About the Script Manager

The **Script Manager** is a set of general text manipulation routines that let applications function correctly with non-Roman scripts and writing styles. This not only includes languages with different alphabets, like Japanese, but also such languages as French and German that use the Latin alphabet, but in ways different from English. To achieve its ability to work in many different systems, the Script Manager employs modules, called Script Interface Systems. Individual modules contain the rules for specific writing systems. You would need the routines in Script Manager if you're writing such applications as a word processor.

Adapting an application to different languages is called "localization" and is detailed more thoroughly in Apple's publication *Human Interface Guidelines.*

The Script Manager's procedures are designed to provide standard tools for manipulating ordinary text and to help you translate your application into another writing system. They include utilities and initialization code to create an environment in which scripts of all kinds can be handled.They also depend on the presence of an Apple Script Interface System, such as the Roman, Kanji, Arabic and Chinese (Hanze) Interface Systems in either the System file or in ROM.

Script Interface Systems provide fonts for the target language, keyboard mapping, character input, conversion, sorting and text manipulation routines, and a desk accessory for system maintenance and control. The Script Manager goes through the Script Interface System to adapt individual calls to particular written languages. An application calls the Script Manager, the Script Manager calls the Script Interface System, the Script Interface System performs the action and returns a result to the Script Manager and the Script Manager returns a result to the application. Up to 64 different Script Interface Systems can be installed on a Macintosh so applications can switch from one form of writing to another. When two or more Script Interface Systems are installed, an icon symbolizing the script currently in use appears at the right side of the men bar.

The Script Manager takes care of making sure that such elements as character set size, writing direction, context dependence, word demarcation and text justification are appropriate for various writing systems.

For text manipulation, the characteristics of different scripts force text to be handled in different ways. Defined scripts and their identification numbers include:

Constant	**Value**	**Script**
smRoman	0	ASCII
smKanji	1	Japanese
smChinese	2	Chinese
smKorean	3	Korean
smArabic	4	Arabic
smHebrew	5	Hebrew
smGreek	6	Greek
smRussian	7	Cyrillic
smReserved1	8	Reserved
smDevanagari	9	Devanagari

smGurmukhi	10	Gurmukhi
smGujarati	11	Gujarati
smOriya	12	Oriya
smBengali	13	Bengali
smTamil	14	Tamil
smTelugu	15	Telugu
smKannada	16	Kannada
smMalayalam	17	Malayalam
smSinhalese	18	Sinhalese
smBurmese	19	Burmese
smKhmer	20	Cambodian
smThai	21	Thai
smLaotian	22	Laotian
smGeorgian	23	Georgian
smArmenian	24	Armenian
smMaldivian	25	Maldivian
smTibetan	26	Tibetan
smMongolian	27	Mongolian
smAmharic	28	Ethiopian
smSlavic	29	Non-Cyrillic Slavic
smVietnamese	30	Vietnamese
smSindhi	31	Sindhi
smUninterp	32	Uninterpreted symbols

The Script Manager will go to the font field of the grafPort to look for one of these values. The script it finds is called the font script. A different script, the key script, determines the keyboard layout and input method but doesn't change the way characters are drawn on the screen. The key and font scripts can be different--as when an ASCII keyboard layout might be maintained for the convenience of a western European user writing in Arabic or Japanese.

The individual Script Interface Systems and the Script Manager work with standard Toolbox routines to make sure Roman and non-Roman text is handled properly. Special routines for justifying text operate like standard drawing and measuring routines, except that they have the extra ability to make the text expand evenly on the line.

For parsing, the fact that many non-Roman character sets use a double byte for some characters imposes special demands.   By using the **CharByte** routine you can make sure that a particular character is not simply a segment of a double-byte character.

If your application needs to use a character code for a non-character purpose (e.g., field delimiters) use codes below 0x020 since they aren't affected by Script Interface Systems. Some of those codes, however, have already been assigned:

Null=0
Enter=3
Backspace=8
Tab=9
Line feed=10
Carriage return=13
System characters=17, 18, 19 and 20
Clear=27
Cursor keys=28, 29, 30 and 31

Since double-byte characters are treated as two key-down events, you need to first use **CharByte** to see if the first byte is part of a character and, if so you need to buffer it and wait for the second byte. When the second byte arrives the character is complete and can be drawn. Let the application do the buffering. Otherwise **TextEdit** performance will slow down. Each time through the event loop, place every byte that results from a key press or auto-repeat into a buffer. Whenever the byte results from anything else, insert the buffer.

Writing direction is handled by the individual Script Interface Systems under user control. Changing writing direction also changes the justification of **TextEdit**, the **Menu Manager** and the **Control Manager**'s radio buttons and check boxes. If you have an application that allows the use of different writing systems on a single line, each piece of text should be drawn and measured in its own system to ensure that characters appear and line up correctly.

If you're going to provide an application that partitions or analyzes text, you need to be aware of situations that require extra care:

- Your application should know that a given character may not always have the same width. The **Font Manager**'s cached width tables should be set aside in favor of **QuickDraw**'s MeasureText routine.
- Your application needs to know that a monospaced font won't always produce monospaced text--as when non-Roman text characters are inserted in an Roman-based string of words.
- Your application should be capable of processing zero-width characters and such characters should never be separated from the previous character when hyphenating a word or wrapping text to the next line. Always truncate lines from the end toward the beginning, one byte at a time. With such an algorithm you avoid breaking zero-width characters away from required predecessors.

Use the Script Manager utilities when text is to be selected, searched or word-wrapped. They are devised to handle individual script requirements.