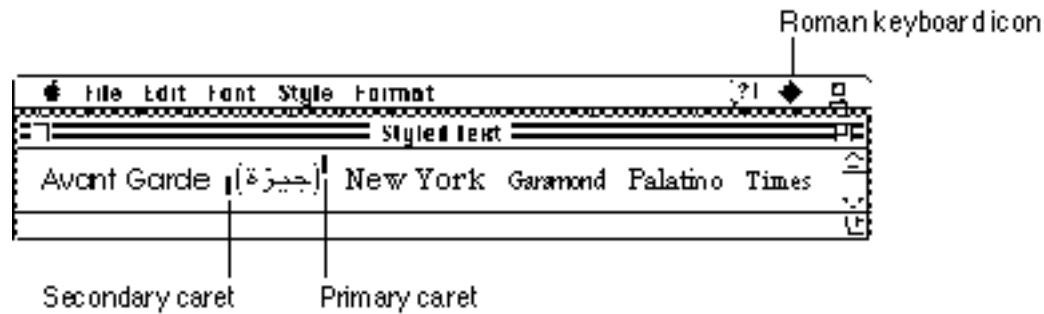
Remember that the meaning of a character code depends on the font, and character codes may be 1 or 2 bytes long. The cursor should move between characters, not bytes, and the Delete key should erase characters, not bytes. Inserted characters should appear between other characters, not between bytes of a 2-byte character. Also be aware of the impact of 2-byte characters on data transmission.

Use the language-specific routines in the Macintosh Script Management System for breaking and wrapping words and for string comparison and sorting. Consider word boundaries and their impact on word wrapping, selection, search, and cut and paste.

Some scripts include multiple sets of numerals. For example, international business in Japan and the Middle East requires the use of Western digits as well as the digits from a Japanese script or an Arabic script. Applications that handle numbers should accept all the numerals in each set as valid. Use the **International Utilities Package** to handle numbers.

You need to provide metric and English measurements. Use numeric routines for international number formatting and interpretation.

Your application should appropriately position the cursor when the user clicks in text. The cursor, or caret, should appear where the next character will appear when typed. If this is ambiguous because of multidirectional text, use dual carets, as shown in The Figure below. For a detailed discussion of using dual carets, see **TextEdit**.



Dual carrets in mixed-directional text

Highlighting should apply to a contiguous set of characters in memory, even though the glyphs may not appear contiguous on the screen. In other words, you should highlight characters in phonetic order (the order in which the user speaks, reads, or writes) rather than the order in which characters appear on screen. However, the arrow keys should move the cursor in the direction that the arrow points, regardless of text direction. This guideline applies across script boundaries when the user displays multiple scripts. A general rule is that the words are highlighted in the order that the user reads, from right to left.

Note: If your application uses **TextEdit** routines, most worldwide text issues are handled for you. If your application needs more sophisticated text handling, you should also consult the **Worldwide Software Overview**.

Applications that work with tokens (abstractions that have multiple representations) or use characters that vary from script system to script system should work correctly in all scripts. For example, a token that represents the concept of "less than or equal to" might have two representations on a U.S. system, the 2-byte sequence `<=` or the 1-byte character `≤`. If you use the **IntlTokenize** function to handle these details, your application does not have to be aware of the character codes.

Default Alignment of Interface Elements

When dialog boxes are localized for use with worldwide versions of system software, the text in the dialog box may become longer or shorter. Also, the alignment of controls in the dialog box may vary with localization. Arabic and Hebrew are written right to left, so the alignment of items in an Arabic or a Hebrew dialog box is generally right to left, just as dialog box items in English or Russian are generally left to right. The low-memory global variable **TESysJust** controls the alignment of interface elements.

When **TESysJust** is -1, the **Control Manager** reverses the alignment of check boxes and radio buttons, the **Menu Manager** reverses the alignment of menu items to be ordered and aligned on the right, and **TextEdit** aligns text by default on the right. Create your application so that it supports both left alignment and right alignment of controls and adjust the alignment as appropriate. Provide a way for the user to change the default line direction of text. Use the **SetSysJust** procedure to set the value of the global variable **TESysJust**.

When the alignment of items is reversed, it's important that the elements

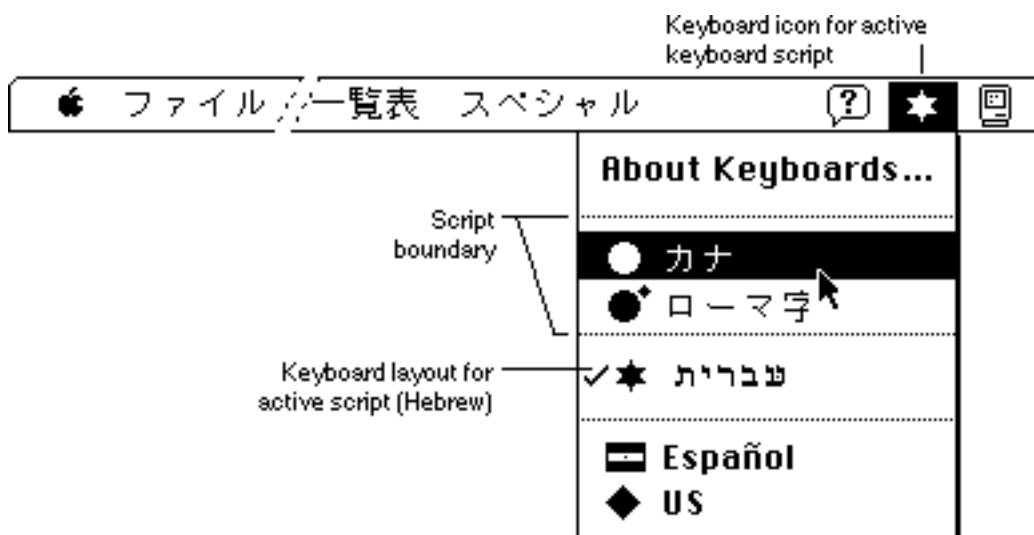
appear symmetrical. Therefore when you create dialog box items, try to make sure that their display rectangles are the same size. The Figure below shows a typical dialog box and the same dialog box with the alignment of its elements reversed. You can see why it's important to create display rectangles of the same size.

Keyboards

As stated previously, in system 7.0 users can install multiple script systems. If the Operating System or an application determines that all conditions are met, it enables the script system, making it available to users. A script system can contain more than one keyboard layout that maps character codes to keys on a physical keyboard, and it can support more than one attached physical keyboard. All keyboards do not have the same set or number of keys and users may have more than one keyboard attached to their computer. See the **Worldwide Software Overview** for information on installing and enabling script systems and keyboard resources.

System 7.0 adds a new Keyboard menu when more than one script system is present or a localizable resource flag is set. This menu simplifies the user's access to scripts and keyboards. The icon for the Keyboard menu appears between the icons for the **Help** menu and the **Application** menu. A keyboard icon appears next to each keyboard name, and the icon of the active keyboard appears in the menu bar. The Keyboard menu displays a list of installed keyboard layouts for each enabled script system.

The Keyboard menu groups the keyboard layouts by script system, which are separated by dotted or gray lines. In the Figure below, there are two Roman keyboard layouts (Spanish and United States); a single Hebrew keyboard layout; and two Japanese keyboard layouts. Only one keyboard layout and one physical keyboard are active at a time; the active condition is indicated by a checkmark in the menu.



The Keyboard menu



















Users can change keyboard layouts by using this menu or by using a keyboard equivalent, Command-Space bar, to cycle through the keyboard layouts. Do not use the keyboard equivalents Command-Space bar and Command-*modifier* key-Space bar in your application, since they are reserved for use by the **Script Manager**. See **Keyboard Equivalents** under the section, **Menus**

for a complete listing of reserved keyboard equivalents.

The Table below shows some new black-and-white versions of keyboard icons for localized versions of Macintosh system software. A keyboard icon represents a localized keyboard layout. If you develop keyboards or keyboard resources, you must provide customized icons like these. You create a 16-by-16 pixel icon in 1-bit, 4-bit, and 8-bit color. If you use the same colors for the 4-bit and the 8-bit color icons, you only need to provide one 4-bit icon. This scheme takes up less space in the System file.

To represent your keyboard layout for system 7.0, replace the black-and-white symbol you previously used to represent a localized keyboard layout with an icon similar to those shown in the Table below

Examples of keyboard icons

Icon	Name
	Arabic
	Canada
	Cyrillic
	Cyrillic transliterated
	Denmark
	Faeroe Islands
	Germany
	Hebrew
	Japanese Romaji
	Japanese Katakana
	Korean
	Netherlands, period decimal separator (previously )
	Netherlands, comma decimal separator
	Roman (U.S.)
	Spain
	Swiss French
	Swiss German



Swiss Italian



Turkey



Turkish U.S. modified



United Kingdom (previously)



United States

If you are designing a new keyboard icon, use a solid symbol to represent a keyboard layout for a region that is larger or smaller than a country or province. For example, a diamond represents the Roman Script System, which is used in the United States, Central America, and most of Europe. Use the flag of a country or province if the keyboard layout is only used in that area. For example, the Union Jack represents the keyboard layout localized for use in the United Kingdom. Be sure to use the colors that appear on the nation's flag. You can also add a visual indicator to the flag to show some modification. Use a superscript diamond to indicate a QWERTY transliteration, which is a mapping of sounds from a language to the Roman keyboard layout. Use a subscript comma or period to indicate which decimal separator is used.

When you design the black-and-white version of a flag icon, use black and a 50 percent gray pattern. These choices provide the best contrast and legibility. To avoid confusion between flags of similar design, use the pattern substitutions for colors shown in the Table.

Table Pattern substitutions for colors in keyboard icons

Pattern Color



Black

Black or blue



50 percent gray

Red



25 percent gray

Light blue



Diagonal stripes

Green



White

White or yellow

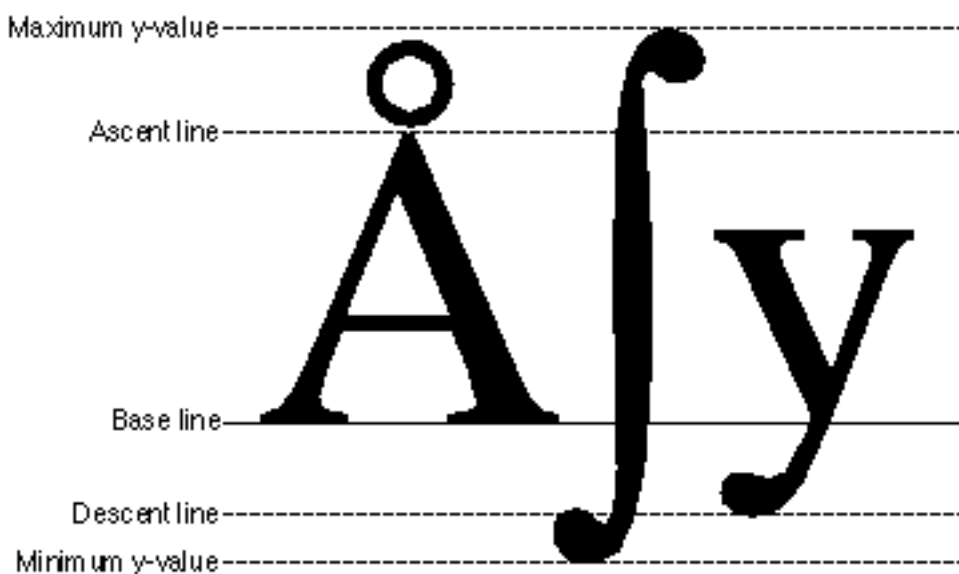
When the user changes the keyboard layout, you should synchronize the font to that keyboard layout. You can use the **FontScript** function to periodically poll the Operating System to find out if the user has changed the keyboard layout. Choosing a font should set the keyboard layout to the script of that font. For example, if a user chooses a Japanese font such as Osaka, your application should change the keyboard script to Japanese. When a user clicks in text, your application should set the keyboard layout to correspond to the font of that text. For a well-designed application, the keyboard icon in the menu bar should always indicate the status of the font script. **TextEdit** provides an example of automatically synchronizing the font and the keyboard layout.

See the **Worldwide Software Overview** for more information on the Keyboard menu.

Fonts

When you write software that supports non-Roman scripts, do not make assumptions about font sizes; let the user choose them. For example, system or application fonts may be preset to 12 or 18 points and a font with a resource ID of 0 is not always set to Chicago. Pay attention to the use of system and application fonts when the user cannot choose the font. If you must assign font sizes, use the **Script Manager** to find appropriate fonts and sizes. Use the proper font names as defined by worldwide system software. Whenever possible, display font names in the proper script and font in your **Font** menu.

Diacritical marks may extend beyond the ascent line. Some fonts, such as Japanese fonts, contain glyphs that extend to the boundaries of the enclosing rectangle of the font, or to *both* minimum-y and maximum-y lines. You should leave room for space between lines of text and between the top and bottom lines of any enclosing rectangle. See the **Font Manager** for more information. The Figure below shows some glyphs that demonstrate the boundaries you need to allow for lines of text.



The boundaries of a font