This appendix describes support for script-specific behavior that is built into Macintosh system software. The code and data described here can work with the resources of many script systems to give the script systems their unique behaviors.

For historical reasons, much of the behavior of the Roman script system is built into Macintosh system software code resources and ROM. The rest of its behavior is expressed in international and keyboard resources that are installed in every version of Macintosh system software. This appendix summarizes that behavior, which represents the default set of Macintosh text-handling features.

WorldScript I is a script extension, consisting of code that implements table-driven measuring and drawing behavior for all 1-byte complex script systems. Using tables in each script system's international resources, WorldScript I properly performs caret placement, hit-testing, justification, text layout, and drawing for all supported scripts. This appendix describes how WorldScript I works and how you can replace some of its individual routines.

WorldScript II is another script extension that implements table-driven measuring and drawing behavior for all 2-byte script systems. Like WorldScript I, WorldScript II uses tables in each script system's international resources to perform text manipulation properly for all supported scripts. This appendix describes how WorldScript II works; note, however, that you cannot replace any of its routines.

Read this appendix for information on Roman character encoding, U.S. Roman keyboard-layout resource features, and the default Roman sorting routines. Read this appendix also if you wish to understand how the WorldScript extensions work, and especially if you intend to replace or modify any of the WorldScript I routines.

Before reading this appendix, read the chapter "Introduction to Text on the Macintosh" in this book. If you intend to modify WorldScript I, read also the discussion on replacing a script system's default routines in the chapter "Script Manager" in this book. Additional information on sorting behavior can be found in the description of the string-manipulation resource in the appendix "International Resources" in this book. Additional information on the keyboard-layout resource can be found in the appendix "Keyboard Resources" in this book.

The Roman Script System

The Roman script system is available on all localized versions of Macintosh system software throughout the world. It is not entirely uniform; in different localized systems, the Roman script may have different features. Nevertheless, its character set and its sorting and formatting rules provide baselines that non-Roman script systems adopt, modify, or replace as their needs align with or diverge from Roman conventions.

The Standard Roman character set is implemented by the U.S. keyboard-layout (`'KCHR'`) resource—included with every Macintosh system—and by the keyboard-layout resources of other localized versions of the Roman script system (such as French or Spanish).

The standard U.S. Roman sorting routines are the basis for sorting strings composed of characters from the Standard Roman character set. The routines can be modified with code in a script system's string-manipulation resource; many non-U.S. Roman script systems and many non-Roman script systems override the standard U.S. routines.

The Standard Roman Character Set

The **Standard Roman character set** is an extended version of the **Macintosh character set**, documented in Volume I of the original *Inside Macintosh.* The Macintosh character set is itself an extended version of the **ASCII character set.** The conventional ASCII character set, also called *low ASCII,* defines control codes, symbols, numbers, and letters, assigning them character codes from $00 through $7F. The Macintosh character set adds codes from $80 through $D8, representing accented characters and additional symbols. Current Macintosh file-system sorting, as well as the sorting order used by several Text Utilities routines such as `RelString`, is based on the Macintosh character set.

The Standard Roman character set adds more accented forms, symbols, and diacritical marks, assigning them character codes from $D9 through $FF. It thus consists of all the character codes from $00–$FF, and it includes uppercase versions of all of the lowercase accented forms, a number of symbols, and other forms. See Figure A-1.

The Standard Roman character set is the closest to a universal character encoding that exists in the Roman script system. The Standard Roman characters are available in most Roman outline fonts, but not all are available in the Apple bitmapped versions of Chicago, Geneva, New York, and Monaco.

Figure A-1 The Standard Roman character set

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
x0	nul	dle	sp	0	@	P	`	p	A	ê	†	∞	¿	-	‡	
x1	soh	DC1	!	1	A	Q	a	q	Å	ë	°	±	¡	—	·	Ò
x2	stx	DC2	"	2	B	R	b	r	Ç	ì	¢	≤	¬	“	‚	Ú
x3	etx	DC3	#	3	C	S	c	s	É	ì	£	≥	√	”	„	Û
x4	eot	DC4	$	4	D	T	d	t	Ñ	Î	§	¥	ƒ	‘	‰	Ù
x5	enq	nak	%	5	E	U	e	u	Ö	ï	•	µ	≈	’	Â	ı
x6	ack	syn	&	6	F	V	f	v	Ü	ñ	¶	∂	∆	÷	Ê	ˆ
x7	bel	etb	'	7	G	W	g	w	á	ó	ß	∑	≪	◊	Á	˜
x8	bs	can	(	8	H	X	h	x	à	ò	®	∏	≫	ÿ	Ë	¯
x9	ht	em	)	9	I	Y	i	y	â	ô	©	π	…	Ÿ	È	˘
xA	lf	sub	*	:	J	Z	j	z	ä	ö	™	∫	nbsp	/	Í	˙
xB	vt	esc	+	;	K	[	k	{	ã	õ	´	ª	À	¤	Î	°
xC	ff	fs	,	<	L	\	l	|	å	ú	¨	º	Ã	<	Ï	¸
xD	cr	gs	-	=	M	]	m	}	ç	ù	≠	Ω	Õ	>	Ì	˝
xE	so	rs	.	>	N	^	n	~	é	û	Æ	æ	Œ	fi	Ó	˛
xF	si	us	/	?	O	_	o	del	è	ü	Ø	ø	œ	fl	Ô	ˇ

Nonprinting Characters

Table A-1 lists the nonprinting characters in the Standard Roman character set. The Unicode 1.0 name and the Macintosh character code (in hexadecimal and decimal) are provided also. **(Unicode** is an ISO standard for 16-bit universal worldwide character encoding.)

Table A-1 Nonprinting characters in the Standard Roman character set

Unicode name	Hexadecimal	Decimal
NULL	$00	0
START OF HEADING	$01	1
START OF TEXT	$02	2
END OF TEXT	$03	3
END OF TRANSMISSION	$04	4
ENQUIRY	$05	5
ACKNOWLEDGE	$06	6
BELL	$07	7
BACKSPACE	$08	8
HORIZONTAL TABULATION	$09	9
LINE FEED	$0A	10
VERTICAL TABULATION	$0B	11
FORM FEED	$0C	12
CARRIAGE RETURN	$0D	13
SHIFT OUT	$0E	14
SHIFT IN	$0F	15
DATA LINK ESCAPE	$10	16
DEVICE CONTROL ONE	$11	17
DEVICE CONTROL TWO	$12	18
DEVICE CONTROL THREE	$13	19
DEVICE CONTROL FOUR	$14	20
NEGATIVE ACKNOWLEDGE	$15	21

Table A-1 Nonprinting characters in the Standard Roman character set (continued)

Unicode name	Hexadecimal	Decimal
SYNCHRONOUS IDLE	$16	22
END OF TRANSMISSION BLOCK	$17	23
CANCEL	$18	24
END OF MEDIUM	$19	25
SUBSTITUTE	$1A	26
ESCAPE	$1B	27
FILE SEPARATOR	$1C	28
GROUP SEPARATOR	$1D	29
RECORD SEPARATOR	$1E	30
UNIT SEPARATOR	$1F	31
DELETE	$7F	127

Using Roman Character Codes as Delimiters

Your application may need to use a character code or range of codes to represent noncharacter data (such as field delimiters). Character codes below $20 are never affected by the script system. Some of these character codes can be used safely for special purposes. Note, however, that most characters in this range are already assigned special meanings by parts of Macintosh system software, such as TextEdit, or by programming languages like C. Table A-2 lists the low-ASCII characters to avoid in your application.

Table A-2 Low-ASCII characters to avoid as delimiters

Character	Hexadecimal representation
Null	$00
Home	$01
Enter	$03
End	$04
Help	$05
Backspace	$08
Tab	$09

continued

Table A-2 Low-ASCII characters to avoid as delimiters (continued)

Character	Hexadecimal representation
Page up	$0B
Page down	$0C
Carriage return	$0D
F1 through F15	$10
System characters	$11, $12, $13, $14*
Clear	$1B
Arrow keys	$1C, $1D, $1E, $1F

* System fonts use these codes for the printing characters PROPELLER, LOZENGE , RADICAL, and APPLE LOGO, respectively.

For certain writing systems, font layouts (tables that map glyph codes to glyphs) may use some of these character codes internally, for ligatures or other contextual forms. Also, as noted in Table A-2, system fonts use codes $11 through $14 for printing special symbols such as the Apple logo. Thus in unusual situations font changes may have an impact on the glyph representation of stored character codes with values less than $20, even though a user cannot generate those codes directly.

Printing Characters

Table A-3 shows the printing characters that exist in the Standard Roman character set. Macintosh applications can assume that glyphs for these characters exist in every Roman font. (However, see also the discussion of Roman fonts on page A-18.) The Unicode 1.0 and PostScript names and Macintosh character code (in hexadecimal and decimal) are provided along with a glyph example for printable characters. Modified versions of the Standard Roman character set exist for Croatian, Romanian, Turkish, and Icelandic/ Faroese, with different character assignments for the same codes. See Table A-4 through Table A-7.

Table A-3 Printing characters in the Standard Roman character set

Glyph	Unicode name	PostScript name	Hexadecimal	Decimal
	SPACE	space	$20	32
!	EXCLAMATION MARK	exclam	$21	33
"	QUOTATION MARK	quotedbl	$22	34
#	NUMBER SIGN	numbersign	$23	35
$	DOLLAR SIGN	dollar	$24	36
%	PERCENT SIGN	percent	$25	37
&	AMPERSAND	ampersand	$26	38
'	APOSTROPHE-QUOTE	quotesingle	$27	39
(	OPENING PARENTHESIS	parenleft	$28	40
)	CLOSING PARENTHESIS	parenright	$29	41
*	ASTERISK	asterisk	$2A	42
+	PLUS SIGN	plus	$2B	43
,	COMMA	comma	$2C	44
-	HYPHEN-MINUS	hyphen	$2D	45
.	PERIOD	period	$2E	46
/	SLASH	slash	$2F	47
0	DIGIT ZERO	zero	$30	48
1	DIGIT ONE	one	$31	49
2	DIGIT TWO	two	$32	50
3	DIGIT THREE	three	$33	51
4	DIGIT FOUR	four	$34	52
5	DIGIT FIVE	five	$35	53
6	DIGIT SIX	six	$36	54
7	DIGIT SEVEN	seven	$37	55
8	DIGIT EIGHT	eight	$38	56
9	DIGIT NINE	nine	$39	57
:	COLON	colon	$3A	58
;	SEMICOLON	semicolon	$3B	59
<	LESS-THAN SIGN	less	$3C	60
=	EQUALS SIGN	equal	$3D	61

continued

Table A-3 Printing characters in the Standard Roman character set (continued)

Glyph	Unicode name	PostScript name	Hexadecimal	Decimal
>	GREATER-THAN SIGN	greater	$3E	62
?	QUESTION MARK	question	$3F	63
@	COMMERCIAL AT	at	$40	64
A	LATIN CAPITAL LETTER A	A	$41	65
B	LATIN CAPITAL LETTER B	B	$42	66
C	LATIN CAPITAL LETTER C	C	$43	67
D	LATIN CAPITAL LETTER D	D	$44	68
E	LATIN CAPITAL LETTER E	E	$45	69
F	LATIN CAPITAL LETTER F	F	$46	70
G	LATIN CAPITAL LETTER G	G	$47	71
H	LATIN CAPITAL LETTER H	H	$48	72
I	LATIN CAPITAL LETTER I	I	$49	73
J	LATIN CAPITAL LETTER J	J	$4A	74
K	LATIN CAPITAL LETTER K	K	$4B	75
L	LATIN CAPITAL LETTER L	L	$4C	76
M	LATIN CAPITAL LETTER M	M	$4D	77
N	LATIN CAPITAL LETTER N	N	$4E	78
O	LATIN CAPITAL LETTER O	O	$4F	79
P	LATIN CAPITAL LETTER P	P	$50	80
Q	LATIN CAPITAL LETTER Q	Q	$51	81
R	LATIN CAPITAL LETTER R	R	$52	82
S	LATIN CAPITAL LETTER S	S	$53	83
T	LATIN CAPITAL LETTER T	T	$54	84
U	LATIN CAPITAL LETTER U	U	$55	85
V	LATIN CAPITAL LETTER V	V	$56	86
W	LATIN CAPITAL LETTER W	W	$57	87
X	LATIN CAPITAL LETTER X	X	$58	88
Y	LATIN CAPITAL LETTER Y	Y	$59	89
Z	LATIN CAPITAL LETTER Z	Z	$5A	90
[	OPENING SQUARE BRACKET	bracketleft	$5B	91
\	BACK SLASH	backslash	$5C	92

Table A-3 Printing characters in the Standard Roman character set (continued)

Glyph	Unicode name	PostScript name	Hexadecimal	Decimal
]	CLOSING SQUARE BRACKET	bracketright	$5D	93
^	SPACING CIRCUMFLEX	asciicircum	$5E	94
_	SPACING UNDERSCORE	underscore	$5F	95
`	SPACING GRAVE	grave	$60	96
a	LATIN SMALL LETTER A	a	$61	97
b	LATIN SMALL LETTER B	b	$62	98
c	LATIN SMALL LETTER C	c	$63	99
d	LATIN SMALL LETTER D	d	$64	100
e	LATIN SMALL LETTER E	e	$65	101
f	LATIN SMALL LETTER F	f	$66	102
g	LATIN SMALL LETTER G	g	$67	103
h	LATIN SMALL LETTER H	h	$68	104
i	LATIN SMALL LETTER I	i	$69	105
j	LATIN SMALL LETTER J	j	$6A	106
k	LATIN SMALL LETTER K	k	$6B	107
l	LATIN SMALL LETTER L	l	$6C	108
m	LATIN SMALL LETTER M	m	$6D	109
n	LATIN SMALL LETTER N	n	$6E	110
o	LATIN SMALL LETTER O	o	$6F	111
p	LATIN SMALL LETTER P	p	$70	112
q	LATIN SMALL LETTER Q	q	$71	113
r	LATIN SMALL LETTER R	r	$72	114
s	LATIN SMALL LETTER S	s	$73	115
t	LATIN SMALL LETTER T	t	$74	116
u	LATIN SMALL LETTER U	u	$75	117
v	LATIN SMALL LETTER V	v	$76	118
w	LATIN SMALL LETTER W	w	$77	119
x	LATIN SMALL LETTER X	x	$78	120
y	LATIN SMALL LETTER Y	y	$79	121
z	LATIN SMALL LETTER Z	z	$7A	122

continued

Table A-3 Printing characters in the Standard Roman character set (continued)

Glyph	Unicode name	PostScript name	Hexadecimal	Decimal
{	OPENING CURLY BRACKET	braceleft	$7B	123
|	VERTICAL BAR	bar	$7C	124
}	CLOSING CURLY BRACKET	braceright	$7D	125
~	TILDE	asciitilde	$7E	126
	DELETE (nonprinting)		$7F	127
Ä	LATIN CAPITAL LETTER A DIAERESIS	Adieresis	$80	128
Å	LATIN CAPITAL LETTER A RING	Aring	$81	129
Ç	LATIN CAPITAL LETTER C CEDILLA	Ccedilla	$82	130
É	LATIN CAPITAL LETTER E ACUTE	Eacute	$83	131
Ñ	LATIN CAPITAL LETTER N TILDE	Ntilde	$84	132
Ö	LATIN CAPITAL LETTER O DIAERESIS	Odieresis	$85	133
Ü	LATIN CAPITAL LETTER U DIAERESIS	Udieresis	$86	134
á	LATIN SMALL LETTER A ACUTE	aacute	$87	135
à	LATIN SMALL LETTER A GRAVE	agrave	$88	136
â	LATIN SMALL LETTER A CIRCUMFLEX	acircumflex	$89	137
ä	LATIN SMALL LETTER A DIAERESIS	adieresis	$8A	138
ã	LATIN SMALL LETTER A TILDE	atilde	$8B	139
å	LATIN SMALL LETTER A RING	aring	$8C	140
ç	LATIN SMALL LETTER C CEDILLA	ccedilla	$8D	141
é	LATIN SMALL LETTER E ACUTE	eacute	$8E	142
è	LATIN SMALL LETTER E GRAVE	egrave	$8F	143
ê	LATIN SMALL LETTER E CIRCUMFLEX	ecircumflex	$90	144
ë	LATIN SMALL LETTER E DIAERESIS	edieresis	$91	145
í	LATIN SMALL LETTER I ACUTE	iacute	$92	146
ì	LATIN SMALL LETTER I GRAVE	igrave	$93	147
î	LATIN SMALL LETTER I CIRCUMFLEX	icircumflex	$94	148
ï	LATIN SMALL LETTER I DIAERESIS	idiaresis	$95	149
ñ	LATIN SMALL LETTER N TILDE	ntilde	$96	150
ó	LATIN SMALL LETTER O ACUTE	oacute	$97	151
ò	LATIN SMALL LETTER O GRAVE	ograve	$98	152

Table A-3 Printing characters in the Standard Roman character set (continued)

Glyph	Unicode name	PostScript name	Hexadecimal	Decimal
ô	LATIN SMALL LETTER O CIRCUMFLEX	ocircumflex	$99	153
ö	LATIN SMALL LETTER O DIAERESIS	odieresis	$9A	154
õ	LATIN SMALL LETTER O TILDE	otilde	$9B	155
ú	LATIN SMALL LETTER U ACUTE	uacute	$9C	156
ù	LATIN SMALL LETTER U GRAVE	ugrave	$9D	157
û	LATIN SMALL LETTER U CIRCUMFLEX	ucircumflex	$9E	158
ü	LATIN SMALL LETTER U DIAERESIS	udieresis	$9F	159
†	DAGGER	dagger	$A0	160
°	DEGREE SIGN	degree	$A1	161
¢	CENT SIGN	cent	$A2	162
£	POUND SIGN	sterling	$A3	163
§	SECTION SIGN	section	$A4	164
•	BULLET	bullet	$A5	165
¶	PARAGRAPH SIGN	paragraph	$A6	166
ß	LATIN SMALL LETTER SHARP S	germandbls	$A7	167
®	REGISTERED TRADEMARK SIGN	registered	$A8	168
©	COPYRIGHT SIGN	copyright	$A9	169
™	TRADEMARK	trademark	$AA	170
´	SPACING ACUTE	acute	$AB	171
¨	SPACING DIAERESIS	dieresis	$AC	172
≠	NOT EQUAL TO	notequal	$AD	173
Æ	LATIN CAPITAL LETTER AE	AE	$AE	174
Ø	LATIN CAPITAL LETTER O SLASH	Oslash	$AF	175
∞	INFINITY	infinity	$B0	176
±	PLUS-OR-MINUS SIGN	plusminus	$B1	177
≤	LESS THAN OR EQUAL TO	lessequal	$B2	178
≥	GREATER THAN OR EQUAL TO	greaterequal	$B3	179
¥	YEN SIGN	yen	$B4	180
µ	MICRO SIGN	mu	$B5	181
∂	PARTIAL DIFFERENTIAL	partialdiff	$B6	182

continued

Table A-3 Printing characters in the Standard Roman character set (continued)

Glyph	Unicode name	PostScript name	Hexadecimal	Decimal
∑	N-ARY SUMMATION	summation	$B7	183
∏	N-ARY PRODUCT	product	$B8	184
π	GREEK SMALL LETTER PI	pi	$B9	185
∫	INTEGRAL	integral	$BA	186
ª	FEMININE ORDINAL INDICATOR	ordfeminine	$BB	187
º	MASCULINE ORDINAL INDICATOR	ordmasculine	$BC	188
Ω	OHM	Omega	$BD	189
æ	LATIN SMALL LETTER AE	ae	$BE	190
ø	LATIN SMALL LETTER O SLASH	oslash	$BF	191
¿	INVERTED QUESTION MARK	questiondown	$C0	192
¡	INVERTED EXCLAMATION MARK	exclamdown	$C1	193
¬	NOT SIGN	logicalnot	$C2	194
√	SQUARE ROOT	radical	$C3	195
ƒ	LATIN SMALL LETTER SCRIPT F	florin	$C4	196
≈	ALMOST EQUAL TO	approxequal	$C5	197
Δ	INCREMENT	Delta	$C6	198
«	LEFT POINTING GUILLEMET	guillemotleft	$C7	199
»	RIGHT POINTING GUILLEMET	guillemotright	$C8	200
…	HORIZONTAL ELLIPSIS	ellipsis	$C9	201
	NON-BREAKING SPACE		$CA	202
À	LATIN CAPITAL LETTER A GRAVE	Agrave	$CB	203
Ã	LATIN CAPITAL LETTER A TILDE	Atilde	$CC	204
Õ	LATIN CAPITAL LETTER O TILDE	Otilde	$CD	205
Œ	LATIN CAPITAL LETTER O E	OE	$CE	206
œ	LATIN SMALL LETTER O E	oe	$CF	207
–	EN DASH	endash	$D0	208
—	EM DASH	emdash	$D1	209
“	DOUBLE TURNED COMMA QUOTATION MARK	quotedblleft	$D2	210
”	DOUBLE COMMA QUOTATION MARK	quotedblright	$D3	211
‘	SINGLE TURNED COMMA QUOTATION MARK	quoteleft	$D4	212

Table A-3 Printing characters in the Standard Roman character set (continued)

Glyph	Unicode name	PostScript name	Hexadecimal	Decimal
’	SINGLE COMMA QUOTATION MARK	quoteright	$D5	213
÷	DIVISION SIGN	divide	$D6	214
◊	LOZENGE	lozenge	$D7	215
ÿ	LATIN SMALL LETTER Y DIAERESIS	ydieresis	$D8	216
Ÿ	LATIN CAPITAL LETTER Y DIAERESIS	Ydieresis	$D9	217
⁄	FRACTION SLASH	fraction	$DA	218
¤	CURRENCY SIGN	currency	$DB	219
‹	LEFT POINTING SINGLE GUILLEMET	guilsingleleft	$DC	220
›	RIGHT POINTING SINGLE GUILLEMET	guilsingleright	$DD	221
fi	(no Unicode designation)	fi	$DE	222
fl	(no Unicode designation)	fl	$DF	223
‡	DOUBLE DAGGER	daggerdbl	$E0	224
·	MIDDLE DOT	periodcentered	$E1	225
‚	LOW SINGLE COMMA QUOTATION MARK	quotesinglbase	$E2	226
„	LOW DOUBLE COMMA QUOTATION MARK	quotedblbase	$E3	227
‰	PER MILLE SIGN	perthousand	$E4	228
Â	LATIN CAPITAL LETTER A CIRCUMFLEX	Acircumflex	$E5	229
Ê	LATIN CAPITAL LETTER E CIRCUMFLEX	Ecircumflex	$E6	230
Á	LATIN CAPITAL LETTER A ACUTE	Aacute	$E7	231
Ë	LATIN CAPITAL LETTER E DIAERESIS	Edieresis	$E8	232
È	LATIN CAPITAL LETTER E GRAVE	Egrave	$E9	233
Í	LATIN CAPITAL LETTER I ACUTE	Iacute	$EA	234
Î	LATIN CAPITAL LETTER I CIRCUMFLEX	Icircumflex	$EB	235
Ï	LATIN CAPITAL LETTER I DIAERESIS	Idieresis	$EC	236
Ì	LATIN CAPITAL LETTER I GRAVE	Igrave	$ED	237
Ó	LATIN CAPITAL LETTER O ACUTE	Oacute	$EE	238
Ô	LATIN CAPITAL LETTER O CIRCUMFLEX	Ocircumflex	$EF	239
	APPLE LOGO	Apple	$F0	240
Ò	LATIN CAPITAL LETTER O GRAVE	Ograve	$F1	241
Ú	LATIN CAPITAL LETTER U ACUTE	Uacute	$F2	242
Û	LATIN CAPITAL LETTER U CIRCUMFLEX	Ucircumflex	$F3	243

continued

Table A-3 Printing characters in the Standard Roman character set (continued)

Glyph	Unicode name	PostScript name	Hexadecimal	Decimal
Ù	LATIN CAPITAL LETTER U GRAVE	Ugrave	$F4	244
ı	LATIN SMALL LETTER DOTLESS I	dotlessi	$F5	245
ˆ	MODIFIER LETTER CIRCUMFLEX	circumflex	$F6	246
˜	SPACING TILDE	tilde	$F7	247
¯	SPACING MACRON	macron	$F8	248
˘	SPACING BREVE	breve	$F9	249
˙	SPACING DOT ABOVE	dotaccent	$FA	250
°	SPACING RING ABOVE	ring	$FB	251
¸	SPACING CEDILLA	cedilla	$FC	252
˝	SPACING DOUBLE ACUTE	hungarumlaut	$FD	253
˛	SPACING OGONEK	ogonek	$FE	254
ˇ	MODIFIER LETTER HACEK	caron	$FF	255

Variations in the Character Set

Two types of variations from the Standard Roman character set can occur. First, several languages and regional variations of Roman reassign parts of the character set; second, many specialized Roman fonts completely override the character set to provide other types of symbols.

Table A-4 shows the glyph assignments in the Croatian version of the Roman character set that diverge from the Standard Roman character set, their Unicode and PostScript names, and their Macintosh character codes in hexadecimal and decimal. For example, the code (hexadecimal $A9) that is assigned to the copyright sign in the Standard Roman character set is replaced by the Scaron (that is, the Roman capital letter “S” with a hacek). The copyright sign appears later at position $D9, which is assigned to the Latin capital letter “Y” diaeresis in the Standard Roman character set.

Table A-4 Croatian variations from the Standard Roman character set

Glyph	Unicode name	PostScript name	Hexadecimal	Decimal
S	LATIN CAPITAL LETTER S HACEK	Scaron	$A9	169
Z	LATIN CAPITAL LETTER Z HACEK	Zcaron	$AE	174
∆	INCREMENT	Delta	$B4	180
s	LATIN SMALL LETTER S HACEK	scaron	$B9	185
z	LATIN SMALL LETTER Z HACEK	zcaron	$BE	190

Table A-4 Croatian variations from the Standard Roman character set (continued)

Glyph	Unicode name	PostScript name	Hexadecimal	Decimal
B	LATIN CAPITAL LETTER C ACUTE	Cacute	$C6	198
C	LATIN CAPITAL LETTER C HACEK	Ccaron	$C8	200
D	LATIN CAPITAL LETTER D BAR	Dmacron	$D0	208
	APPLE LOGO	apple	$D8	216
©	COPYRIGHT SIGN	copyright	$D9	217
Æ	LATIN CAPITAL LETTER A E	AE	$DE	222
»	RIGHT POINTING GUILLEMET	guillemotright	$DF	223
–	EN DASH	endash	$E0	224
b	LATIN SMALL LETTER C ACUTE	cacute	$E6	230
c	LATIN SMALL LETTER C HACEK	ccaron	$E8	232
e	LATIN SMALL LETTER D BAR	dmacron	$F0	240
π	GREEK SMALL LETTER PI	pi	$F9	249
Ë	LATIN CAPITAL LETTER E DIAERESIS	Edieresis	$FA	250
Ê	LATIN CAPITAL LETTER E CIRCUMFLEX	Ecircumflex	$FD	253
æ	LATIN SMALL LETTER A E	ae	$FE	254

Table A-5 shows the glyph assignments in the Romanian version of the Roman character set that diverge from the Standard Roman character set, their Unicode and PostScript names, and their Macintosh character codes in hexadecimal and decimal.

Table A-5 Romanian variations from the Standard Roman character set

Glyph	Unicode name	PostScript name	Hexadecimal	Decimal
A	LATIN CAPITAL LETTER A BREVE	Abreve	$AE	174
R	LATIN CAPITAL LETTER S CEDILLA (COMMA VARIANT)	Scedilla	$AF	175
a	LATIN SMALL LETTER A BREVE	abreve	$BE	190
r	LATIN SMALL LETTER S CEDILLA (COMMA VARIANT)	scedilla	$BF	191
T	LATIN CAPITAL LETTER T CEDILLA (COMMA VARIANT)	Tcedilla	$DE	222
t	LATIN SMALL LETTER T CEDILLA (COMMA VARIANT)	tcedilla	$DF	223

Table A-6 shows the glyph assignments in the Turkish version of the Roman character set that diverge from the Standard Roman character set, their Unicode and PostScript names, and their Macintosh character codes in hexadecimal and decimal.

Table A-6 Turkish variations from the Standard Roman character set

Glyph	Unicode name	PostScript name	Hexadecimal	Decimal
G	LATIN CAPITAL LETTER G BREVE	Gbreve	$DA	218
g	LATIN SMALL LETTER G BREVE	gbreve	$DB	219
I	LATIN CAPITAL LETTER I DOT	Idot	$DC	220
ı	LATIN SMALL LETTER DOTLESS I	dotlessi	$DD	221
Q	LATIN CAPITAL LETTER S CEDILLA	Scedilla	$DE	222
q	LATIN SMALL LETTER S CEDILLA	scedilla	$DF	223

Table A-7 shows the glyph assignments in the Icelandic and Faroese versions of the Roman character set that diverge from the Standard Roman character set, their Unicode and PostScript names, and their Macintosh character codes in hexadecimal and decimal.

Table A-7 Icelandic and Faroese variations from the Standard Roman character set

Glyph	Unicode name	PostScript name	Hexadecimal	Decimal
Y	LATIN CAPITAL LETTER Y ACUTE	Yacute	$A0	160
D	LATIN CAPITAL LETTER ETH	Eth	$DC	220
d	LATIN SMALL LETTER ETH	eth	$DD	221
P	LATIN CAPITAL LETTER THORN	Thorn	$DE	222
p	LATIN SMALL LETTER THORN	thorn	$DF	223
y	LATIN SMALL LETTER Y ACUTE	yacute	$E0	224

In addition to regional variations in the character set, the Roman script system in particular contains many fonts with unique glyphs. Since the character encoding is limited to 256 values, specialized fonts such as Symbol and ITC Zapf Dingbats override the Standard Roman character encoding. For example, in the Standard Roman character set $70 corresponds to lowercase “p”, but it is the numeric symbol for pi (π) in the Symbol font, an outlined square (❒) in Zapf Dingbats, and the musical symbol pianissimo for *play quietly* in the Sonata font. Hence, there is no guarantee that a Roman character code will always represent the same character in every font.

The U.S. Keyboard-Layout ('KCHR') Resource

The U.S. system software keyboard-layout resource (resource type `'KCHR'`, ID = 0) is included with every Macintosh system. It implements the Standard Roman character set shown in Figure A-1 on page A-5. The structure of the keyboard-layout resource is documented in the appendix "Keyboard Resources" in this book. This section describes in general how the U.S. `'KCHR'` resource handles key combinations and dead keys.

The U.S. system software's keyboard-layout resource makes it possible to enter accented forms in the Standard Roman character set with **dead keys,** designated keys or modifier-plus-key combinations that produce no immediate effect when pressed but instead affect the character or characters produced by the next keys typed, called **completer keys.** Users can enter all the accented forms in the Standard Roman character set with dead keys. For example, pressing Option-E on the U.S. keyboard produces nothing (no event is posted), but subsequently typing "e" produces "é".

Note
Other keyboard layouts may produce accented characters in other ways. On the French keyboard (`'KCHR'` resource ID = 1), for example, pressing Option-E directly produces "é". ◆

The U.S. keyboard-layout resource provides the following key-combination features for consistency:

- Because the Shift key is ignored if the Command key is pressed, the Caps Lock key is also ignored if the Command key is pressed.
- Handling of the Option-Shift and Option–Caps Lock key combinations is based on the following principles:
  - If either the Option or the Option-Shift key combination produces a letter, then the Option–Caps Lock key combination produces the same character as the Option, not the Option-Shift, key combination.
  - If the Option key combination is a dead key for a particular accent, then the Option-Shift key combination produces the accent directly.

A **no-match character** (also called a *default completion character*) is the character that is produced when the keystroke following a dead key is either a space or a key for a character that cannot take the accent corresponding to the dead key. In system software versions 7.0 and later, default completion characters are "real" accent characters instead of low-ASCII approximations.

Standard Sorting Routines

The standard Macintosh sorting routines are contained in the Pack 6 resource, a system code resource (type = `'PACK'`, ID = 6) initialized at startup. As they process each character or sorting unit, the standard routines first call equivalent routines in the current script system's string-manipulation (`'itl2'`) resource. Those routines, called **sorting hooks,** are described with the string-manipulation resource in the appendix "International Resources" in this book.

The U.S. string-manipulation (`'itl2'`) resource contains only empty sorting hooks. Other localized versions of the Roman script system—and non-Roman script systems—provide their own string-manipulation resources that may have nonempty routines to modify or replace any of the standard routines, on a character-by-character basis.

Table A-8 describes the sorting behavior implemented by the standard Macintosh sorting routines. All characters of the Standard Roman character set are sorted. Primary sorting is shown in vertical order; secondary sorting is horizontal. This is the default sorting behavior used by the Text Utilities, and is appropriate for U.S. and similar localized versions of the Roman script system. All Text Utilities sorting routines (other than `RelString` and `EqualString`) use the sorting behavior specified in the string-manipulation resource, which may or may not be identical to the standard behavior. (`RelString` and `EqualString` use an invariant sorting behavior that is described in the chapter "Text Utilities" in this book.)

Table A-8 Standard sorting order (for Standard Roman character set)

Primary	Secondary
$00	NUL
…	
$1F	US
$20	space ($20) non-breaking space ($CA)
$21	!
$22	" « ($C7) » ($C8) “ ($D2) ” ($D3)
$23	#
$24	$
$25	%
$26	&

Table A-8 Standard sorting order (for Standard Roman character set) (continued)

Primary	Secondary
$27	' ‘ ($D4) ’ ($D5)
$28	(
…	
$40	@
$41	A Á À Â Ä Ã Å a á à â ä ã å
$42	B b
$43	C Ç c ç
$44	D d
$45	E É È Ê Ë e é è ê ë
…	
$49	I Í Ì Î Ï i ı í ì î ï
…	
$4E	N Ñ n ñ
$4F	O Ó Ò Ô Ö Õ Ø o ó ò ô ö õ ø
…	
$55	U Ú Ù Û Ü u ú ù û ü
…	
$59	Y y ÿ Ÿ
$5A	Z z
$5B	[
…	
$60	`
$7B	{
…	
$7F	DEL
$A0	†
…	

Low-ASCII characters (other than letters) not listed in Table A-8 have primary sorting only and are sorted according to numeric code. Low-ASCII letters not listed in Table A-8 have a primary sorting order that is alphabetical and a secondary sorting order of uppercase followed by lowercase (like "B b"). All characters with codes above $A0 that are not listed in Table A-8 are sorted after $A0 according to numeric character code (except for ligatures; see note on sorting of ligatures).

Note the following details and anomalies in the standard Macintosh sorting order:

- The symbols ª ($BB) and º ($BC) are explicitly treated as symbols, not as letters, and their primary sorting positions are not respectively the same as A and O.
- The en-dash ($D0) and em-dash ($D1) do not have the same primary sorting position as hyphen-minus ($2D).
- The double low quotation mark „ ($E3) does not have the same primary sorting position as quotation " ($22).
- The single low quotation mark ‚ ($E2), and the left and right single guillemet ‹ › ($DC, $DD) do not have the same primary sorting position as apostrophe ' ($27).
- The secondary sorting position for dotless-lower-i ı ($F5) is indeterminate. It sorts at exactly the same place (primary and secondary order) as regular lower i ($69).
- The character Ÿ does not sort between Y and y.

Sorting of ligatures

For a ligature, the primary sorting position is equivalent to the separate characters that make up the ligature. The secondary sorting position is just following the equivalent separate characters. Ligatures are sorted by the following first and second characters:

Ligature	First character	Second character
Æ	A	E
æ	a	e
fi	f	i
fl	f	l
Œ	O	E
œ	o	e
ß	s	s

Diacritical Stripping and Case Conversion

The Text Utilities routines `LowercaseText`, `UppercaseText`, `StripDiacritics`, and `UppercaseStripDiacritics` use information in a script system's string-manipulation (`'itl2'`) resource to perform their tasks. A Roman string-manipulation resource is included with every Macintosh system; the U.S. version of the Roman string-manipulation resource converts case and strips diacritical marks according to the following rules:

- The unaccented letters A–Z and a–z are converted to unaccented a–z and A–Z, respectively, by case conversion. They are unaffected by stripping.
- Accented versions (Å, ê, Ñ) are converted to equivalent unaccented versions (A, e, N) by stripping.
- Accented versions (Å, ê, Ñ) are converted to identically accented case-changed versions (å, Ê, ñ) by case conversion.
- Ligatures are unaffected by stripping, but are converted as appropriate by case conversion. Only the ligatures ß, fi, and fl, which have no uppercase versions, are unaffected by case conversion as well as by stripping.

U.S. International Resources and Keyboard Resources

When Macintosh system software is localized for a non-U.S. market, its system script may be Roman or non-Roman. In either case, it contains replacements for or modifications to the U.S. international resources and keyboard resources. Any U.S. Roman resource that is not replaced is included with the localized system.

Table A-9 shows which international resources are included in the U.S. system software, and how localized versions of the system software (and secondary scripts) add resources or replace them in the System file. Note that all non-Roman script systems and most localized versions of system software include localized versions of the `'itl0'`, `'itl1'`, `'itl2'`, and `'itl4'` resources. (Not all non-U.S. Roman script systems add a non-U.S. `'itl4'`.) Some non-Roman systems may also use optional `'itl5'` or `'trsl'` resources.

Table A-9 International resources in U.S. system software

Resource type	U.S. system software including Roman script system	Localized versions of system software or other script systems
`'itlc'`	Roman `'itlc'`	May replace `'itlc'`
`'itlm'`	Default `'itlm'`	May replace `'itlm'`
`'itlb'`	Roman `'itlb'`	May add non-Roman `'itlb'`

continued

Table A-9 International resources in U.S. system software (continued)

Resource type	U.S. system software including Roman script system	Localized versions of system software or other script systems
'itl0'	U.S. 'itl0'	Adds non-U.S. 'itl0'
'itl1'	U.S. 'itl1'	Adds non-U.S. 'itl1'
'itl2'	U.S. 'itl2'	Adds non-U.S. 'itl2'
'itl4'	U.S. 'itl4'	May add non-U.S. 'itl4'
'itl5'	(none)	May add non-Roman 'itl5'
'trsl'	(none)	May add non-Roman 'trsl'

Table A-10 lists the keyboard resources that are included in the U.S. system software and shows how localized versions of the system software (and auxiliary scripts) add resources or replace them in the System file. Note that all localized versions of system software and all non-Roman script systems include at least one keyboard-layout resource and keyboard icon family; some provide a key-remap resource; and none need provide a key-map resource or key-caps resource.

Table A-10 Keyboard resources in U.S. system software

Resource type	U.S. system software including Roman script system	Localized versions of system software or other script systems
'itlk'	(none)	May add an 'itlk'*
'KCHR'	U.S. 'KCHR'	Adds non-U.S. 'KCHR'
'KSWP'	Standard 'KSWP'	May replace 'KSWP'
'KMAP'	All necessary 'KMAP's	(none)
'kcs#'	U.S. 'kcs#'	Adds non-U.S. 'kcs#'*
'kcs4'	U.S. 'kcs4'	Adds non-U.S. 'kcs4'*
'kcs8'	U.S. 'kcs8'	Adds non-U.S. 'kcs8'*
'KCAP'	All necessary 'KCAP's	(none)

* ID number equal to corresponding 'KCHR' ID number

For more information on these resources, see the appendixes "International Resources" and "Keyboard Resources" in this book.

WorldScript I

WorldScript I is a system extension, available with system software version 7.1 and later, that can support all types of 1-byte complex script systems. It contains code that implements many script-aware text-manipulation routines, eliminating the need for each script to maintain its own code extensions.

WorldScript I is a single file located in the Extensions folder within the System Folder on the user's Macintosh computer. It installs and initializes all compatible script systems present in the System Folder, and provides each with a set of standard routines. Script systems compatible with WorldScript I are called *universal scripts*, because they make use of the universally applicable WorldScript I routines.

About WorldScript I

Script systems developed prior to system software version 7.1 contain their own code to handle language-specific text processing. Each script system also has its own initialization and configuration code, installing itself at startup and adding its own modifications to the system. This process can result in a layering of patches to the same traps, inconsistent behavior, and inefficient use of memory.

WorldScript I redefines what a script system consists of by combining the executable code for many routines for all 1-byte script systems. It includes initialization and formatting routines that support all contextual forms required by all 1-byte scripts; script-specific behavior is encoded in resource-based tables. This approach reduces memory requirements for multiscript systems and avoids layering of patches.

WorldScript I routes script-aware calls through each universal script's own *dispatch table;* the dispatch table initially points back to the *script utility* routines in WorldScript I. This indirection allows your application to add to or replace existing routines on a script-by-script basis. Script Manager calls allow you to modify or add to any script's utility routine or patch. You can even replace an individual script system's routine completely if you need features significantly different from those provided by WorldScript I. Script utilities and dispatch tables are described in the next section and under "Flexible Dispatching Method" beginning on page A-28.

Shared Script Utilities and QuickDraw Patches

The **script utilities** are the low-level routines through which an individual script system implements script-aware Text Utilities, QuickDraw, and Script Manager routines. When an application makes a script-aware call, the script management system converts it to a script utility call and passes it on to the appropriate script system. Previous to system software version 7.1, individual script systems provided their own script utilities. With WorldScript I, a single set of script utilities can work with all 1-byte complex scripts.

Two Script Manager routines, `GetScriptUtilityAddress` and `SetScriptUtilityAddress`, allow you to access and override a script's utility routines.

Table A-11 lists the script utilities implemented by WorldScript I, along with the chapters in this book that describe their corresponding high-level routines.

Table A-11 Script utilities supported by WorldScript I

Script utility	Chapter in this book
`CharacterByteType`	Script Manager
`CharacterType`	Script Manager
`CharToPixel`	QuickDraw Text
`DrawJustified`	QuickDraw Text
`FillParseTable`	Script Manager
`FindScriptRun`*	Text Utilities
`FindWordBreaks`*	Text Utilities
`GetScriptQDPatchAddress`	Script Manager
`GetScriptUtilityAddress`	Script Manager
`GetScriptVariable`†	Script Manager
`HiliteText`	QuickDraw Text
`MeasureJustified`	QuickDraw Text
`PixelToChar`	QuickDraw Text
`PortionLine`	QuickDraw Text
`SetScriptQDPatchAddress`	Script Manager
`SetScriptUtilityAddress`	Script Manager
`SetScriptVariable`	Script Manager
`TransliterateText`	Script Manager
`VisibleLength`	QuickDraw Text

NOTE WorldScript I supports the following script utilities for backward compatibility. They call newer versions of themselves to handle their tasks. They are: `Pixel2Char` (calls `PixelToChar`), `Char2Pixel` (calls `CharToPixel`), `DrawJust` (calls `DrawJustified`), `MeasureJust` (calls `MeasureJustified`), `PortionText` (calls `PortionLine`), `CharByte` (calls `CharacterByteType`), `CharType` (calls `CharacterType`), `ParseTable` (calls `FillParseTable`), `Transliterate` (calls `TransliterateText`).

* The Script Manager handles these routines directly if the necessary tables are in the script's `'itl2'` resource. Otherwise, they are passed to WorldScript I.

† The Script Manager handles these routines directly if the standard selectors documented in this book are used. The routines are passed to WorldScript I if private selectors are used.

WorldScript I also patches four standard QuickDraw text-handling routines: `StdText`, `StdTxMeas`, `MeasureText`, and `FontMetrics`. (`FontMetrics` is a Font Manager routine, but for simplicity all four routines and patches are referred to in this appendix as QuickDraw routines and patches.) The purpose of the QuickDraw patches is to lay out text according to context and line direction. The original QuickDraw routine is called after the text is laid out properly. The QuickDraw dispatch table has special entries to support developer patching of routines for printing as well as for display.

Two Script Manager routines, `GetScriptQDPatchAddress` and `SetScriptQDPatchAddress`, allow you to access and override a script's QuickDraw patches.

Table A-12 lists the QuickDraw patches implemented by WorldScript I, along with the chapters in this book that describe the original routines.

Table A-12 QuickDraw patches supported by WorldScript I

QuickDraw patch	Chapter in this book
`FontMetrics`	Font Manager
`MeasureText`	QuickDraw Text
`StdText`	QuickDraw Text
`StdTxMeas`	QuickDraw Text

Table-Based Script Behavior

The shared script utilities determine script-specific characteristics from the tables in each script system's international resources.

WorldScript I uses tables in a script system's string-manipulation (`'itl2'`) resource for analyzing character types, finding word breaks, and performing case conversion. This use of tables predates the existence of WorldScript I, but WorldScript I extends the use of tables to all routines for 1-byte complex script systems. New tables that are required are put in the script's encoding/rendering (`'itl5'`) resource, using a tagged-table index for storage and retrieval. For example, the contextual formatting routines (described in the next section) uses tables in the encoding/rendering resource.

The international resources are described in the appendix "International Resources" in this book.

Contextual Formatting Routines

WorldScript I uses a set of table-driven routines to format text according to each script's requirements and attributes. The WorldScript I script utilities and QuickDraw patches that perform text formatting and layout rely on these table-driven routines. Each script has several tables in its encoding/rendering (`'itl5'`) resource to specify the display characteristics of its text.

Flexible Dispatching Method

Each enabled script system has a **script record,** a private data structure that holds information and addresses pertinent to that script. When an application makes a script-aware call, the script management system determines the current script and consults that script's script record for the address of the script's **dispatch routine** (which is actually part of WorldScript I). It passes the call to the dispatch routine, which uses the script's **dispatch table** to execute the proper script utility. Every script system has its own pointer to the dispatch routine and its own dispatch table, separate from other scripts. When the application makes a script-aware QuickDraw call, WorldScript I uses the script's QuickDraw dispatch table to execute the proper QuickDraw patch.

At run time, the application call has been converted to a lower-level script utility call or QuickDraw call. Each script utility call includes a *script utility selector*, a number that the dispatch routine uses to select the proper routine from the dispatch table.

Initialization Sequence

The startup code for enabling all available 1-byte complex script systems and script utility routines is in WorldScript I. WorldScript I is located in the Extensions folder; its file type is `'scri'` and its creator is `'univ'`.

At startup, WorldScript I does the following:

1. It checks for a valid machine configuration. WorldScript I works on Macintosh Plus models and later; it requires Script Manager version 2.0 or later and system software version 7.1 or later.
2. WorldScript I gets the number of valid 1-byte script bundles in the System file. A valid script bundle consists of a set of international resources (`'itlb'` and `'itl5'` required, `'itl0'`, `'itl1'`, `'itl2'`, and `'itl4'` optional) and at least one font in the script system's ID range. The `smsfSingByte` bit in the international bundle (`'itlb'`) resource must also be set to indicate that the script is 1-byte.

   If no 1-byte script bundles are present, WorldScript I does not load any of its script utilities or QuickDraw patches. It exits without signaling an error.
3. If one or more valid script bundles are present, WorldScript I does the following for each script:
   1. It checks for enough memory to load the script.
   2. It checks the `smsfUnivExt` bit in the script's international bundle resource. If the flag is set, the script system is a universal script system, and WorldScript I proceeds. If the flag is clear, WorldScript I goes on to the next script.
   3. It creates a script record and initializes the record with the script's values.
   4. If this is the first universal 1-byte script allocated, WorldScript I loads its script utilities and QuickDraw patches into the system heap.

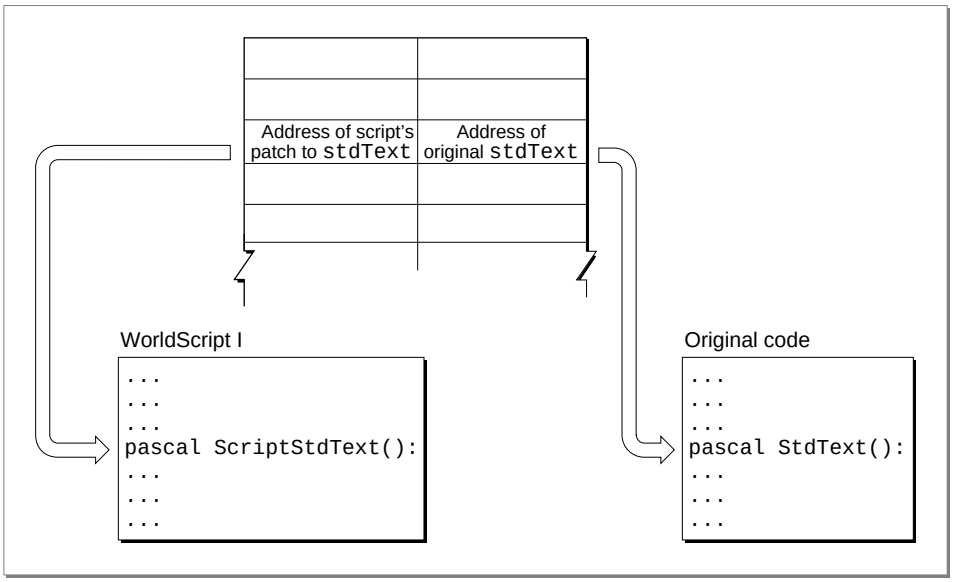
5. WorldScript I allocates the script's dispatch table and sets the table's elements to point to the WorldScript I script utilities and QuickDraw patches or to the original Roman script utilities and QuickDraw calls, as appropriate for the script.
6. WorldScript I makes default settings for the script system based on information from the configuration table in the script's encoding/rendering (`'itl5'`) resource. It then looks in the Preferences folder for a script preferences file. If one is found, and if the file contains a configuration resource (type `'CNFG'`) for this script system, WorldScript I uses that resource to reinitialize the script record's fields. If no preferences file is available, WorldScript I keeps the default settings loaded from the encoding/rendering resource.
7. WorldScript I initializes Script Manager data structures that point to this script record.

Any initialization errors that occur are reported to the user via the Notification Manager.

How Calls Are Dispatched

In every script system that is compatible with WorldScript I, the dispatch-table element for every script utility and QuickDraw patch consists of two pointers: one to the WorldScript I implementation of the routine and one to the original routine. In all cases, the WorldScript I routine executes first. In some cases, WorldScript I calls the original routine after its own; in other cases, the pointer to the original routine is `NIL` and the WorldScript I routine is all that executes. See Figure A-2. This design allows you to place a patched routine so that it executes either before, in place of, or after the WorldScript I routine and allows you to either call the original routine or not call it.

Figure A-2 Dispatch table entry for script utilities and QuickDraw patches

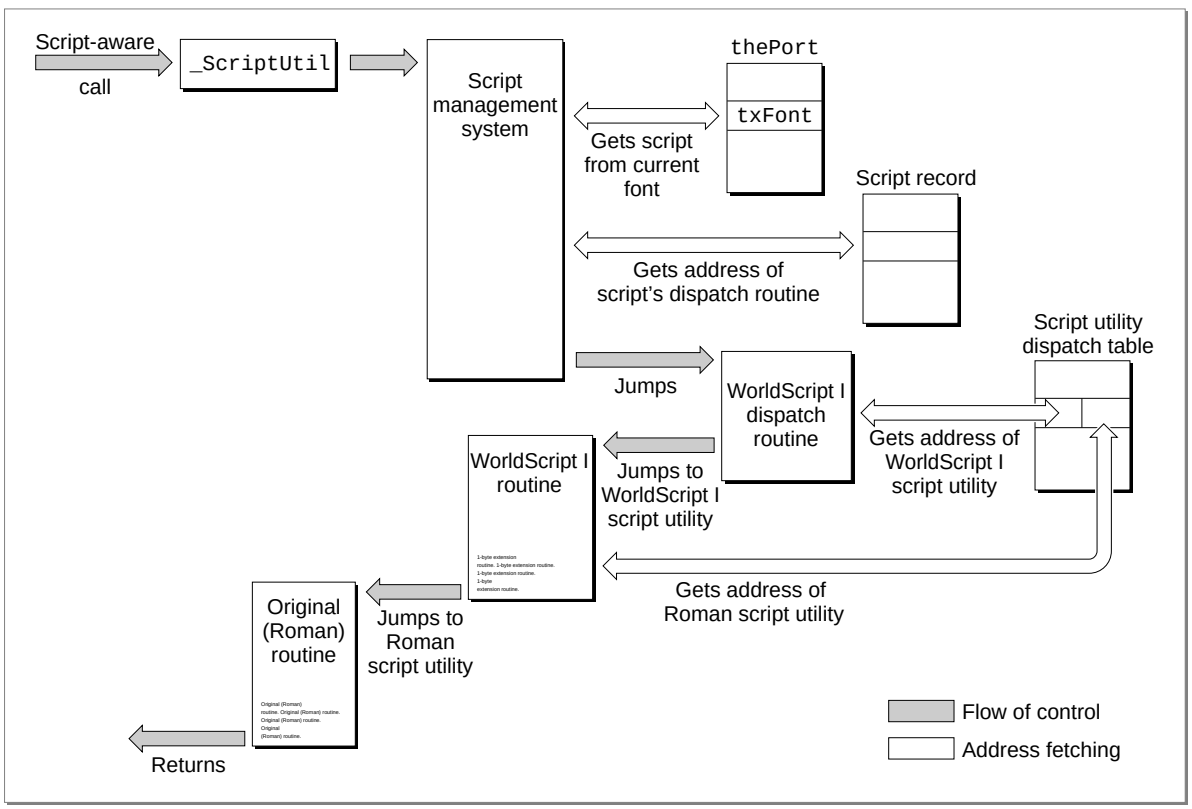


Every script-aware call to system software that executes as a script utility goes through the `_ScriptUtil` trap ($A8B5). The script management system handles some of those calls, such as `GetSMVariable`, itself; other calls, such as `DrawJustified`, it passes to a script system through the script system's script record. Those calls are listed in Table A-11 on page A-26. The script system uses its script utility dispatch table to call the right script utility. See Figure A-3.

When it receives a script utility call, a script's dispatch routine does the following:

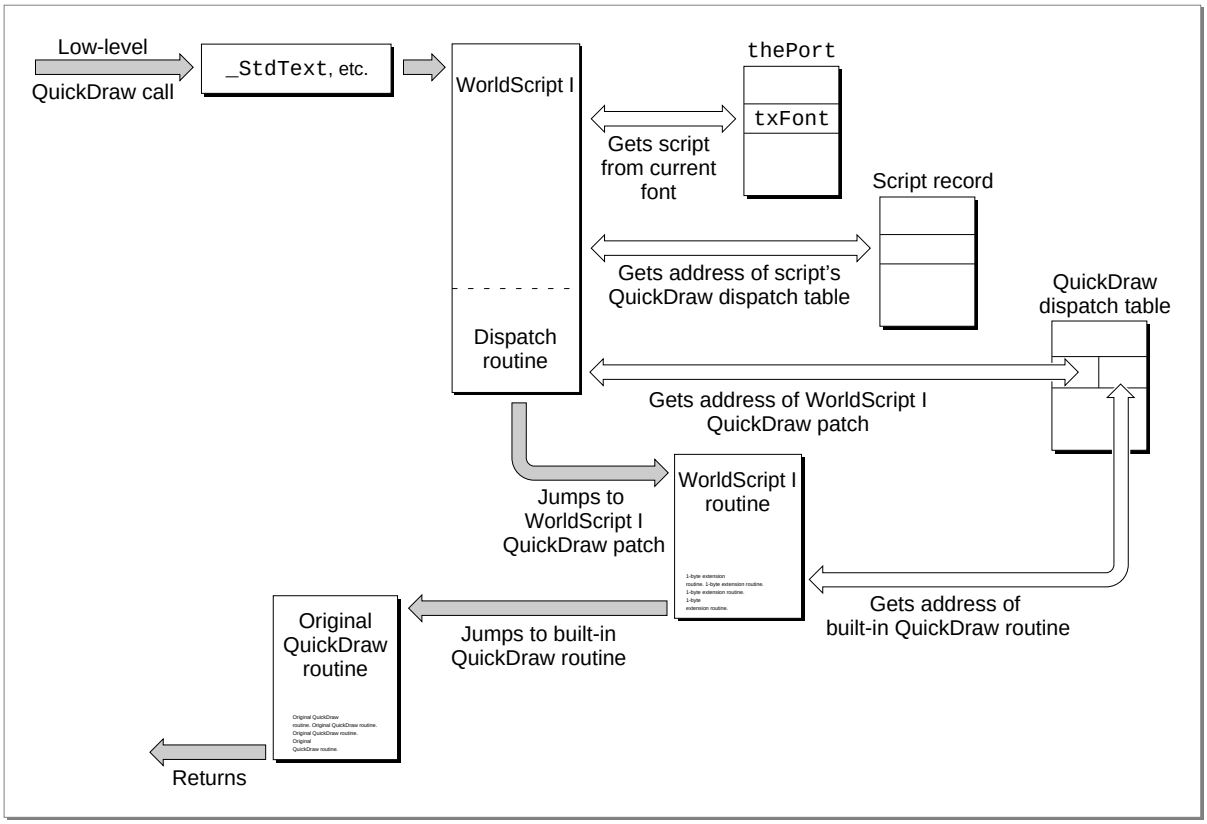
1. It checks to see if the call (as defined by the script utility selector) is within the range of routines handled by the script.
2. It gets the address of the script utility from the script's dispatch table, using the script utility selector.
3. It replaces the selector on the stack with the address of its own script record.
4. It jumps into the WorldScript I routine obtained in step 2. As the routine executes, it obtains script-specific characteristics from the script record passed to it in step 3.
5. The WorldScript I routine gets the address of the original (Roman) routine from the dispatch table and, if it is not `NIL`, jumps to that routine upon completion.

Figure A-3 How calls are dispatched to the 1-byte script utilities



A patched low-level QuickDraw call follows a similar path, except that it goes through a QuickDraw trap that has been patched to execute code in WorldScript I instead of passing through the script management system. After determining which script should handle this call (by examining the current font), WorldScript I uses the script's QuickDraw dispatch table to jump to the proper routine. See Figure A-4.

Figure A-4 How calls are dispatched to the 1-byte QuickDraw patches



Saving User Preferences

The Operating System and some individual script systems use control panels to let users change text-related system settings, such as line direction, associated font, kashida preferences, and caret style. Script systems may store these settings in a script preferences file in the Extensions folder. This file contains resources of type `'CNFG'`, whose resource ID numbers are equal to the script codes of the script systems they represent.

As noted under "Initialization Sequence" on page A-28, when installing a script system WorldScript I looks for a script preferences file in the Preferences folder. For each script system it initializes, WorldScript I loads the `'CNFG'` resource for that script from the script preferences file and uses it to reconfigure the script.

Note
The `'CNFG'` resource in the script preferences file has exactly the same format as the script configuration table in the encoding/rendering resource. However, it may not have exactly the same elements. The types of configuration settings specified in the `'itl5'` resource may be different from those settable by the user through a control panel. ◆

See the discussion of the encoding/rendering resource in the appendix "International Resources" in this book for a description of the script configuration table.

Replacing a Script Utility or QuickDraw Patch

Developers of 1-byte complex script systems should be able to specify most of their script system's behavior in tables in the script system's encoding/rendering (`'itl5'`) resource. In cases where the WorldScript I routines are insufficient to handle the script's specific needs, the developers may create patches and install them as described here. The patches may be installed by an extensions file that is executed at system startup.

Application developers who need specific script-based behavior for their programs should not alter the tables in a 1-byte script's encoding/rendering (`'itl5'`) resource. However, they can replace one or more 1-byte script utilities or QuickDraw patches for their target script system, as described here.

IMPORTANT
When you patch a script system's script utility or QuickDraw call, you alter that script's behavior for as long as it remains enabled. Therefore, be sure to restore the patches to their original state whenever your application quits or is switched out by the Process Manager. ▲

The script-based dispatch table design gives you a simple, flexible way to replace individual routines without having to patch out all of the `_ScriptUtil` trap or any of the QuickDraw low-level routines in their entirety. Furthermore, in a multiscript environment each patch of this type applies only to its own script system. A developer might, for example, patch `StdText` for the Thai script system only, leaving all the other scripts unchanged.

In addition, you can choose the point at which your patched routine executes: either before (which also means *in place of* if your routine does not call the WorldScript I routine at all) or after the WorldScript I routine executes. For example, the WorldScript I version of `StdText` works by first performing contextual analysis and reordering of characters on the supplied text, and then calling the original version of `StdText`. Suppose you want to keep the WorldScript I contextual analysis and reordering of characters, but you want to do some additional processing before calling the original `StdText`. To do that, just patch out the WorldScript I routine's call to the original `StdText`, instead of patching out the entire WorldScript I routine. Then do your own processing and call `StdText` yourself.

Because you can patch at two points, and because you can perform your own processing either before or after a patch, your flexibility is great. To replace only the WorldScript I routine, replace its pointer in the dispatch table; to keep the WorldScript I routine while replacing or patching the original routine, replace the original-routine pointer in the dispatch table. The four Script Manager routines that allow you to make those patches are `GetScriptUtilityAddress`, `SetScriptUtilityAddress`, `GetScriptQDPatchAddress`, and `SetScriptQDPatchAddress`. Either pointer in the dispatch table may be `NIL`, meaning that WorldScript I either doesn't patch the original routine or doesn't call the original routine.

Patching Script Utilities

In terms of how to patch them, the script utilities can be divided into different groups, depending on whether or not WorldScript I performs contextual formatting and whether or not it subsequently calls the original Roman version of the utility. See Table A-13. For utilities that perform contextual formatting, keep in mind that if you replace them you will have to handle formatting yourself. For utilities that subsequently call their original Roman version, you can replace either the WorldScript I version of the call or the Roman version or both, depending on what your needs are.

Table A-13 Classification of 1-byte script utilities by function

No formatting, do not call original Roman routine	No formatting, do call original Roman routine	May do formatting, do not call original Roman routine
`CharacterByteType`	`CharacterType`	`HiliteText`
`GetScriptVariable`	`TransliterateText`	`VisibleLength`
`SetScriptVariable`	`FindWordBreaks`	`PixelToChar`
`FillParseTable`	`FindScriptRun`	`CharToPixel`
`PortionLine`		`DrawJustified`
		`MeasureJustified`

Note that those script utilities that do not call their equivalent Roman routine nevertheless call QuickDraw `StdText` or `StdTxMeas` *if* the `grafProcs` field in the graphics port has been changed. Thus, if you have changed (patched) either of those QuickDraw routines, your patch will still be called. Conversely, if the `grafProcs` field is `NIL`, The WorldScript I script utilities do not necessarily call `StdText` or `StdTxMeas`.

If you are replacing a script utility, remember that its interface is similar to that of its equivalent high-level call as described in *Inside Macintosh*. The utility takes the same parameters in the same order, except that it gets one additional last parameter on the stack: a pointer to the script record. For example, if you are replacing `VisibleLength`, whose high-level interface is

```
FUNCTION VisibleLength (textPtr: Ptr;
                        textLen: LongInt): LongInt;
```

your patch to the equivalent script utility should expect to receive parameters as if the high-level interface were

```
FUNCTION VisibleLength (textPtr: Ptr;
                        textLen: LongInt;
                        scriptRecord: Ptr): LongInt;
```

Also, if your replacement calls the original routine, don't forget to pass the extra parameter to it.

Patching QuickDraw Routines

WorldScript I patches the low-level QuickDraw text-handling routines `StdText` and `StdTxMeas`, the high-level QuickDraw routine `MeasureText`, and the Font Manager routine `FontMetrics`.

The QuickDraw patches lay out lines of text according to the context and line-direction rules for a script system. In each case (except for `MeasureText`) the patch calls the original QuickDraw routine after performing the contextual formatting. The contextual formatting routines are called only for contextual scripts.

Table A-14 lists the patches and what they do. For those patches that perform contextual formatting, if you replace them you will have to handle formatting for line layout yourself. For all of them, you can replace either the WorldScript I patch or the standard QuickDraw call or both, depending on your needs.

Table A-14 Classification of 1-byte QuickDraw patches by function

Call	Function
`FontMetrics`	Returns font measurements
`MeasureText`	Calls `MeasureJustified` (with slop = 0)
`StdText`	Does formatting, then calls original routine
`StdTxMeas`	Does formatting, then calls original routine

Issues in Designing a Script Utility or QuickDraw Patch

Keep the following points in mind if you plan to replace one or more script utilities or QuickDraw patches in WorldScript I:

- In script systems compatible with WorldScript I, text handling typically involves WorldScript I contextual analysis followed by a call to the original Roman version of the routine. You need to know whether it is the WorldScript I functionality or the original functionality that you want your routine to replace, and you need to be sure that your routine is called only at the correct points in the process. More detailed information on text layout is found in the chapter "QuickDraw Text" in this book.
- Script utilities process text in individual style runs, whose boundaries are defined by the application. If your application supports styled text, each script utility will need to handle only individual style runs. But if your application supports unstyled text only, there may be mixed Roman and non-Roman characters in a single font. Before performing text layout, your script utility will have to separate the Roman characters into their own style runs, and assign them to an associated font, if your script system uses associated fonts.
- If you provide your own script utility, you need to be sure that text is not formatted more than once. Because script utilities might be called reentrantly during printing, you may want to save the port for each contextual analysis. Check this port against the current port for each possible contextual analysis request, so you can prevent the text from being formatted twice.
- Printing adds another level of complexity to the WorldScript I QuickDraw patches and your ability to patch out those patches. The QuickDraw dispatch table has special entries to support developer patching of routines in printing as well as for display. See the descriptions of the routines `GetScriptQDPatchAddress` and `SetScriptQDPatchAddress` in the chapter "Script Manager" in this book for more information. See also Macintosh Technical Note #174 for additional information on printing.

WorldScript II

WorldScript II is a system extension, available with system software versions 7.1 and later, that can support 2-byte script systems: Chinese (traditional and simplified characters), Japanese, and Korean. It contains code that implements script-aware text-manipulation routines, eliminating the need for each script to maintain its own code extensions. WorldScript II supports the input, display, and printing of the thousands of characters needed by the 2-byte script systems.

WorldScript II is a single file located in the Extensions folder within the System Folder on the user's Macintosh. It installs all compatible 2-byte script systems that are present in the System Folder and provides each with a set of standard routines.

Note
Unlike WorldScript I, WorldScript II does not support the Script Manager routines (such as `SetScriptUtilityAddress`) that allow replacement of script utilities or QuickDraw calls. ◆

About WorldScript II

The 2-byte script systems developed prior to system software version 7.1 contain their own code to handle language-specific text processing. Each also has its own initialization and configuration code, installing itself at startup and adding its own modifications to the system. Each script system patches three different areas of system software: QuickDraw, the Event Manager, and the script management system. This can result in a layering of patches to the same traps, inconsistent behavior, and inefficient use of memory.

WorldScript II, working with the Text Services Manager and other parts of system software, eliminates code duplication and provides for the special text-input needs of the 2-byte systems:

- Enhancements to QuickDraw and the Font Manager now support the display of the thousands of Chinese, Korean, and Japanese characters. To handle a character set that is larger than the 256-character ASCII range, the Font Manager and other parts of system software contain the code necessary to retrieve and render the characters.
- The Text Services Manager, using enhancements to the Event Manager, provides broad support for input methods. Input methods that employ the Text Services Manager intercept every key-down event, map the event to a character code, and pass the result to the application.
- WorldScript II provides language-specific capabilities for script-aware text-handling routines called **script utilities.** For instance, WorldScript II provides routines that tell whether a byte in a string is a 1-byte or 2-byte character.

WorldScript II redefines what a 2-byte script system consists of. WorldScript II combines the executable code for many routines for all 2-byte script systems. Script-specific

behavior is encoded in resource-based tables. This reduces memory requirements for multiscript systems and avoids layering of patches.

Shared Script Utilities

WorldScript II contains the code for all script utilities. Script-specific behavior is determined by tables in each script's international resources. In a multiscript environment, WorldScript II loads only one copy of its code into memory. Furthermore, the user needs only the WorldScript II file in the Extensions folder, rather than one extension file per script system. This eases memory requirements and saves disk space.

Table A-11 lists the script utilities implemented by WorldScript II, along with the chapters in this book that describe their corresponding high-level routines.

Table A-15 Script utilities supported by WorldScript II

Script utility	Chapter in this book
`CharacterByteType`	Script Manager
`CharacterType`	Script Manager
`CharToPixel`	QuickDraw Text
`DrawJustified`	QuickDraw Text
`FillParseTable`	Script Manager
`FindScriptRun`*	Text Utilities
`FindWordBreaks`*	Text Utilities
`GetScriptVariable`†	Script Manager
`HiliteText`	QuickDraw Text
`MeasureJustified`	QuickDraw Text
`PixelToChar`	QuickDraw Text
`PortionLine`	QuickDraw Text
`SetScriptVariable`†	Script Manager
`TransliterateText`	Script Manager
`VisibleLength`	QuickDraw Text

NOTE WorldScript II supports the following script utilities for backward compatibility. They call newer versions of themselves to handle their tasks. They are: `Pixel2Char` (calls `PixelToChar`), `Char2Pixel` (calls `CharToPixel`), `DrawJust` (calls `DrawJustified`), `MeasureJust` (calls `MeasureJustified`), `PortionText` (calls `PortionLine`), `CharByte` (calls `CharacterByteType`), `CharType` (calls `CharacterType`), `ParseTable` (calls `FillParseTable`), `Transliterate` (calls `TransliterateText`).

* The Script Manager handles these routines directly if the necessary tables are in the script's 'itl2' resource. Otherwise, they are passed to WorldScript II.

† The Script Manager handles these routines directly if the standard selectors documented in this book are used. The routines are passed to WorldScript II if private selectors are used.

Table-Based Script Behavior

Script-specific text behavior is controlled by tables in each script system's international resources. The encoding/rendering resource (type `'itl5'`) contains character encoding information, and the transliteration resource (type `'trsl'`) contains information for character conversion among subscripts of a 2-byte script.

For example, the byte-type table in a script's encoding/rendering resource typically contains information about the type of a specific byte in the range of $00–$FF—whether it can be the high-order byte of a 2-byte character, the low-order byte of a 2-byte character, or a 1-byte character. The character-type table in the same resource gives more detailed information about a character in a particular coding scheme.

Currently, there are two transliteration formats used by WorldScript II and supported by tables in a script's transliteration resource. One of them is used to transliterate Jamo to Hangul (and Hangul to Jamo) in the Korean system. The other is a more general rule-based transliteration. You cannot customize the Jamo-to-Hangul transliteration. You can customize the rule-based transliteration by supplying the proper tables in a transliteration resource.

The encoding/rendering resource and the transliteration resource are described in the appendix "International Resources" in this book.

Initialization Sequence

The startup code for enabling all available 2-byte script systems and script utility routines is in WorldScript II. WorldScript II is located in the Extensions folder; its file type is `'scri'` and its creator is `'doub'`.

At system startup, all extension files in the Extensions folder are executed; script files are executed before all other extensions.

At startup, WorldScript II does the following:

1. It gets the total number of international bundle (`'itlb'`) resources from the System Folder.
2. For each bundle resource that belongs to a 2-byte script, WorldScript II
   - □ creates a script record and copies the byte-type table from the script's encoding/rendering (`'itl5'`) resource into the script record
   - □ gets handles to all transliteration resources for that script and adds them to the script record
   - □ initializes the script's script record, a private data structure that holds information and addresses pertinent to that script

How Calls Are Dispatched

WorldScript II does not implement all of the script utilities because some of them (such as `GetScriptManagerVariable`) are handled by the script management system itself. The script utilities that the script management system does not handle are listed in Table A-11 on page A-26. For those, the script management system passes execution to the WorldScript II dispatch routine, which in turn uses the script system's dispatch table to call the appropriate utility routine in WorldScript II.

When the WorldScript II dispatch routine calls a script utility, it replaces the selector of a normal script utility call with the address of the script record of the font script. It then calls the script utility. For instance:

1. In assembly language, when you call `CharacterByteType`, you typically call it through a macro in the following way:

```
subq.w   #2, sp            ; room for result.
   move.l   textPtr,-(sp)  ; push text pointer.
   clr.w    -(sp)          ; push text offset.
   CharacterByteType       ; find out whether it is 1-byte or
                           ; 2-byte character.
   tst.w    (sp)+          ; is it 1-byte?
```

2. The `CharacterByteType` macro expands into

```
move.l   $82060010,-(sp)   ; push the selector.
ScriptUtil                 ; call Script Manager trap.
```

3. For this example, the stack looks like the following when the trap has been called:

return address	(long) <-- top of stack
routine selector	(long)
text offset	(word)
text pointer	(long)
result	(word)

4. If the Script Manager does not handle the call, it passes the call to the current script's dispatch routine. The dispatch routine figures out by the value of the routine selector whether the call is in the range of calls it handles. If it is not, the dispatch routine strips the stack and returns without doing anything.
5. If the call is in the valid range, the dispatch routine performs these tasks:
   1. It gets the address of the WorldScript II version of the script utility from the script's dispatch table.
   2. It gets the address of the script record and replaces the selector on the stack with the address of the script record.
   3. It jumps into the routine.

6. So the stack becomes

return address	(long) <-- top of stack
address of script record	(long)
text offset	(word)
text pointer	(long)
result	(word)

The script's dispatch routine passes the script record to the WorldScript II script utilities so that they can use the script-specific information (such as the byte-type tables) in the script's international resources.