Script Manager Reference

(Not Recommended)

Carbon > Text & Fonts

2007-12-11

Contents

Script Manager Reference (Not Recommended)

Framework:	CoreServices/CoreServices.h
Declared in	Script.h

Overview

Important: The Script Manager is deprecated as of Mac OS X v10.5. Instead, you should update your application to handle Unicode text using the facilities of the Cocoa system (see *Text System Overview*) or Core Text (see *Core Text Programming Guide*). See also *Internationalization Programming Topics*.

The Script Manager makes script systems available and coordinates the interaction between many parts of the Mac OS and those available script systems. A script system (or script for short) is a collection of resources that provides for the representation of a particular writing system.

The Script Manager also provides several services directly to your application. Through them you can get information about the current text environment, modify that environment, and perform a variety of text-handling tasks.

The Script Manager has evolved through several versions. It started with sole responsibility for all international-compatibility and multilingual text issues, but as more power and features have been added, many of its specific functions have been moved to the other parts of system software.

For many text-related tasks, the Script Manager's role is transparent when you make a script-aware Text Utilities or QuickDraw call while processing text, that routine may get the information it needs through the Script Manager. For example, when you call the QuickDraw function `DrawText` to draw a line of text, `DrawText` in turn calls the Script Manager to determine which script system your text belongs to before drawing it. In other situations you may need to call the Script Manager explicitly, to properly interpret the text you are processing.

Carbon supports most Script Manager functions. However, Apple recommends that whenever possible you should replace Script Manager calls with the appropriate Unicode functionality. For more information, see Unicode Utilities Reference and Supporting Unicode Input.

See also the `KeyScript` function documentation.

Functions by Task

Analyzing Characters

`CharacterByteType` (page 95) Deprecated in Mac OS X v10.4
: Identifies a byte in a text buffer as a single-byte character or as the first or second byte of a double-byte character. (Deprecated. You should update your application to handle Unicode text. There is no replacement function because Unicode handles encoding in a different manner.)

`CharacterType` (page 96) Deprecated in Mac OS X v10.4
: Returns a variety of information about the character represented by a given byte, including its type, class, orientation, direction, case, and size (in bytes). (Deprecated. You should update your application to handle Unicode text. There is no replacement function because Unicode handles encoding in a different manner.)

`FillParseTable` (page 99) Deprecated in Mac OS X v10.4
: Helps your application to quickly process a buffer of mixed single-byte and double-byte characters. (Deprecated. You should update your application to handle Unicode text. There is no replacement function because Unicode handles encoding in a different manner.)

Checking and Setting Script Manager Variables

`GetScriptManagerVariable` (page 110) Deprecated in Mac OS X v10.5
: Retrieves the value of the specified Script Manager variable. (Deprecated. The replacement for this function depends on the selector used with it, as described in the Special Considerations section.)

`SetScriptManagerVariable` (page 113) Deprecated in Mac OS X v10.5
: Sets the specified Script Manager variable to the value of the input parameter. (Deprecated. This is mainly used to set the value of variables that control the internal operation of the Script Manager (selectors `smIntlForce` and `smGenFlags`), and therefore there is no modern replacement.)

Checking and Setting Script Variables

`GetScriptVariable` (page 111) Deprecated in Mac OS X v10.5
: Retrieves the value of the specified script variable from the specified script system. (Deprecated. The replacement for this function depends on the selector used with it, as described in the Special Considerations section.)

`SetScriptVariable` (page 113) Deprecated in Mac OS X v10.5
: Sets the specified script variable for the specified script system to the value of the input parameter. (Deprecated. The replacement for this function depends on the purpose for which it is used, as described in the Special Considerations section.)

Checking and Setting the System Direction

`GetSysDirection` (page 102) Deprecated in Mac OS X v10.4
: Returns the current value of `SysDirection`, the global variable that determines the system direction (primary line direction). (Deprecated. This function does not return anything useful in Mac OS X.)

`SetSysDirection` (page 106) Deprecated in Mac OS X v10.4
: Sets the value of `SysDirection`, the global variable that determines the system direction (primary line direction). (Deprecated. There is no replacement because this function is no longer needed in Mac OS X.)

Determining Script Codes From Font Information

`FontScript` (page 100) Deprecated in Mac OS X v10.4
: Returns the script code for the current script (usually the font script). (Deprecated. Use `ATSFontFamilyGetEncoding` instead.)

`FontToScript` (page 101) Deprecated in Mac OS X v10.4
: Translates a font family ID number into its corresponding script code, if that script system is currently enabled. (Deprecated. Use `ATSFontFamilyGetEncoding` instead.)

`IntlScript` (page 103) Deprecated in Mac OS X v10.4
: Identifies the script system used by the Text Utilities date-formatting, time-formatting, and string-sorting functions. (Deprecated. Use `ATSFontFamilyGetEncoding` instead.)

Directly Accessing International Resources

`GetIntlResource` (page 109) Deprecated in Mac OS X v10.5
: Returns a handle to one of the international resources. (Deprecated. The replacement for this function depends on the purpose for which it is used, as described in the Special Considerations section.)

`ClearIntlResourceCache` (page 98) Deprecated in Mac OS X v10.4
: Clears the application's international resources cache, which contains the resource ID numbers of the string-manipulation (`'itl2'`) and tokens (`'itl4'`) resources for the current script. (Deprecated. There is no replacement because this function is no longer needed in Mac OS X.)

`GetIntlResourceTable` (page 101) Deprecated in Mac OS X v10.4
: Obtains a specific word-selection, line-break, number-parts, untoken, or whitespace table from the appropriate international resource. (Deprecated. There is no replacement because this function is no longer needed in Mac OS X.)

Converting Text

`IntlTokenize` (page 104) Deprecated in Mac OS X v10.4
: Allows your application to convert text into a sequence of language-independent tokens. (Deprecated. There is no replacement because this function is no longer needed in Mac OS X.)

`TransliterateText` (page 106) Deprecated in Mac OS X v10.4
: Converts characters from one subscript to the closest possible approximation in a different subscript within the same double-byte script system. (Deprecated. Use `CFStringUppercase` instead.)

Data Types

CharByteTable

Represents an array of `char` values.

```
typedef  char CharByteTable[256];
```

Discussion
Used by the function `FillParseTable` (page 99).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Script.h`

CommentType

Represents an array of `ScriptTokenType` values.

```
typedef ScriptTokenType CommentType[4];
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Script.h`

DelimType

Represents an array of `ScriptTokenType` values.

```
typedef ScriptTokenType DelimType[2];
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Script.h`

ScriptTokenType

Defins a data type for the script token type.

```
typedef  short ScriptTokenType;
```

Discussion

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Script.h`

TokenBlock

Contains information about text that is to be converted to tokens, the destination of the token list, a handle to the tokens resource, and a set of options.

```
struct TokenBlock {
    Ptr source;
    long sourceLength;
    Ptr tokenList;
    long tokenLength;
    long tokenCount;
    Ptr stringList;
    long stringLength;
    long stringCount;
    Boolean doString;
    Boolean doAppend;
    Boolean doAlphanumeric;
    Boolean doNest;
    ScriptTokenType leftDelims[2];
    ScriptTokenType rightDelims[2];
    ScriptTokenType leftComment[4];
    ScriptTokenType rightComment[4];
    ScriptTokenType escapeCode;
    ScriptTokenType decimalCode;
    Handle itlResource;
    long reserved[8];
};
typedef struct TokenBlock TokenBlock;
typedef TokenBlock * TokenBlockPtr;
```

Fields

`source`
A pointer to a stream of characters. On input to the function `IntlTokenize` (page 104), a pointer to the beginning of the source text (not a Pascal string) to be converted.

`sourceLength`
The length of the source stream. On input, the number of bytes in the source text.

`tokenList`
A pointer to an array of tokens. On input, a pointer to a buffer you have allocated. On output, a pointer to a list of token structures generated by the `IntlTokenize` function.

`tokenLength`
The maximum length of `TokenList`. On input, the maximum size of token list (in number of tokens, not bytes) that will fit into the buffer pointed to by the `tokenList` field.

`tokenCount`
The number of tokens generated by the tokenizer. On input (if `doAppend = TRUE`), must contain the correct number of tokens currently in the token list. (Ignored if `doAppend = FALSE`.) On output, the number of tokens currently in the token list.

`stringList`
: A pointer to a stream of identifiers. On input (if `doString = TRUE`), a pointer to a buffer you have allocated. (Ignored if `doString = FALSE`) On output, a pointer to a list of strings generated by the `IntlTokenize` function.

`stringLength`
: The length of the string list. On input (if `doString = TRUE`), the size in bytes of the string list buffer pointed to by the `stringList` field. (Ignored if `doString = FALSE`.)

`stringCount`
: The number of bytes currently used. On input (if `doString = TRUE` and `doAppend = TRUE`), the correct current size in bytes of the string list. (Ignored if `doString = FALSE` or `doAppend = FALSE`.) On output, the current size in bytes of the string list. (Indeterminate if `doString = FALSE`.)

`doString`
: A Boolean value. On input, if `TRUE`, instructs `IntlTokenize` to create a Pascal string representing the contents of each token it generates. If `FALSE`, `IntlTokenize` generates a token list without an associated string list.

`doAppend`
: A Boolean value. On input, if `TRUE`, instructs `IntlTokenize` to append tokens and strings it generates to the current token list and string list. If `FALSE`, `IntlTokenize` writes over any previous contents of the buffer pointed to by `tokenList` and `stringList`.

`doAlphanumeric`
: A Boolean value. On input, if `TRUE`, instructs `IntlTokenize` to interpret numeric characters as alphabetic when mixed with alphabetic characters. If `FALSE`, all numeric characters are interpreted as numbers.

`doNest`
: A Boolean value. A value of type `Boolean`. On input, if `TRUE`, instructs `IntlTokenize` to allow nested comments (to any depth of nesting). If `FALSE`, comment delimiters may not be nested within other comment delimiters.

`leftDelims`
: A value of type `DelimType`. On input, an array of two integers, each of which contains the token code of the symbol that may be used as an opening delimiter for a quoted literal. If only one opening delimiter is needed, the other must be specified to be `delimPad`.

`rightDelims`
: A value of type `DelimType`. On input, an array of two integers, each of which contains the token code of the symbol that may be used as the matching closing delimiter for the corresponding opening delimiter in the `leftDelims` field.

`leftComment`
: A value of type `CommentType`. On input, an array of two pairs of integers, each pair of which contains codes for the two token types that may be used as opening delimiters for comments.

`rightComment`
: A value of type `CommentType`. On input, an array of two pairs of integers, each pair of which contains codes for the two token types that may be used as closing delimiters for comments.

`escapeCode`
: A value of type `TokenType`. On input, a single integer that contains the token code for the symbol that may be an escape character within a quoted literal.

`decimalCode`
: A value of type `TokenType`. On input, a single integer that contains the token type of the symbol to be used for a decimal point.

`itlResource`
: A value of type `Handle`. On input, a handle to the tokens (`'itl4'`) resource of the script system under which the source text was created.

`reserved`
: An 8-byte array of type `LongInt`. On input, this must be set to 0.

Discussion
The token block structure is a parameter block used to pass information to the `IntlTokenize` (page 104) function and to retrieve results from it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Script.h`

TokenRec

Contains information about the conversion of a sequence of characters to a token.

```
struct TokenRec {
    ScriptTokenType theToken;
    Ptr position;
    long length;
    StringPtr stringPosition;
};
typedef struct TokenRec TokenRec;
typedef TokenRec * TokenRecPtr;
```

Fields

`theToken`
: A numeric code that specifies the type of token (such as whitespace, opening parenthesis, alphabetic or numeric sequence) described by this token structure. Constants for all defined token codes are listed in "Obsolete Token Codes" (page 93).

`position`
: A pointer to the first character in the source text that caused this particular token to be generated.

`length`
: The length, in bytes, of the source text that caused this particular token to be generated.

`stringPosition`
: If `doString` = `TRUE`, a pointer to a null-terminated Pascal string, padded if necessary so that its total number of bytes (length byte + text + null byte + padding) is even. If `doString` = `FALSE`, this field is `NULL`.

: The value in the length byte of the null-terminated Pascal string does not include either the terminating zero byte or the possible additional padding byte. There may be as many as two additional bytes beyond the specified length.

Discussion
The token structure holds the results of the conversion of a sequence of characters to a token by the `IntlTokenize` (page 104) function. When it analyzes text, `IntlTokenize` generates a token list, which is a sequence of token structures.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Script.h`

Constants

Assorted Constants

Calendar Codes

Specify constants for various calendars.

```
enum {
    calGregorian = 0,
    calArabicCivil = 1,
    calArabicLunar = 2,
    calJapanese = 3,
    calJewish = 4,
    calCoptic = 5,
    calPersian = 6
};
```

Constants

`calGregorian`

Specifies the Gregorian calendar.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`calArabicCivil`

Specifies the Arabic civil calendar.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`calArabicLunar`

Specifies the Arabic lunar calendar.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`calJapanese`

Specifies the Japanese calendar.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`calJewish`

Specifies the Jewish calendar.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`calCoptic`
Specifies the Coptic calendar.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`calPersian`
Specifies the Persian calendar.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Discussion

These calendar codes are bit numbers, not masks.

Character Byte Types

Specify character byte types.

```
enum {
    smSingleByte = 0,
    smFirstByte = -1,
    smLastByte = 1,
    smMiddleByte = 2
};
```

Constants

`smSingleByte`
Specifes a single byte.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smFirstByte`
Specifies the first byte.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smLastByte`
Specifies the last byte.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smMiddleByte`
Specifies the middle byte.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Character Types

Specify basic character types.

```
enum {
    smCharPunct = 0x0000,
    smCharAscii = 0x0001,
    smCharEuro = 0x0007,
    smCharExtAscii = 0x0007,
    smCharKatakana = 0x0002,
    smCharHiragana = 0x0003,
    smCharIdeographic = 0x0004,
    smCharTwoByteGreek = 0x0005,
    smCharTwoByteRussian = 0x0006,
    smCharBidirect = 0x0008,
    smCharContextualLR = 0x0009,
    smCharNonContextualLR = 0x000A,
    smCharHangul = 0x000C,
    smCharJamo = 0x000D,
    smCharBopomofo = 0x000E,
    smCharGanaKana = 0x000F,
    smCharFISKana = 0x0002,
    smCharFISGana = 0x0003,
    smCharFISIdeo = 0x0004
};
```

Constants

`smCharPunct`

Specifies punctuation characters.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smCharAscii`

Specifies ASCII characters.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smCharEuro`

Specifies `smCharEuro`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smCharExtAscii`

Specifies a more correct synonym for `smCharEuro`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smCharKatakana`

Specifies additional character types for Japanese Katakana.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smCharHiragana`

Specifies additional character types for Japanese Hiragana.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smCharIdeographic`
: Specifies additional character types for Hanzi, Kanji, and Hanja.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smCharTwoByteGreek`
: Specifies additional character types for double-byte Greek in Far East systems.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smCharTwoByteRussian`
: Specifies additional character types for double-byte Cyrillic in Far East systems.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smCharBidirect`
: Specifies additional character types for Arabic/Hebrew.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smCharContextualLR`
: Specifies contextual left-right: Thai, Indic scripts.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smCharNonContextualLR`
: Specifies additional character types for non-contextual left-right: Cyrillic, Greek.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smCharHangul`
: Specifies additional character types for Korean Hangul.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smCharJamo`
: Specifies additional character types for Korean Jamo.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smCharBopomofo`
: Specifies additional character types for Chinese Bopomofo.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smCharGanaKana`
: Specifies additional character types shared for Japanese Hiragana and Katakana.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smCharFISKana`
: Specifies obsolete Katakana names, for backward compatibility.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smCharFISGana`
: Specifies obsolete Hiragana namde, for backward compatibility.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smCharFISIdeo`
: Specifies obsolete Hanzi, Kanji, and Hanja names, for backward compatibility.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Character Type Classes

Specify character-type classes for double-byte script systems.

```
enum {
    smCharFISGreek = 0x0005,
    smCharFISRussian = 0x0006,
    smPunctNormal = 0x0000,
    smPunctNumber = 0x0100,
    smPunctSymbol = 0x0200,
    smPunctBlank = 0x0300,
    smPunctRepeat = 0x0400,
    smPunctGraphic = 0x0500,
    smKanaSmall = 0x0100,
    smKanaHardOK = 0x0200,
    smKanaSoftOK = 0x0300,
    smIdeographicLevel1 = 0x0000,
    smIdeographicLevel2 = 0x0100,
    smIdeographicUser = 0x0200,
    smFISClassLvl1 = 0x0000,
    smFISClassLvl2 = 0x0100,
    smFISClassUser = 0x0200,
    smJamoJaeum = 0x0000,
    smJamoBogJaeum = 0x0100,
    smJamoMoeum = 0x0200,
    smJamoBogMoeum = 0x0300
};
```

Constants

`smCharFISGreek`
: Specfies character-type classes for double-byte Greek in Far East systems.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smCharFISRussian`
: Specfies character-type classes for double-byte Cyrillic in Far East systems.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smPunctNormal`
Specfies character-type classes for normal punctuation (`smCharPunct`).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smPunctNumber`
Specfies character-type classes for number punctuation (`smCharPunct`).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smPunctSymbol`
Specfies character-type classes for symbol punctuation (`smCharPunct`).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smPunctBlank`
Specfies additional character-type classes for punctuation in double-byte systems.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smPunctRepeat`
Specifies a character-type class for repeat markers.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smPunctGraphic`
Specifies a character-type class forl ine graphics.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKanaSmall`
Specfies character-type classes for Katakana and Hiragana double-byte systems.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKanaHardOK`
Specfies character-type classes for Katakana and Hiragana double-byte systems; can have dakuten.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKanaSoftOK`
Specfies character-type classes for Katakana and Hiragana double-byte systems; can have dakuten or han-dakuten.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smIdeographicLevel1`
Specfies character-type classes for Ideographic double-byte systems; level 1 char.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smIdeographicLevel2`
: Specfies character-type classes for Ideographic double-byte systems; level 2 char.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smIdeographicUser`
: Specfies character-type classes for Ideographic double-byte systems; user char.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smFISClassLvl1`
: Obsolete, for backward compatibility; level 1 char.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smFISClassLvl2`
: Obsolete, for backward compatibility; level 2 char.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smFISClassUser`
: Obsolete, for backward compatibility; user char.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smJamoJaeum`
: Specfies character-type Jamo classes for Korean systems; simple consonant char.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smJamoBogJaeum`
: Specfies character-type Jamo classes for Korean systems; complex consonant char.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smJamoMoeum`
: Specfies character-type Jamo classes for Korean systems; simple vowel char.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smJamoBogMoeum`
: Specfies character-type Jamo classes for Korean systems; complex vowel char.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Character Type Field Masks

Specify masks used to extract information from the return value of the `CharacterType` function.

```
enum {
    smcTypeMask = 0x000F,
    smcReserved = 0x00F0,
    smcClassMask = 0x0F00,
    smcOrientationMask = 0x1000,
    smcRightMask = 0x2000,
    smcUpperMask = 0x4000,
    smcDoubleMask = 0x8000
};
```

Constants

`smcTypeMask`

Character-type mask.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smcReserved`

Reserved.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smcClassMask`

Character-class mask.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smcOrientationMask`

Character orientation (double-byte scripts).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smcRightMask`

Writing direction (bidirectional scripts); main character set or subset (double-byte scripts)

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smcUpperMask`

Uppercase or lowercase.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smcDoubleMask`

Size (1 or 2 bytes).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Discussion

These bit masks are used to extract fields from the return value of the `CharacterType` (page 96) function.

The character type of the character in question is the result of performing an `AND` operation with `smcTypeMask` and the `CharacterType` result.

The character class of the character in question is the result of performing an `AND` operation with `smcClassMask` and the `CharacterType` result. Character classes can be considered as subtypes of character types.

The orientation of the character in question is the result of performing an `AND` operation with `smcOrientationMask` and the `CharacterType` result. The orientation value can be either `smCharHorizontal` or `smCharVertical`.

The direction of the character in question is the result of performing an `AND` operation with `smcRightMask` and the `CharacterType` result. The direction value can be either `smCharLeft` (left-to-right) or `smCharRight` (right-to-left).

The case of the character in question is the result of performing an `AND` operation with `smcUpperMask` and the `CharacterType` result. The case value can be either `smCharLower` or `smCharUpper`.

The size of the character in question is the result of performing an `AND` operation with `smcDoubleMask` and the `CharacterType` result. The size value can be either `smChar1byte` or `smChar2byte`.

Character Set Extensions

Specify extensions to character sets.

```
enum {
    diaeresisUprY = 0xD9,
    fraction = 0xDA,
    intlCurrency = 0xDB,
    leftSingGuillemet = 0xDC,
    rightSingGuillemet = 0xDD,
    fiLigature = 0xDE,
    flLigature = 0xDF,
    dblDagger = 0xE0,
    centeredDot = 0xE1,
    baseSingQuote = 0xE2,
    baseDblQuote = 0xE3,
    perThousand = 0xE4,
    circumflexUprA = 0xE5,
    circumflexUprE = 0xE6,
    acuteUprA = 0xE7,
    diaeresisUprE = 0xE8,
    graveUprE = 0xE9,
    acuteUprI = 0xEA,
    circumflexUprI = 0xEB,
    diaeresisUprI = 0xEC,
    graveUprI = 0xED,
    acuteUprO = 0xEE,
    circumflexUprO = 0xEF,
    appleLogo = 0xF0,
    graveUprO = 0xF1,
    acuteUprU = 0xF2,
    circumflexUprU = 0xF3,
    graveUprU = 0xF4,
    dotlessLwrI = 0xF5,
    circumflex = 0xF6,
    tilde = 0xF7,
    macron = 0xF8,
    breveMark = 0xF9,
    overDot = 0xFA,
    ringMark = 0xFB,
    cedilla = 0xFC,
    doubleAcute = 0xFD,
    ogonek = 0xFE,
    hachek = 0xFF
};
```

Keyboard Script Synchronization

Specifies to disable font and keyboard script synchronization.

```
enum {
    smfDisableKeyScriptSync = 27
};
```

Glyph Orientations

Specify character-type glyph orientation for double-byte systems.

```
enum {
    smCharHorizontal = 0x0000,
    smCharVertical = 0x1000,
    smCharLeft = 0x0000,
    smCharRight = 0x2000,
    smCharLower = 0x0000,
    smCharUpper = 0x4000,
    smChar1byte = 0x0000,
    smChar2byte = 0x8000
};
```

Constants

`smCharHorizontal`

Specifies horizontal character form.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smCharVertical`

Specifies vertical character form.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smCharLeft`

Specifies left character direction.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smCharRight`

Specifies right character direction.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smCharLower`

Specifies lowercase character modifers.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smCharUpper`

Specifies uppercase character modifers.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smChar1byte`

Specifies character size modifiers (single or multiple bytes).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smChar2byte`

Specifies character size modifiers (single or multiple bytes).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Keyboard Script Switching Selectors

Specify a keyboard script switching flag and mask.

```
enum {
    smKeyForceKeyScriptBit = 7,
    smKeyForceKeyScriptMask = 1 << smKeyForceKeyScriptBit
};
```

Constants

`smKeyForceKeyScriptBit`

A flag that specifies to force keyboard script switching.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeyForceKeyScriptMask`

A mask that specifies to force keyboard script switching.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Keyboard Script Values

Specify actions for keyboard scripts.

```
enum {
    smKeyNextScript = -1,
    smKeySysScript = -2,
    smKeySwapScript = -3,
    smKeyNextKybd = -4,
    smKeySwapKybd = -5,
    smKeyDisableKybds = -6,
    smKeyEnableKybds = -7,
    smKeyToggleInline = -8,
    smKeyToggleDirection = -9,
    smKeyNextInputMethod = -10,
    smKeySwapInputMethod = -11,
    smKeyDisableKybdSwitch = -12,
    smKeySetDirLeftRight = -15,
    smKeySetDirRightLeft = -16,
    smKeyRoman = -17
};
```

Constants

`smKeyNextScript`

Specifies to switch to the next available script.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeySysScript`

Specfiies to switch to the system script.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKeySwapScript`
: Specifies to switch to the previously-used script

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smKeyNextKybd`
: Specifies to switch to the next keyboard in current keyscript.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smKeySwapKybd`
: Specfies to switch to a previously-used keyboard in the current keyscript.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smKeyDisableKybds`
: Specifies to disable keyboards not in the system or Roman script.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smKeyEnableKybds`
: Specifies to enable keyboards for all enabled scripts.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smKeyToggleInline`
: Specifies to toggle inline input for the current keyscript

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smKeyToggleDirection`
: Specifies to toggle the default line direction (`TESysJust`).

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smKeyNextInputMethod`
: Specfies to switch to the next input method in the current keyscript.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smKeySwapInputMethod`
: Specfies to switch to the last-used input method in the current keyscript.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smKeyDisableKybdSwitch`
: Specfies to disable switching from the current keyboard.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smKeySetDirLeftRight`
: Specfies to set the default line direction to left-right, align left.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smKeySetDirRightLeft`
: Specfies to set the default line direction to right-left, align right.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smKeyRoman`
: Specfies to set the keyscript to Roman. Does nothing if on a Roman-only system. This is unlike `KeyScript(smRoman)` which forces an update to current default Roman keyboard. See `KeyScript` documentation for more information.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Keyboard Synchronization Mask

Disables font and keyboard script synchronization mask

```
enum {
    smfDisableKeyScriptSyncMask = 1L << smfDisableKeyScriptSync
};
```

Constants

`smfDisableKeyScriptSyncMask`
: Disable font and keyboard script synchronization mask

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Discussion

Meta Script Codes

Specify implicit script codes.

```
enum {
    smSystemScript = -1,
    smCurrentScript = -2,
    smAllScripts = -3
};
```

Constants

`smSystemScript`
: Specifies the system script.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smCurrentScript`
: Specifies the font script.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smAllScripts`
: Specfies any script.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Discussion

You can specify script systems with implicit and explicit script code constants in the `script` parameter of the `GetScriptVariable` (page 111) and `SetScriptVariable` (page 113) functions. The implicit script codes `smSystemScript` and `smCurrentScript` are special negative values for the system script and the font script, respectively.

Negative Verbs

Specify special negative verbs that were associated with WorldScript I.

```
enum {
    smLayoutCache = -309,
    smOldVerbSupport = -311,
    smSetKashidas = -291,
    smSetKashProp = -287,
    smScriptSysBase = -281,
    smScriptAppBase = -283,
    smScriptFntBase = -285,
    smScriptLigatures = -263,
    smScriptNumbers = -267
};
```

Constants

`smLayoutCache`
: Specifies that `HiWrd(param)` is the number of entries, `LoWrd` is the maximum input length

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smOldVerbSupport`
: Specifies that a parameter is added to old verbs to map to WorldScript I verb.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smSetKashidas`
: Specifies parameter is on or off; obsolete verb = -36.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smSetKashProp`
: Specifies parameter is kashida proportion; obsolete verb = -32.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptSysBase`
: Specifies parameter is associated font to use with the system font; obsolete verb = -26)

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptAppBase`
: Specifies parameter is associated font to use with application font; obsolete verb = -28.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptFntBase`
: Specifies that a parameter is associated font to use with all other fonts; obsolete verb = -30.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptLigatures`
: Obsolete verb = -8.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptNumbers`
: Obsolete verb = -12.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

Numeral Codes

Specify the kinds of numerals used by a script.

```
enum {
    intWestern = 0,
    intArabic = 1,
    intRoman = 2,
    intJapanese = 3,
    intEuropean = 4,
    intOutputMask = 0x8000
};
```

Constants

`intWestern`

Specifies Western numerals.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`intArabic`

Specifies Native Arabic numerals.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`intRoman`

Specifies Roman numerals.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`intJapanese`

Specifies Japanese numerals.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`intEuropean`

Specifies European numerals.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`intOutputMask`

Specifies an output mask.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Discussion

These constants specify bit numbers, not masks.

Script Redraw Selectors

Specify values for script redraw flags.

```
enum {
    smRedrawChar = 0,
    smRedrawWord = 1,
    smRedrawLine = -1
};
```

Constants

`smRedrawChar`

Specifies to redraw character only.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smRedrawWord`

Specifies to redraw entire word (double-byte systems).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smRedrawLine`

Specifies to redraw entire line (bidirectional systems).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

Script Codes

Specify Mac OS encodings that are related to a `FOND` ID range.

```
enum {
    smRoman = 0,
    smJapanese = 1,
    smTradChinese = 2,
    smKorean = 3,
    smArabic = 4,
    smHebrew = 5,
    smGreek = 6,
    smCyrillic = 7,
    smRSymbol = 8,
    smDevanagari = 9,
    smGurmukhi = 10,
    smGujarati = 11,
    smOriya = 12,
    smBengali = 13,
    smTamil = 14,
    smTelugu = 15,
    smKannada = 16,
    smMalayalam = 17,
    smSinhalese = 18,
    smBurmese = 19,
    smKhmer = 20,
    smThai = 21,
    smLao = 22,
    smGeorgian = 23,
    smArmenian = 24,
    smSimpChinese = 25,
    smTibetan = 26,
    smMongolian = 27,
    smEthiopic = 28,
    smGeez = 28,
    smCentralEuroRoman = 29,
    smVietnamese = 30,
    smExtArabic = 31,
    smUninterp = 32
};
```

Constants

`smRoman`

Specifies the Roman script system.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smJapanese`

Specifies the Japanese script system.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smTradChinese`

Specifies the traditional Chinese script system.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smKorean`
Specifies the Korean script system.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smArabic`
Specifies the Arabic script system.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smHebrew`
Specifies the Hebrew script system.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smGreek`
Specifies the Greek script system.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smCyrillic`
Specifies the Cyrillic script system.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smRSymbol`
Specifies right-to-left symbols. The script code represented by the constant `smRSymbol` is available as an alternative to `smUninterp`, for representation of special symbols that have a right-to-left line direction. Note, however, that the script management system provides no direct support for representation of text with this script code.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smDevanagari`
Specifies the Devanagari script system.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smGurmukhi`
Specifies the Gurmukhi script system.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smGujarati`
Specifies the Gujarati script system.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smOriya`
Specifies the Oriya script system.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smBengali`
    Specifies the Bengali script system.
    Available in Mac OS X v10.0 and later.
    Declared in `Script.h`.

`smTamil`
    Specifies the Tamil script system.
    Available in Mac OS X v10.0 and later.
    Declared in `Script.h`.

`smTelugu`
    Specifies the Telugu script system.
    Available in Mac OS X v10.0 and later.
    Declared in `Script.h`.

`smKannada`
    Specifies the Kannada/Kanarese script system.
    Available in Mac OS X v10.0 and later.
    Declared in `Script.h`.

`smMalayalam`
    Specifies the Malayalam script system.
    Available in Mac OS X v10.0 and later.
    Declared in `Script.h`.

`smSinhalese`
    Specifies the Sinhalese script system.
    Available in Mac OS X v10.0 and later.
    Declared in `Script.h`.

`smBurmese`
    Specifies the Burmese script system.
    Available in Mac OS X v10.0 and later.
    Declared in `Script.h`.

`smKhmer`
    Specifies the Khmer script system.
    Available in Mac OS X v10.0 and later.
    Declared in `Script.h`.

`smThai`
    Specifies the Thai script system.
    Available in Mac OS X v10.0 and later.
    Declared in `Script.h`.

`smLao`
    Specifies the Laotian script system.
    Available in Mac OS X v10.0 and later.
    Declared in `Script.h`.

`smGeorgian`
: Specifies the Georgian script system.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smArmenian`
: Specifies the Armenian script system.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smSimpChinese`
: Specifies the simplified Chinese script system.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smTibetan`
: Specifies the Tibetan script system.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smMongolian`
: Specifies the Mongolian script system.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smEthiopic`
: Specifies the Geez/Ethiopic script system. This constant is the same as `smGeez`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smGeez`
: Specifies the Geez/Ethiopic script system.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smCentralEuroRoman`
: Used for Czech, Slovak, Polish, Hungarian, Baltic languages.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smVietnamese`
: Specifies the Extended Roman script system for Vietnamese.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smExtArabic`
: Specifies the extended Arabic for Sindhi script system.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smUninterp`

Uninterpreted symbols. The script code represented by the constant `smUninterp` is available for representation of special symbols, such as items in a tool palette, that must not be considered as part of any actual script system. For manipulating and drawing such symbols, the `smUninterp` constant should be treated as if it indicated the Roman script system rather than the system script; that is, the default behavior of uninterpreted symbols should be Roman.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Script Code - Unicode Input

Specifies the extended script code for full Unicode input.

```
enum {
    smUnicodeScript = 0x7E
};
```

Script Constants

Specify constants used to get and set script variables.

```
enum {
    smScriptNumDate = 30,
    smScriptKeys = 32,
    smScriptIcon = 34,
    smScriptPrint = 36,
    smScriptTrap = 38,
    smScriptCreator = 40,
    smScriptFile = 42,
    smScriptName = 44,
    smScriptMonoFondSize = 78,
    smScriptPrefFondSize = 80,
    smScriptSmallFondSize = 82,
    smScriptSysFondSize = 84,
    smScriptAppFondSize = 86,
    smScriptHelpFondSize = 88,
    smScriptValidStyles = 90,
    smScriptAliasStyle = 92
};
```

Constants

`smScriptNumDate`

(2 bytes) The numeral code and calendar code for the script. The numeral code specifies the kind of numerals the script uses, and is in the high-order byte of the word the calendar code specifies the type of calendar it uses and is in the low-order byte of the word. The value of this variable is initialized from the script system's international bundle resource. It may be changed during execution when the user selects, for example, a new calendar from a script system's control panel. See "Numeral Codes" (page 27) and "Calendar Codes" (page 12) for the different codes.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptKeys`
: (2 bytes) The resource ID of the script's current keyboard-layout (`'KCHR'`) resource. The keyboard-layout resource is used to map virtual key codes into the correct character codes for the script. The value of this variable is initialized from the script system's international bundle resource. It is updated when the user selects a new keyboard layout, or when the application calls the `KeyScript` function. You can force a particular keyboard layout to be used with your application by setting the value of this variable and then calling `KeyScript`.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptIcon`
: (2 bytes) The resource ID of the script's keyboard icon family (resource types `'kcs#'`, `'kcs4'`, and `'kcs8'`). The keyboard icon family consists of the keyboard icons displayed in the keyboard menu. The value of this variable is initialized from the script system's international bundle resource. Note that, unlike `smScriptKeys`, the value of this variable is not automatically updated when the keyboard layout changes. (System software assumes that the icon family has an identical ID to the keyboard-layout resource, and usually ignores this variable.)

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptPrint`
: (4 bytes) The print action function vector, set up by the script system (or by the Script Manager if the `smsfAutoInit` bit is set) when the script is initialized.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptTrap`
: (4 bytes) A pointer to the script's script-structure dispatch function (for internal use only).

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptCreator`
: (4 bytes) The 4-character creator type for the script system's file, that is, the file containing the script system. For the Roman script system, it is `'ZSYS'`, for WorldScript I it is `'univ'`, and for World Script II it is `'doub'`.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptFile`
: (4 bytes) A pointer to the Pascal string that contains the name of the script system's file, that is, the file containing the script system. For the Roman script system, the string is `'System'`.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptName`
: (4 bytes) A pointer to a Pascal string that contains the script system's name. For the Roman script system and single-byte simple script systems, the string is `'Roman'`. For single-byte complex script systems, this name is taken from the encoding/rendering (`'itl5'`) resource. For double-byte script systems, it is taken from the WorldScript II extension and is `'WorldScript II'`.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptMonoFondSize`
: (4 bytes) The default font family ID and size (in points) for monospaced text. The ID is stored in the high-order word, and the size is stored in the low-order word. The value of this variable is taken from the script system's international bundle resource. Note that not all script systems have a monospaced font.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptPrefFondSize`
: (4 bytes) Currently not used.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptSmallFondSize`
: (4 bytes) The default font family ID and size (in points) for small text, generally the smallest font and size combination that is legible on screen. The ID is stored in the high-order word, and the size is stored in the low-order word. Sizes are important for example, a 9-point font may be too small in Chinese. The value of this variable is taken from the script system's international bundle resource.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptSysFondSize`
: (4 bytes) The default font family ID and size (in points) for this script system's preferred system font. The ID is stored in the high-order word, and the size is stored in the low-order word. The value of this variable is taken from the script system's international bundle resource.

  This variable holds similar information to the variable accessed through the `smScriptSysFond` selector. If you need font family ID only and don't want size information, it is simpler to use `smScriptSysFond`. Note, however, that changing the value of this variable has no effect on the value accessed through `smScriptSysFond`.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptAppFondSize`

(4 bytes) The default font family ID and size (in points) for this script system's preferred application font. The ID is stored in the high-order word, and the size is stored in the low-order word. The value of this variable is taken from the script system's international bundle resource.

This variable holds similar information to the variable accessed through the `smScriptAppFond` selector. If you need font family ID only and don't want size information, it is simpler to use `smScriptAppFond`. Note, however, that changing the value of this variable has no effect on the value accessed through `smScriptAppFond`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptHelpFondSize`

(4 bytes) The default font family ID and size (in points) for Balloon Help. The ID is stored in the high-order word, and the size is stored in the low-order word. Sizes are important for example, a 9-point font may be too small in Chinese. The value of this variable is taken from the script system's international bundle resource.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptValidStyles`

(1 byte) The set of all valid styles for the script. For example, the Extended style is not valid in the Arabic script. When the `GetScriptVariable` function is called with the `smScriptValidStyles` selector, the low-order byte of the returned value is a style code that includes all of the valid styles for the script (that is, the bit corresponding to each QuickDraw style is set if that style is valid for the specified script). The value of this variable is taken from the script system's international bundle resource.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptAliasStyle`

(1 byte) The style to use for indicating aliases. When the `GetScriptVariable` function is called with `smScriptAliasStyle`, the low-order byte of the returned value is the style code that should be used in that script for indicating alias names (for example, in the Roman script system, alias names are indicated in italics). The value of this variable is taken from the script system's international bundle resource.

Some script systems, such as Arabic and Hebrew, have private script-system selectors that are unique to those scripts. Those private selectors are negative, whereas selectors that extend across script systems are positive.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

Script Flag Attributes

Specify bits used to examine attributes in the script flags word.

```
enum {
    smsfIntellCP = 0,
    smsfSingByte = 1,
    smsfNatCase = 2,
    smsfContext = 3,
    smsfNoForceFont = 4,
    smsfB0Digits = 5,
    smsfAutoInit = 6,
    smsfUnivExt = 7,
    smsfSynchUnstyledTE = 8,
    smsfForms = 13,
    smsfLigatures = 14,
    smsfReverse = 15,
    smfShowIcon = 31,
    smfDualCaret = 30,
    smfNameTagEnab = 29,
    smfUseAssocFontInfo = 28
};
```

Constants

`smsfIntellCP`

Specifies the script can support intelligent cut and paste (it uses spaces as word delimiters).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smsfSingByte`

Specifies the script has only single-byte characters.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smsfNatCase`

Specifies the script has both uppercase and lowercase native characters.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smsfContext`

Specifies the script is contextual.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smsfNoForceFont`

Specifies the script does not support font forcing (ignores the font force flag).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smsfB0Digits`

Specifies the script has alternate digits at $B0–$B9. Arabic and Hebrew, for example, have their native numeric forms at this location in their character sets.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smsfAutoInit`
: Specifies the script is initialized by the Script Manager. Single-byte simple script systems can set this bit to avoid having to initialize themselves.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smsfUnivExt`
: Specifies the script uses the WorldScript I extension.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smsfSynchUnstyledTE`
: Specifies the script synchronizes keyboard with font for monostyled TextEdit.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smsfForms`
: Specifies to use contextual forms if this bit is set; do not use them if it is cleared.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smsfLigatures`
: Specifies to use contextual ligatures if this bit is set; do not use them if it is cleared.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smsfReverse`
: Specifies reverse right-to-left text to draw it in (left-to-right) display order if this bit is set; do not reorder text if this bit is cleared.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smfShowIcon`
: Specifies to show icon even if only one script; bits in the `smGenFlags` long.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smfDualCaret`
: Specifies to use dual caret for mixed direction text; bits in the `smGenFlags` long.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smfNameTagEnab`
: Reserved for internal use; bits in the `smGenFlags` long.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smfUseAssocFontInfo`
: Specifies to set the associated font info for `FontMetrics` calls; bits in the `smGenFlags` long.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Discussion
These constants are available for examining attributes in the script flags word. Bits above 8 are nonstatic, meaning that they may change during program execution. (Note that the constant values represent bit numbers in the flags word, not masks.)

Script Manager Selectors

Specify selectors you can use with the functions `GetScriptManagerVariable` and `SetScriptManagerVariable`.

```
enum {
    smVersion = 0,
    smMunged = 2,
    smEnabled = 4,
    smBidirect = 6,
    smFontForce = 8,
    smIntlForce = 10,
    smForced = 12,
    smDefault = 14,
    smPrint = 16,
    smSysScript = 18,
    smLastScript = 20,
    smKeyScript = 22,
    smSysRef = 24,
    smKeyCache = 26,
    smKeySwap = 28,
    smGenFlags = 30,
    smOverride = 32,
    smCharPortion = 34,
    smDoubleByte = 36,
    smKCHRCache = 38,
    smRegionCode = 40,
    smKeyDisableState = 42
};
```

Constants

`smVersion`

The Script Manager version number (2 bytes) . This variable has the same format as the version number obtained from calling the `Gestalt` function with the Gestalt selector `gestaltScriptMgrVersion`. The high-order byte contains the major version number, and the low-order byte contains the minor version number.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smMunged`

The modification count for Script Manager variables (2 bytes) . At startup, `smMunged` is initialized to 0, and it is incremented when the `KeyScript` function changes the current keyboard script and updates the variables accessed via `smKeyScript` and `smLastScript`. The `smMunged` selector is also incremented when the `SetScriptManagerVariable` function is used to change a Script Manager variable. You can check this variable at any time to see whether any of your own data structures that may depend on Script Manager variables need to be updated.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smEnabled`

The script count (1 byte) ; the number of currently enabled script systems. At startup time, the Script Manager initializes the script count to 0, then increments it for each installed and enabled script system (including Roman). You can use `smEnabled` to determine whether more than one script system is installed—that is, whether your application needs to handle non-Roman text.

Never call `SetScriptManagerVariable` with the `smEnabled` selector. It could result in inconsistency with other script system values.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smBidirect`

The bidirectional flag, which indicates when at least one bidirectional script system is enabled. This flag is set to `TRUE` ($FF) if the Arabic or Hebrew script system is enabled.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smFontForce`

The font force flag (1 byte). At startup, the Script Manager sets its value from the system script's international configuration (`'itlc'`) resource. The flag returns 0 for `FALSE` and $FF for `TRUE`. If the system script is non-Roman, the font force flag controls whether a font with ID in the Roman script range is interpreted as belonging to the Roman script or to the system script.

When you call `SetScriptManagerVariable` with the `smFontForce` selector, be sure to pass only the value 0 or $FF, or a later call to `GetScriptManagerVariable` may return an unrecognized value.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smIntlForce`

The international resources selection flag (1 byte). At startup, the Script Manager sets its value from the system script's international configuration (`'itlc'`) resource. The flag returns 0 for `FALSE` and `$FF` for `TRUE`. This flag controls whether international resources of the font script or the system script are used for string manipulation.

When you call `SetScriptManagerVariable` with the `smIntlForce` selector, be sure to pass only the value 0 or $FF, or a later call to `GetScriptManagerVariable` may return an unrecognized value.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smForced`
: The script-forced result flag (1 byte). If the current script has been forced to the system script, this flag is set to `TRUE`. Use the `smForced` selector to obtain reports of the actions of the `FontScript`, `FontToScript`, and `IntlScript` functions. This variable is for information only; never set its value with `SetScriptManagerVariable`.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smDefault`
: The script-defaulted result flag (1 byte). If the script system corresponding to a specified font is not available, this flag is set to `TRUE`. Use this selector to obtain reports of the actions of the `FontScript`, `FontToScript`, and `IntlScript` functions. This variable is for information only; never set its value with `SetScriptManagerVariable`.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smPrint`
: The print action function vector, set up by the Script Manager at startup (4 bytes).

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smSysScript`
: The system script code (2 bytes) . At startup, the Script Manager initializes this variable from the system script's international configuration (`'itlc'`) resource. This variable is for information only; never set its value with `SetScriptManagerVariable`. Constants for all defined script codes are listed in "Region Codes A" (page 70).

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smLastScript`
: The previously used keyboard script (2 bytes). When you change keyboard scripts with the `KeyScript` function, the Script Manager moves the old value of `smKeyScript` into `smLastScript`. `KeyScript` can also swap the current keyboard script with the previous keyboard script, in which case the contents of `smLastScript` and `smKeyScript` are swapped. Constants for all defined script codes are listed in "Region Codes A" (page 70). Never set the value of this variable with `SetScriptManagerVariable`.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smKeyScript`
: The current keyboard script (2 bytes) . The `KeyScript` function tests and updates this variable. When you change keyboard scripts with the `KeyScript` function, the Script Manager moves the old value of `smKeyScript` into `smLastScript`. `KeyScript` can also swap the current keyboard script with the previous keyboard script, in which case the contents of `smLastScript` and `smKeyScript` are swapped. The Script Manager also uses this variable to get the proper keyboard icon and to retrieve the proper keyboard-layout (`'KCHR'`) resource. Constants for all defined script codes are listed in "Region Codes A" (page 70). Never set the value of this variable directly with `SetScriptManagerVariable`; call `KeyScript` to change keyboard scripts.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smSysRef`
: The System Folder volume reference number (2 bytes) . Its value is initialized from the system global variable `BootDrive` at startup.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smKeyCache`
: An obsolete variable (4 bytes). This variable at one time held a pointer to the keyboard cache. The value it provided was not correct and should not be used.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smKeySwap`
: A handle to the keyboard-swap (`'KSWP'`) resource (4 bytes). The Script Manager initializes the handle at startup. The keyboard-swap resource controls the key combinations with which the user can invoke various actions with the `KeyScript` function, such as switching among script systems.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smGenFlags`
: The general flags used by the Script Manager (4 bytes). The Script Manager general flags is a long word value its high-order byte is set from the flags byte in the system script's international configuration (`'itlc'`) resource. These constants are available to designate bits in the variable accessed through `smGenFlags`:

  - `smfNameTagEnab` (a value of 29)Reserved for internal use.
  - `smfDualCaret` (a value of 30)Use a dual caret for mixed-directional text.
  - `smfShowIcon` (a value of 31)Show the keyboard menu even if only one keyboard layout or one script (Roman) is available. (This bit is checked only at system startup.)

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smOverride`
: The script override flags (4 bytes). At present, these flags are not set or used by the Script Manager. They are, however, reserved for future use.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smCharPortion`
: A value used by script systems to allocate intercharacter and interword spacing when justifying text (2 bytes). It denotes the weight allocated to intercharacter space versus interword space. The value of this variable is initialized to 10 percent by the Script Manager, although it currently has no effect on text of the Roman script system. The variable is in 4.12 fixed-point format, which is a 16-bit signed number with 4 bits of integer and 12 bits of fraction. (In that format, 10 percent has the hexadecimal value $0199.)

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smDoubleByte`
: The double-byte flag, a Boolean value that is `TRUE` if at least one double-byte script system is enabled. (1 byte)

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smKCHRCache`
: (A pointer to the cache that stores a copy of the current keyboard-layout (`'KCHR'`) resource 4 bytes).

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smRegionCode`
: The region code for this localized version of system software, obtained from the system script's international configuration (`'itlc'`) resource. This variable identifies the localized version of the system script. Constants for all defined region codes are listed in "Region Codes A" (page 70) (2 bytes).

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smKeyDisableState`

The current disable state for keyboards (1 byte). The Script Manager disables some keyboard scripts or keyboard switching when text input must be restricted to certain script systems or when script systems are being moved into or out of the System file. These are the possible values for the variable accessed through `smKeyDisableState`:

- 0All keyboards are enabled; switching is enabled.
- 1Keyboard switching is disabled.
- $FFKeyboards for all non-Roman secondary scripts are disabled

The script management system maintains the keyboard disable state separately for each application. Never set the value of this variable directly with `SetScriptManagerVariable`; call `KeyScript` to change the keyboard disable state for your application.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

Discussion

This section lists and describes the selector constants for accessing the Script Manager variables through calls to the `GetScriptManagerVariable` (page 110) and `SetScriptManagerVariable` (page 113) functions. In every case the variable parameter passed to or from the function is a long integer (4 bytes); the number in parentheses indicates how many of the 4 bytes are necessary to hold the input or return value for that variable. If fewer than 4 bytes are needed, the low byte or low word contains the information.

Script Variable Selectors

Specify script variables to get or set using the functions `GetScriptVariable` and `SetScriptVariable`.

```
enum {
    smScriptVersion = 0,
    smScriptMunged = 2,
    smScriptEnabled = 4,
    smScriptRight = 6,
    smScriptJust = 8,
    smScriptRedraw = 10,
    smScriptSysFond = 12,
    smScriptAppFond = 14,
    smScriptBundle = 16,
    smScriptNumber = 16,
    smScriptDate = 18,
    smScriptSort = 20,
    smScriptFlags = 22,
    smScriptToken = 24,
    smScriptEncoding = 26,
    smScriptLang = 28
};
```

Constants

`smScriptVersion`

The script system's version number (2 bytes). When the Script Manager loads the script system, the script system puts its current version number into this variable. The high-order byte contains the major version number, and the low-order byte contains the minor version number.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptMunged`

The modification count for this script system's script variables. (2 bytes)The Script Manager increments the variable accessed by the `smScriptMunged` selector each time the `SetScriptVariable` function is called for this script system. You can check this variable at any time to see whether any of your own data structures that depend on this script system's script variables need to be updated.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptEnabled`

The script-enabled flag, a Boolean value that indicates whether the script has been enabled (1 byte). It is set to $FF when enabled and to 0 when not enabled. Note that this variable is not equivalent to the Script Manager variable accessed by the `smEnabled` selector, which is a count of the total number of enabled script systems.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptRight`

The right-to-left flag, a Boolean value that indicates whether the primary line direction for text in this script is right-to-left or left-to-right (1 byte). It is set to $FF for right-to-left text (used in Arabic and Hebrew script systems) and to 0 for left-to-right (used in Roman and other script systems).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptJust`

The script alignment flag, a byte that specifies the default alignment for text in this script system (1 byte). It is set to $FF for right alignment (common for Arabic and Hebrew), and it is set to 0 for left alignment (common for Roman and other script systems). This flag usually has the same value as the `smScriptRight` flag.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptRedraw`

The script-redraw flag, a byte that provides redrawing recommendations for text of this script system (1 byte). It describes how much of a line should be redrawn when a user adds, inserts, or deletes text. It is set to 0 when only a character should be redrawn (used by the Roman script system), to 1 when an entire word should be redrawn (used by the Japanese script system), and to –1 when the entire line should be redrawn (used by the Arabic and Hebrew script systems). These constants are available for the script-redraw flag:

- `smRedrawChar` (a value of 0)Redraw the character only.
- `smRedrawWord` (a value of 1)Redraw the entire word.
- `smRedrawLine` (a value of –1)Redraw the entire line.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptSysFond`

The preferred system font, the font family ID of the system font preferred for this script (2 bytes). In the Roman script system, this variable specifies Chicago font, whose font family ID is 0 if Roman is the system script. The preferred system font in the Japanese script system is 16384, the font family ID for Osaka.

This variable holds similar information to the variable accessed through the `smScriptSysFondSize` selector. However, changing the value of this variable has no effect on the value accessed through `smScriptSysFondSize`.

Remember that in all localized versions of system software the special value of 0 is remapped to the system font ID. Thus, if an application running under Japanese system software specifies a font family ID of 0 in a function or in the `txFont` field of the current graphics port, Osaka will be used. However, the variable accessed by `smScriptSysFond` will still show the true ID for Osaka (16384).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptAppFond`

The preferred application font (2 bytes); the font family ID of the application font preferred for this script. In the Roman script system, the value of this variable is the font family ID for Geneva.

This variable holds similar information to the variable accessed through the `smScriptAppFondSize` selector. However, changing the value of this variable has no effect on the value accessed through `smScriptAppFondSize`.

Remember that in all localized versions of system software the special value of 1 is remapped to the application font ID. For example, if an application running under Arabic system software specifies a font family ID of 1 in a function, Nadeem will be used. However, the variable accessed by `smScriptSysFond` will still show the true ID for Nadeem (17926).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptBundle`

The beginning of `itlb` values.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptNumber`

The resource ID of the script's numeric-format (`'itl0'`) resource (2 bytes). The numeric-format resource includes formatting information for the correct display of numbers, times, and short dates. The value of this variable is initialized from the script system's international bundle resource.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptDate`

The resource ID of the script's long-date-format (`'itl1'`) resource (2 bytes). The long-date-format resource includes formatting information for the correct display of long dates (dates that include month or day names). The value of this variable is initialized from the script system's international bundle resource.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptSort`

The resource ID of the script's string-manipulation (`'itl2'`) resource (2 bytes). The string-manipulation resource contains functions for sorting and tables for word selection, line breaks, character types, and case conversion of text. The value of this variable is initialized from the script system's international bundle resource.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Script.h`.

`smScriptFlags`
: The script flags word, which contains bit flags specifying attributes of the script (2 bytes). The value of this variable is initialized from the script system's international bundle resource. The "Language Codes A" (page 54) constants are available for examining attributes in the script flags word. Bits above 8 are nonstatic, meaning that they may change during program execution. (Note that the constant values represent bit numbers in the flags word, not masks.)

  The `smsfIntellCP` flag is set if this script system uses spaces as word delimiters. In such a script system it is possible to implement intelligent cut and paste, in which extra spaces are removed when a word is cut from text, and any needed spaces are added when a word is pasted into text. Macintosh Human Interface Guidelines recommends that you implement intelligent cut and paste in script systems that support it.

  If you use the `CharToPixel` function to determine text widths, such as for line breaking, you need to clear the `smsfReverse` bit first.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptToken`
: The resource ID of the script's tokens (`'itl4'`) resource (2 bytes). The tokens resource contains information for tokenizing and number formatting. The value of this variable is initialized from the script system's international bundle resource.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptEncoding`
: The resource ID of the script's (optional) encoding/rendering (`'itl5'`) resource (2 bytes)For single-byte scripts, the encoding/rendering resource specifies text-rendering behavior for double-byte scripts, it specifies character-encoding information. The value of this variable is taken from the script system's international bundle resource.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

`smScriptLang`
: The language code for this version of the script. A language is a specialized variation of a specific script system (2 bytes). Constants for all defined language codes are listed in "Language Codes A" (page 54). The value of this variable is initialized from the script system's international bundle resource.

  Available in Mac OS X v10.0 and later.

  Not available to 64-bit applications.

  Declared in `Script.h`.

Discussion

This section lists and describes the selector constants for accessing script variables through calls to the `GetScriptManagerVariable` (page 110) and `SetScriptManagerVariable` (page 113) functions. In every case the variable parameter passed to or from the function is a long integer (4 bytes); the number in parentheses indicates how many of the 4 bytes are necessary to hold the input or return value for that variable. If fewer than 4 bytes are needed, the low byte or low word contains the information.

In many cases the value of a script variable is taken from the script system's international bundle (`'itlb'`) resource.

Script Token Types

Specify script token types.

```
enum {
    tokenIntl = 4,
    tokenEmpty = -1
};
```

Constants

`tokenIntl`

The `'itl'` resource number of the tokenizer.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenEmpty`

Represents an empty flag.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Source Masks

Specify general transliterate text source masks.

```
enum {
    smMaskAll = 0xFFFFFFFF,
    smMaskAscii = 0x00000001,
    smMaskNative = 0x00000002,
    smMaskAscii1 = 0x00000004,
    smMaskAscii2 = 0x00000008,
    smMaskKana1 = 0x00000010,
    smMaskKana2 = 0x00000020,
    smMaskGana2 = 0x00000080,
    smMaskHangul2 = 0x00000100,
    smMaskJamo2 = 0x00000200,
    smMaskBopomofo2 = 0x00000400
};
```

Table Selectors

Specify selectors for the international table

```
enum {
    smWordSelectTable = 0,
    smWordWrapTable = 1,
    smNumberPartsTable = 2,
    smUnTokenTable = 3,
    smWhiteSpaceList = 4,
    iuWordSelectTable = 0,
    iuWordWrapTable = 1,
    iuNumberPartsTable = 2,
    iuUnTokenTable = 3,
    iuWhiteSpaceList = 4
};
```

Constants

`smWordSelectTable`

Specifies to get the word select break table from `'itl2'`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smWordWrapTable`

Specifies to get the word wrap break table from `'itl2'` .

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smNumberPartsTable`

Specifies to get the default number parts table from `'itl4'`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smUnTokenTable`

Specifies to get the `unToken` table from `'itl4'` .

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`smWhiteSpaceList`

Specifies to get the white space list from `'itl4'` .

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`iuWordSelectTable`

Obsolete; specifies to get the word select break table from `'itl2'` .

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`iuWordWrapTable`

Obsolete; specifies to get the word wrap break table from `'itl2'` .

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`iuNumberPartsTable`

Obsolete; specifies to get the default number parts table from `''itl4'`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`iuUnTokenTable`
: Obsolete; specifies to get the `unToken` table from `'itl4'`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`iuWhiteSpaceList`
: Obsolete; specifies to get the white space list from `'itl4'`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Discussion

These constants can be used as the value of the `tableCode` variable, passed as a parameter to the `GetIntlResourceTable` (page 101) function.

Transliteration Target Types 1

Specify transliterate text target types for Roman or for double-byte scripts

```
enum {
    smTransAscii = 0,
    smTransNative = 1,
    smTransCase = 0xFE,
    smTransSystem = 0xFF,
    smTransAscii1 = 2,
    smTransAscii2 = 3,
    smTransKana1 = 4,
    smTransKana2 = 5
};
```

Constants

`smTransAscii`
: Specifies to convert to ASCII.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smTransNative`
: Specifies to convert to the font script.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smTransCase`
: Specifies to convert case for all text.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smTransSystem`
: Specifies to convert to the system script.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smTransAscii1`
: Specifies to single-byte Roman.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smTransAscii2`
: Specifies to double-byte Roman.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smTransKana1`
: Specifies to single-byte Japanese Katakana.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smTransKana2`
: Specifies to double-byte Japanese Katakana.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Transliteration Target Types 2

Specify transliteration targets for double-byte script systems.

```
enum {
    smTransGana2 = 7,
    smTransHangul2 = 8,
    smTransJamo2 = 9,
    smTransBopomofo2 = 10,
    smTransLower = 0x4000,
    smTransUpper = 0x8000,
    smTransRuleBaseFormat = 1,
    smTransHangulFormat = 2,
    smTransPreDoubleByting = 1,
    smTransPreLowerCasing = 2
};
```

Constants

`smTransGana2`
: Specifies double-byte Japanese Hiragana (no single-byte Hiragana).

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smTransHangul2`
: Specfies double-byte Korean Hangul.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smTransJamo2`
: Specifies double-byte Korean Jamo.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smTransBopomofo2`
: Specifies double-byte Chinese Bopomofo.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smTransLower`
: Specifies target becomes lowercase.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smTransUpper`
: Specifies target becomes uppercase .

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smTransRuleBaseFormat`
: Specifies rule-based `trsl` resource format.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smTransHangulFormat`
: Specifies table-based Hangul `trsl` resource format.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smTransPreDoubleByting`
: Specifies to convert all text to double byte before transliteration.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`smTransPreLowerCasing`
: Specifies to convert all text to lower case before transliteration.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Language Codes

Language Codes A

Specify language codes (values 0 though 23).

```
enum {
    langEnglish = 0,
    langFrench = 1,
    langGerman = 2,
    langItalian = 3,
    langDutch = 4,
    langSwedish = 5,
    langSpanish = 6,
    langDanish = 7,
    langPortuguese = 8,
    langNorwegian = 9,
    langHebrew = 10,
    langJapanese = 11,
    langArabic = 12,
    langFinnish = 13,
    langGreek = 14,
    langIcelandic = 15,
    langMaltese = 16,
    langTurkish = 17,
    langCroatian = 18,
    langTradChinese = 19,
    langUrdu = 20,
    langHindi = 21,
    langThai = 22,
    langKorean = 23
};
```

Constants

`langEnglish`

Represents the English language. The associated script code is `smRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langFrench`

Represents the French language. The associated script code is `smRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langGerman`

Represents the German language. The associated script code is `smRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langItalian`

Represents the Italian language. The associated script code is `smRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langDutch`

Represents the Dutch language. The associated script code is `smRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langSwedish`
: Represents the Swedish language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langSpanish`
: Represents the Spanish language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langDanish`
: Represents the Danish language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langPortuguese`
: Represents the Portuguese language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langNorwegian`
: Represents the Norwegian language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langHebrew`
: Represents the Hebrew language. The associated script code is `smHebrew`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langJapanese`
: Represents the Japanese language. The associated script code is `smJapanese`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langArabic`
: Represents the Arabic language. The associated script code is `smArabic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langFinnish`
: Represents the Finnish language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langGreek`
: Represents the Greek language. The associated script code is `smGreek`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langIcelandic`
: Represents the Icelandic language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langMaltese`
: Represents the Maltese language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langTurkish`
: Represents the Turkish language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langCroatian`
: Represents the Croatian language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langTradChinese`
: Represents the Chinese (traditional charararacters) language. The associated script code is `smTradChinese`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langUrdu`
: Represents the Urdu language. The associated script code is `smArabic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langHindi`
: Represents the Hindi language. The associated script code is `smDevanagari`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langThai`
: Represents the Thai language. The associated script code is `smThai`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langKorean`
: Represents the Korean language. The associated script code is `smKorean`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Language Codes B

Specify language codes (values 24 though 46).

```
enum {
    langLithuanian = 24,
    langPolish = 25,
    langHungarian = 26,
    langEstonian = 27,
    langLatvian = 28,
    langSami = 29,
    langFaroese = 30,
    langFarsi = 31,
    langPersian = 31,
    langRussian = 32,
    langSimpChinese = 33,
    langFlemish = 34,
    langIrishGaelic = 35,
    langAlbanian = 36,
    langRomanian = 37,
    langCzech = 38,
    langSlovak = 39,
    langSlovenian = 40,
    langYiddish = 41,
    langSerbian = 42,
    langMacedonian = 43,
    langBulgarian = 44,
    langUkrainian = 45,
    langByelorussian = 46,
    langBelorussian = 46
};
```

Constants

`langLithuanian`

Represents the Lithuanian language. The associated script code is `smEastEurRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langPolish`

Represents the Polish language. The associated script code is `smEastEurRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langHungarian`

Represents the Hungarian language. The associated script code is `smEastEurRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langEstonian`

Represents the Estonian language. The associated script code is `smEastEurRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langLatvian`

Represents the Lettish language. The associated script code is `smEastEurRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langSami`
: Represents the language of the Sami people of northern Scandinavia.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langFaroese`
: Modified `smRoman`/Icelandic script

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langFarsi`
: Represents the Farsi language. The associated script code is `smArabic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langPersian`
: Represents the Farsi language. The associated script code is `smArabic`. This is the same as the language code `langFarsi` .

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langRussian`
: Represents the Russian language. The associated script code is `smCyrillic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langSimpChinese`
: Represents the Chinese (simplified chararacters) language. The associated script code is `smSimpChinese`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langFlemish`
: Represents the Flemish language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langIrishGaelic`
: Represents Irish Gaelic. The associated script code is `smRoman` or modified `smRoman`/Celtic script (without dot above).

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langAlbanian`
: Represents the Albanian language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langRomanian`
: Represents the Romanian language. The associated script code is `smEastEurRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langCzech`
: Represents the Czech language. The associated script code is `smEastEurRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langSlovak`
: Represents the Slovak language. The associated script code is `smEastEurRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langSlovenian`
: Represents the Slovenian language. The associated script code is `smEastEurRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langYiddish`
: Represents the Yiddish language. The associated script code is `smHebrew`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langSerbian`
: Represents the Serbian language. The associated script code is `smCyrillic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langMacedonian`
: Represents the Macedonian language. The associated script code is `smCyrillic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langBulgarian`
: Represents the Bulgarian language. The associated script code is `smCyrillic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langUkrainian`
: Represents the Ukrainian language. The associated script code is `smCyrillic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langByelorussian`
: Represents the Byelorussian language. The associated script code is `smCyrillic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langBelorussian`
: Represents a synonym for `langByelorussian`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Language Codes C

Specify language codes (values 47 though 70).

```
enum {
    langUzbek = 47,
    langKazakh = 48,
    langAzerbaijani = 49,
    langAzerbaijanAr = 50,
    langArmenian = 51,
    langGeorgian = 52,
    langMoldavian = 53,
    langKirghiz = 54,
    langTajiki = 55,
    langTurkmen = 56,
    langMongolian = 57,
    langMongolianCyr = 58,
    langPashto = 59,
    langKurdish = 60,
    langKashmiri = 61,
    langSindhi = 62,
    langTibetan = 63,
    langNepali = 64,
    langSanskrit = 65,
    langMarathi = 66,
    langBengali = 67,
    langAssamese = 68,
    langGujarati = 69,
    langPunjabi = 70
};
```

Constants

`langUzbek`

Represents the Uzbek language. The associated script code is `smCyrillic`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langKazakh`

Represents the Kazakh language. The associated script code is `smCyrillic`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langAzerbaijani`

Represents the Azerbaijani language. The associated script code is `smCyrillic`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langAzerbaijanAr`

Represents the Azerbaijani language. The associated script code is `smArabic`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langArmenian`
: Represents the Armenian language. The associated script code is `smArmenian`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langGeorgian`
: Represents the Georgian language. The associated script code is `smGeorgian`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langMoldavian`
: Represents the Moldovan language. The associated script code is `smCyrillic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langKirghiz`
: Represents the Kirghiz language. The associated script code is `smCyrillic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langTajiki`
: Represents the Tajiki language. The associated script code is `smCyrillic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langTurkmen`
: Represents the Turkmen language. The associated script code is `smCyrillic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langMongolian`
: Represents the Mongolian language. The associated script code is `smMongolian`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langMongolianCyr`
: Represents the Mongolian language. The associated script code is `smCyrillic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langPashto`
: Represents the Pashto language. The associated script code is `smArabic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langKurdish`
: Represents the Kurdish language. The associated script code is `smArabic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langKashmiri`
: Represents the Kashmiri language. The associated script code is `smArabic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langSindhi`
: Represents the Sindhi language. The associated script code is `smExtArabic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langTibetan`
: Represents the Tibetan language. The associated script code is `smTibetan`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langNepali`
: Represents the Nepali language. The associated script code is `smDevanagari`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langSanskrit`
: Represents the Sanskrit language. The associated script code is `smDevanagari`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langMarathi`
: Represents the Marathi language. The associated script code is `smDevanagari`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langBengali`
: Represents the Bengali language. The associated script code is `smBengali`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langAssamese`
: Represents the Assamese language. The associated script code is `smBengali`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langGujarati`
: Represents the Gujarati language. The associated script code is `smGujarati`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langPunjabi`
: Represents the Punjabi language. The associated script code is `smGurmukhi`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Language Codes D

Specify language codes (values 71 though 94).

```
enum {
    langOriya = 71,
    langMalayalam = 72,
    langKannada = 73,
    langTamil = 74,
    langTelugu = 75,
    langSinhalese = 76,
    langBurmese = 77,
    langKhmer = 78,
    langLao = 79,
    langVietnamese = 80,
    langIndonesian = 81,
    langTagalog = 82,
    langMalayRoman = 83,
    langMalayArabic = 84,
    langAmharic = 85,
    langTigrinya = 86,
    langOromo = 87,
    langSomali = 88,
    langSwahili = 89,
    langKinyarwanda = 90,
    langRuanda = 90,
    langRundi = 91,
    langNyanja = 92,
    langChewa = 92,
    langMalagasy = 93,
    langEsperanto = 94
};
```

Constants

`langOriya`

Represents the Oriya language. The associated script code is `smOriya`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langMalayalam`

Represents the Malayalam language. The associated script code is `smMalayalam`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langKannada`

Represents the Kannada language. The associated script code is `smKannada`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langTamil`

Represents the Tamil language. The associated script code is `smTamil`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langTelugu`
: Represents the Telugu language. The associated script code is `smTelugu`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langSinhalese`
: Represents the Sinhalese language. The associated script code is `smSinhalese`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langBurmese`
: Represents the Burmese language. The associated script code is `smBurmese`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langKhmer`
: Represents the Khmer language. The associated script code is `smKhmer`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langLao`
: Represents the Lao language. The associated script code is `smLaotian`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langVietnamese`
: Represents the Vietnamese language. The associated script code is `smVietnamese`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langIndonesian`
: Represents the Indonesian language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langTagalog`
: Represents the Tagalog language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langMalayRoman`
: Represents the Malay language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langMalayArabic`
: Represents the Malay language. The associated script code is `smArabic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langAmharic`
: Represents the Amharic language. The associated script code is `smEthiopic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langTigrinya`
: Represents the Tigrinya language. The associated script code is `smEthiopic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langOromo`
: Represents the Galla language. The associated script code is `smEthiopic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langSomali`
: Represents the Somali language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langSwahili`
: Represents the Swahili language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langKinyarwanda`
: The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langRuanda`
: Represents the Ruanda language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langRundi`
: Represents the Rundi language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langNyanja`
: The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langChewa`
: Represents the Chewa language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langMalagasy`
Represents the Malagasy language. The associated script code is `smRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langEsperanto`
Represents the Esperanto language. The associated script code is `smRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

Language Codes E

Specify lanaguage codes (values 128 though 141).

```
enum {
    langWelsh = 128,
    langBasque = 129,
    langCatalan = 130,
    langLatin = 131,
    langQuechua = 132,
    langGuarani = 133,
    langAymara = 134,
    langTatar = 135,
    langUighur = 136,
    langDzongkha = 137,
    langJavaneseRom = 138,
    langSundaneseRom = 139,
    langGalician = 140,
    langAfrikaans = 141
};
```

Constants

`langWelsh`
Represents the Welsh language. The associated script code is `smRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langBasque`
Represents the Basque language. The associated script code is `smRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langCatalan`
Represents the Catalan language. The associated script code is `smRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langLatin`
Represents the Latin language. The associated script code is `smRoman`.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langQuechua`
: Represents the Quechua language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langGuarani`
: Represents the Guarani language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langAymara`
: Represents the Aymara language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langTatar`
: Represents the Tatar language. The associated script code is `smCyrillic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langUighur`
: Represents the Uighar language. The associated script code is `smArabic`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langDzongkha`
: Represents the Bhutanese language. The associated script code is `smTibetan`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langJavaneseRom`
: Represents the Javanese language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langSundaneseRom`
: Represents the Sundanese language. The associated script code is `smRoman`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langGalician`
: Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langAfrikaans`
: Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Language Codes F

Specify language codes (values 142 through 150).

```
enum {
    langBreton = 142,
    langInuktitut = 143,
    langScottishGaelic = 144,
    langManxGaelic = 145,
    langIrishGaelicScript = 146,
    langTongan = 147,
    langGreekPoly = 148,
    langGreenlandic = 149,
    langAzerbaijanRoman = 150
};
```

Constants

`langBreton`
: The associated script code is `smRoman` or modified `smRoman`/Celtic script

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langInuktitut`
: Inuit script using `smEthiopic` script code

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langScottishGaelic`
: The associated script code is smRoman or modified `smRoman`/Celtic script

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langManxGaelic`
: The associated script code is `smRoman` or modified `smRoman`/Celtic script

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langIrishGaelicScript`
: The associated script code is modified `smRoman`/Gaelic script (using dot above).

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langTongan`
: The associated script code is `smRoman` script

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langGreekPoly`
: The associated script code is `smGreek` script

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langGreenlandic`
: The associated script code is `smRoman` script

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`langAzerbaijanRoman`
: Represents the Azerbaijani language. The associated script code is `Roman` script.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Language Code - Unspecified

Indicates the language is not specified.

```
enum {
    langUnspecified = 32767
};
```

Region Codes

Range Checking Region Code

Specify values for the the minimum and maximum defined region codes.

```
enum {
    minCountry = verUS,
    maxCountry = verGreenland
};
```

Constants

`minCountry`
: The lowest defined region code (for range-checking); currently this is equal to the region code `verUS`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`maxCountry`
: The highest defined region code (for range-checking); currently this is equal to the region code `verThailand`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Region Codes A

Specify codes for a variety of regions (values 0 - 25).

```
enum {
    verUS = 0,
    verFrance = 1,
    verBritain = 2,
    verGermany = 3,
    verItaly = 4,
    verNetherlands = 5,
    verFlemish = 6,
    verSweden = 7,
    verSpain = 8,
    verDenmark = 9,
    verPortugal = 10,
    verFrCanada = 11,
    verNorway = 12,
    verIsrael = 13,
    verJapan = 14,
    verAustralia = 15,
    verArabic = 16,
    verFinland = 17,
    verFrSwiss = 18,
    verGrSwiss = 19,
    verGreece = 20,
    verIceland = 21,
    verMalta = 22,
    verCyprus = 23,
    verTurkey = 24,
    verYugoCroatian = 25
};
```

Constants

`verUS`

Represents the region of the United States.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`verFrance`

Represents the region of France.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`verBritain`

Represents the region of Great Britain.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`verGermany`

Represents the region of Germany.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`verItaly`

Represents the region of Italy.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`verNetherlands`
: Represents the region of the Netherlands.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verFlemish`
: Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verSweden`
: Represents the region of Sweden.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verSpain`
: Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verDenmark`
: Represents the region of Denmark.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verPortugal`
: Represents the region of Portugal.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verFrCanada`
: Represents the French Canadian region.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verNorway`
: Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verIsrael`
: Represents the region of Israel.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verJapan`
: Represents the region of Japan.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verAustralia`
: Represents the region of Australia.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verArabic`
: Represents the Arabic world. This is the same as the region code `verArabia`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verFinland`
: Represents the region of Finland.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verFrSwiss`
: Represents French for the region of Switzerland.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verGrSwiss`
: Represents German for the region of Switzerland.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verGreece`
: Represents the region of Greece.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verIceland`
: Represents the region of Iceland.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verMalta`
: Represents the region of Malta.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verCyprus`
: Represents the region of Cyprus.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verTurkey`
: Represents the region of Turkey.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verYugoCroatian`
: Represents the Croatian system for the region of Yugoslavia.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Discussion

Each region is associated with a particular language code and script code (not shown). The existence of a defined region code does not necessarily imply the existence of a version of Macintosh system software localized for that region.

Region Codes B

Specify region codes (values 26 though 32).

```
enum {
    verNetherlandsComma = 26,
    verBelgiumLuxPoint = 27,
    verCanadaComma = 28,
    verCanadaPoint = 29,
    vervariantPortugal = 30,
    vervariantNorway = 31,
    vervariantDenmark = 32
};
```

Constants

`verNetherlandsComma`

Specifies Dutch.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`verBelgiumLuxPoint`

Specifies Belgium.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`verCanadaComma`

Specifies Canadian ISO.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`verCanadaPoint`

Specifies Canadian; now unused.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`vervariantPortugal`

Unused.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`vervariantNorway`

Unused.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`vervariantDenmark`
: Specifies Danish Mac Plus.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Region Codes C

Specify region codes (values 33 through 61).

```
enum {
    verIndiaHindi = 33,
    verPakistanUrdu = 34,
    verTurkishModified = 35,
    verItalianSwiss = 36,
    verInternational = 37,
    verRomania = 39,
    verGreecePoly = 40,
    verLithuania = 41,
    verPoland = 42,
    verHungary = 43,
    verEstonia = 44,
    verLatvia = 45,
    verSami = 46,
    verFaroeIsl = 47,
    verIran = 48,
    verRussia = 49,
    verIreland = 50,
    verKorea = 51,
    verChina = 52,
    verTaiwan = 53,
    verThailand = 54,
    verScriptGeneric = 55,
    verCzech = 56,
    verSlovak = 57,
    verFarEastGeneric = 58,
    verMagyar = 59,
    verBengali = 60,
    verByeloRussian = 61
};
```

Constants

`verIndiaHindi`
: The Hindi system for the region of India; hi_IN..

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verPakistanUrdu`
: Urdu for Pakistan; ur_PK.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verTurkishModified`
: Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verItalianSwiss`
: Italian Swiss; it_CH.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verInternational`
: English for international use; Z en.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verRomania`
: Romaniza; ro_RO

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verGreecePoly`
: Polytonic Greek (classical); grc.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verLithuania`
: Lithuania; lt_LT.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verPoland`
: Poland; pl_PL.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verHungary`
: Represents the region of Hungary; hu_HU.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verEstonia`
: Represents the region of Estonia; et_EE.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verLatvia`
: Represents the region of Latvia; lv_LV.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verSami`
: se.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verFaroeIsl`
: fo_FO.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verIran`
: Persian/Farsi Represents the region of Iran; fa_IR .

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verRussia`
: Represents the region of Russia; ru_RU.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verIreland`
: Represents Irish Gaelic for Ireland (without dot above); ga_IE.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verKorea`
: Represents the region of Korea; ko_KR.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verChina`
: Simplified Chinese; zh_CN.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verTaiwan`
: Traditional Chinese; zh_TW.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verThailand`
: Represents the region of Thailand; th_TH.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verScriptGeneric`
: Generic script system (no language or script).

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verCzech`
: cs_CZ.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verSlovak`
: sk_SK.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verFarEastGeneric`
: Generic Far East system (no language or script).

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verMagyar`
: Unused; see `verHungary`.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verBengali`
: Bangladesh or India; bn.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verByeloRussian`
: be_B,

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Region Codes D

Specify region codes (values 62 through 97).

```
enum {
    verUkraine = 62,
    verGreeceAlt = 64,
    verSerbian = 65,
    verSlovenian = 66,
    verMacedonian = 67,
    verCroatia = 68,
    verGermanReformed = 70,
    verBrazil = 71,
    verBulgaria = 72,
    verCatalonia = 73,
    verMultilingual = 74,
    verScottishGaelic = 75,
    verManxGaelic = 76,
    verBreton = 77,
    verNunavut = 78,
    verWelsh = 79,
    verIrishGaelicScript = 81,
    verEngCanada = 82,
    verBhutan = 83,
    verArmenian = 84,
    verGeorgian = 85,
    verSpLatinAmerica = 86,
    verTonga = 88,
    verFrenchUniversal = 91,
    verAustria = 92,
    verGujarati = 94,
    verPunjabi = 95,
    verIndiaUrdu = 96,
    verVietnam = 97
};
```

Regions Codes E

Specify region codes (values 98 through 109).

```
enum {
    verFrBelgium = 98,
    verUzbek = 99,
    verSingapore = 100,
    verNynorsk = 101,
    verAfrikaans = 102,
    verEsperanto = 103,
    verMarathi = 104,
    verTibetan = 105,
    verNepal = 106,
    verGreenland = 107,
    verIrelandEnglish = 108
};
```

Token Constants

Tokens - Mathematical

Specify tokens used in mathematical operations.

```
enum {
    tokenLeftCurly = 20,
    tokenRightCurly = 21,
    tokenLeftEnclose = 22,
    tokenRightEnclose = 23,
    tokenPlus = 24,
    tokenMinus = 25,
    tokenAsterisk = 26,
    tokenDivide = 27,
    tokenPlusMinus = 28,
    tokenSlash = 29,
    tokenBackSlash = 30,
    tokenLess = 31,
    tokenGreat = 32,
    tokenEqual = 33,
    tokenLessEqual2 = 34,
    tokenLessEqual1 = 35,
    tokenGreatEqual2 = 36,
    tokenGreatEqual1 = 37,
    token2Equal = 38,
    tokenColonEqual = 39
};
```

Constants

`tokenLeftCurly`

Represents an opening curly bracket.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenRightCurly`

Represents a closing curly bracket.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenLeftEnclose`
: Represents an opening European double quote.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenRightEnclose`
: Represents a closing European double quote.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenPlus`
: Represents a plus sign.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenMinus`
: Represents a minus sign.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenAsterisk`
: Represents a times/multiply sign.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenDivide`
: Represents a divide.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenPlusMinus`
: Represents a plus-or-minus symbol.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenSlash`
: Represents a slash.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenBackSlash`
: Represents a backslash.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenLess`
: Represents a less than sign.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenGreat`
: Represents a greater than sign.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenEqual`
: Represents an equal.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenLessEqual2`
: Represents a less than or equal to sign (2 symbols).

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenLessEqual1`
: Represents a less than or equal to sign (1 symbol).

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenGreatEqual2`
: Represents a greater than or equal to sign (2 symbols).

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenGreatEqual1`
: Represents a greater-than-or-equal-to sign (1 symbol).

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`token2Equal`
: Represents a double equal sign.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenColonEqual`
: Represents a colon equal sign.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Tokens - Punctuation

Specify tokens for various punctuation marks.

```
enum {
    tokenNotEqual = 40,
    tokenLessGreat = 41,
    tokenExclamEqual = 42,
    tokenExclam = 43,
    tokenTilde = 44,
    tokenComma = 45,
    tokenPeriod = 46,
    tokenLeft2Quote = 47,
    tokenRight2Quote = 48,
    tokenLeft1Quote = 49,
    tokenRight1Quote = 50,
    token2Quote = 51,
    token1Quote = 52,
    tokenSemicolon = 53,
    tokenPercent = 54,
    tokenCaret = 55,
    tokenUnderline = 56,
    tokenAmpersand = 57,
    tokenAtSign = 58,
    tokenBar = 59
};
```

Constants

`tokenNotEqual`

Represents a not equal sign.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenLessGreat`

Represents a less/greater sign (not equal in Pascal).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenExclamEqual`

Represents an exclamation equal sign (not equal in C).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenExclam`

Represents as exclamation point.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenTilde`

Represents a centered tilde.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenComma`

Represents a comma.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenPeriod`
Represents a period.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenLeft2Quote`
Represents an opening double quote.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenRight2Quote`
Represents a closing double quote.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenLeft1Quote`
Represents an opening single quote.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenRight1Quote`
Represents a closing single quote.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`token2Quote`
Represents a double quote.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`token1Quote`
Represents a single quote.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenSemicolon`
Represents a semicolon.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenPercent`
Represents a percent sign.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenCaret`
Represents a caret.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenUnderline`
: Represents an underline.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenAmpersand`
: Represents an ampersand.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenAtSign`
: Represents an at sign.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenBar`
: Represents a vertical bar.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Tokens for Symbols

Specify tokens for various symbols.

```
enum {
    tokenQuestion = 60,
    tokenPi = 61,
    tokenRoot = 62,
    tokenSigma = 63,
    tokenIntegral = 64,
    tokenMicro = 65,
    tokenCapPi = 66,
    tokenInfinity = 67,
    tokenColon = 68,
    tokenHash = 69,
    tokenDollar = 70,
    tokenNoBreakSpace = 71,
    tokenFraction = 72,
    tokenIntlCurrency = 73,
    tokenLeftSingGuillemet = 74,
    tokenRightSingGuillemet = 75,
    tokenPerThousand = 76,
    tokenEllipsis = 77,
    tokenCenterDot = 78,
    tokenNil = 127
};
```

Constants

`tokenQuestion`
: Represents a question mark.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenPi`
: Represents a Pi token.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenRoot`
: Represents a square root sign.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenSigma`
: Represents a capital sigma.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenIntegral`
: Represents an integral sign.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenMicro`
: Represents a micro.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenCapPi`
: Represents a capital pi.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenInfinity`
: Represents an infinity sign.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenColon`
: Represents a colon.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenHash`
: Represents a pound sign (U.S. weight).

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenDollar`
: Represents a dollar sign.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenNoBreakSpace`
: Represents a nonbreaking space.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenFraction`
: Represents a fraction.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenIntlCurrency`
: Represents an international currency token.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenLeftSingGuillemet`
: Represents an opening single guillemet.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenRightSingGuillemet`
: Represents a closing single guillemet.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenPerThousand`
: Represents a per thousands token.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenEllipsis`
: Represents an ellipsis character.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenCenterDot`
: Represents a center dot.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenNil`
: Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Token Types

Specify types of tokens.

```
enum {
    tokenUnknown = 0,
    tokenWhite = 1,
    tokenLeftLit = 2,
    tokenRightLit = 3,
    tokenAlpha = 4,
    tokenNumeric = 5,
    tokenNewLine = 6,
    tokenLeftComment = 7,
    tokenRightComment = 8,
    tokenLiteral = 9,
    tokenEscape = 10,
    tokenAltNum = 11,
    tokenRealNum = 12,
    tokenAltReal = 13,
    tokenReserve1 = 14,
    tokenReserve2 = 15,
    tokenLeftParen = 16,
    tokenRightParen = 17,
    tokenLeftBracket = 18,
    tokenRightBracket = 19
};
```

Constants

`tokenUnknown`

Has no existing token type.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenWhite`

Represents a whitespace character.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenLeftLit`

Represents an opening literal marker.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenRightLit`

Represents a closing literal marker.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenAlpha`

Represents an alphabetic token.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenNumeric`

Represents a numeric token.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenNewLine`
Represents a new line.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenLeftComment`
Represents an opening comment marker.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenRightComment`
Represents a closing comment marker.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenLiteral`
Represents a literal token.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenEscape`
Represents an escape character.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenAltNum`
Represents an alternate number (such as at $B0–$B9).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenRealNum`
Represents a real number.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenAltReal`
Represents an alternate real number.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenReserve1`
Reserved.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenReserve2`
Reserved.

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`tokenLeftParen`
: Represents an opening parenthesis.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenRightParen`
: Represents a closing parenthesis.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenLeftBracket`
: Represents an opening square bracket.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenRightBracket`
: Represents a closing square bracket.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Token Results

Specify token conditions returned by the function `IntlTokenize`.

```
enum {
    tokenOK = 0,
    tokenOverflow = 1,
    stringOverflow = 2,
    badDelim = 3,
    badEnding = 4,
    crash = 5
};
typedef SInt8 TokenResults;
```

Constants

`tokenOK`
: Indicates the function exectured without error.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`tokenOverflow`
: Indicates a token overflow.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`stringOverflow`
: Indicates a string overflow.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`badDelim`
: Indicates a bad delimiter,

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`badEnding`
: Indicates a bad ending.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`crash`
: Indicates a crash.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

Discussion

Token results are returned by the function `IntlTokenize` (page 104).

Obsolete Constants

Obsolete Language Codes

Specify obsolete language codes provided for backward compatibility.

```
enum {
    langPortugese = 8,
    langMalta = 16,
    langYugoslavian = 18,
    langChinese = 19,
    langLettish = 28,
    langLapponian = 29,
    langLappish = 29,
    langSaamisk = 29,
    langFaeroese = 30,
    langIrish = 35,
    langGalla = 87,
    langAfricaans = 141
};
```

Discussion

These are obsolete language code names kept for backward compatibility. They have one or more of the following problems: misspelled, ambiguous, misleading, archaic, inappropriate.

Obsolete Regions Codes

Specfiy obsolete region code names provided for backward compatibility.

```
enum {
    verFrBelgiumLux = 6,
    verBelgiumLux = 6,
    verArabia = 16,
    verYugoslavia = 25,
    verIndia = 33,
    verPakistan = 34,
    verRumania = 39,
    verGreekAncient = 40,
    verLapland = 46,
    verFaeroeIsl = 47,
    verGenericFE = 58,
    verBelarus = 61,
    verUkrania = 62,
    verAlternateGr = 64,
    verSerbia = 65,
    verSlovenia = 66,
    verMacedonia = 67,
    verBrittany = 77,
    verWales = 79,
    verArmenia = 84,
    verGeorgia = 85,
    verAustriaGerman = 92,
    verTibet = 105
};
```

Discussion
Obsolete region code names (kept for backward compatibility): Misspelled or alternate form, ambiguous, misleading, considered pejorative, archaic, etc.

Obsolete Roman Script Constants

Specify obsolete constants provided for backward compatibility.

```
enum {
    romanSysFond = 0x3FFF,
    romanAppFond = 3,
    romanFlags = 0x0007,
    smFondStart = 0x4000,
    smFondEnd = 0xC000,
    smUprHalfCharSet = 0x80
};
```

Discussion
You should use the function `GetScriptVariable` (page 111) to obtain the information specified by these constants.

Obsolete Script Codes

Specify obsolete script code names provided for backward compatibility.

```
enum {
    smChinese = 2,
    smRussian = 7,
    smLaotian = 22,
    smAmharic = 28,
    smSlavic = 29,
    smEastEurRoman = 29,
    smSindhi = 31,
    smKlingon = 32
};
```

Obsolete System Script Codes

Specify obsolete script code values for International Utilities provided for backward compatibility.

```
enum {
    iuSystemScript = -1,
    iuCurrentScript = -2
};
```

Obsolete Token Codes

Specify obsolete token names provided for backward compatibility.

```
enum {
    delimPad = -2,
    tokenTilda = 44,
    tokenCarat = 55
};
```

Result Codes

The most common result codes returned by Script Manager are listed below.

Result Code	Value	Description
`smNotInstalled`	0	Routine not available in script. Available in Mac OS X v10.0 and later.
`smBadVerb`	-1	Bad verb passed to a routine. Available in Mac OS X v10.0 and later.
`smBadScript`	-2	Bad script code passed to a routine. Available in Mac OS X v10.0 and later.

Deprecated Script Manager Reference (Not Recommended) Functions

A function identified as deprecated has been superseded and may become unsupported in the future.

Deprecated in Mac OS X v10.4

CharacterByteType

Identifies a byte in a text buffer as a single-byte character or as the first or second byte of a double-byte character. (Deprecated in Mac OS X v10.4. You should update your application to handle Unicode text. There is no replacement function because Unicode handles encoding in a different manner.)

```
short CharacterByteType (
   Ptr textBuf,
   short textOffset,
   ScriptCode script
);
```

Parameters

textBuf

A pointer to a text buffer containing the byte to be identified.

textOffset

The offset to the byte to be identified. The offset is measured in bytes; the first byte has an offset of 0.

script

A value that specifies the script system of the text in the buffer. Constants for all defined script codes are listed on "Meta Script Codes" (page 25). To specify the font script, pass `smCurrentScript` in this parameter.

Return Value

One of three identifications: a single-byte character, the first byte of a double-byte character, or the second byte of a double-byte character. The first byte of a double-byte character—the one at the lower offset in memory—is the high-order byte the second byte of a double-byte character—the one at the higher offset—is the low-order byte. This is the same order in which text is processed and numbers are represented.

Discussion

The script system associated with the character you wish to examine must be enabled in order for the function to provide useful information. For example, if only the Roman script system is available and you attempt to identify a byte in a run of double-byte characters, the `CharacterByteType` function returns 0, indicating that the byte is a single-byte character.

For single-byte script systems, the character-type tables reside in the string-manipulation (`'itl2'`) resource and reflect region-specific or language-specific differences in uppercase conventions.

For double-byte script systems, the character-type tables reside in the encoding/rendering (`'itl5'`) resource, not the string-manipulation resource. Whenever you call `CharacterByteType`, the necessary character-set encoding information is taken from the encoding/rendering resource. You cannot use the `GetIntlResource` function to access double-byte character-type tables directly.

From byte value alone, it is not possible to distinguish the second byte of a double-byte character from a single-byte character. `CharacterByteType` differentiates the second byte of a double-byte character from a single-byte character by assuming that the byte at offset 0 is the first byte of a character. With that assumption, it then sequentially identifies the size and starting position of each character in the buffer up to `textOffset`.

Special Considerations

If you specify `smCurrentScript` for the `script` parameter, the value returned by `CharacterByteType` can be affected by the state of the font force flag. It is unaffected by the state of the international resources selection flag.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Script.h`

CharacterType

Returns a variety of information about the character represented by a given byte, including its type, class, orientation, direction, case, and size (in bytes). (Deprecated in Mac OS X v10.4. You should update your application to handle Unicode text. There is no replacement function because Unicode handles encoding in a different manner.)

```
short CharacterType (
   Ptr textBuf,
   short textOffset,
   ScriptCode script
);
```

Parameters

`textBuf`
: A pointer to a text buffer containing the character to be examined.

`textOffset`
: The offset to the location of the character to be examined. (It can be an offset to either the first or the second byte of a double-byte character.) Offset is in bytes; the first byte of the first character has an offset of 0.

`script`
: A value that specifies the script system the byte belongs to. Constants for all defined script codes are listed in "Meta Script Codes" (page 25). To specify the font script, pass `smCurrentScript` in this parameter.

Return Value

An integer bit field that provides information about the requested character.

Discussion

The `CharacterType` return value is an integer bit field that provides information about the requested character. The field has the following format:

- Bit range 0–3 (Type). The character types.
- Bit range 4–7. Reserved.
- Bit Range 8–11 (Class). Character classes (i.e., subtypes).
- Bit 12 (Orientation). Horizontal or vertical.
- Bit 13 (Direction). Left or right. In double-byte script systems, bit 13 indicates whether or not the character is part of the main character set (not a user-defined character).
- Bit 14 (Case). Uppercase or lowercase.
- Bit 15 (Size). single-byte or double-byte.

The script system associated with the character you wish to examine must be enabled in order for any of these three functions to provide useful information.

For single-byte script systems, the character-type tables reside in the string-manipulation (`'itl2'`) resource and reflect region-specific or language-specific differences in uppercase conventions. The `CharacterType` function gets the tables from the string-manipulation resource using the `GetIntlResource` function.

For double-byte script systems, the character-type tables reside in the encoding/rendering (`'itl5'`) resource, not the string-manipulation resource. Whenever you call `CharacterType`, the necessary character-set encoding information is taken from the encoding/rendering resource. You cannot use the `GetIntlResource` function to access double-byte character-type tables directly.

The Script Manager defines the recognized character types, character classes, and character modifiers (bits 12–15), with constants to describe them. The `CharacterType` field masks are described in "Character Type Field Masks" (page 18).

The Script Manager also defines a set of masks with which you can isolate each of the fields in the `CharacterType` return value. If you perform an `AND` operation with the `CharacterType` result and the mask for a particular field, you select only the bits in that field. Once you've done that, you can test the result, using the constants that represent the possible results.

The function `CharacterType` calls `CharacterByteType` to determine whether the byte at `textOffset` is a single-byte character or the first byte or second byte of a double-byte character. The larger the text buffer, the longer `CharacterByteType` takes to execute. To be most efficient, place the pointer `textBuf` at the beginning of the character of interest before calling `CharacterType`. (If you want to be compatible with older versions of `CharacterType`, also set `textOffset` to 1, rather than 0, for double-byte characters.)

Special Considerations

The function `CharacterType` may move memory; your application should not call this function at interrupt time.

If you specify `smCurrentScript` for the `script` parameter, `CharacterType` always assumes that the text in the buffer belongs to the font script. It is unaffected by the state of the font force flag or the international resources selection flag.

For single-byte script systems, the character-type tables are in the string-manipulation (`'itl2'`) resource. For double-byte script systems, they are in the encoding/rendering (`'itl5'`) resource. If the appropriate resource does not include these tables, `CharacterType` exits without doing anything.

Some Roman fonts (for example, Symbol) substitute other characters for the standard characters in the Standard Roman character set. Since the Roman script system `CharacterType` function assumes the Standard Roman character set, it may return inappropriate results for nonstandard characters.

Version Notes
In versions of system software earlier than 7.0, the `textOffset` parameter to the `CharacterType` function must point to the second byte of a double-byte character.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Script.h`

ClearIntlResourceCache

Clears the application's international resources cache, which contains the resource ID numbers of the string-manipulation (`'itl2'`) and tokens (`'itl4'`) resources for the current script. (Deprecated in Mac OS X v10.4. There is no replacement because this function is no longer needed in Mac OS X.)

Not recommended

```
void ClearIntlResourceCache (
   void
);
```

Discussion
At application launch, the script management system sets up an international resources cache for the application. The cache contains the resource ID numbers of the string-manipulation and tokens resources for all enabled scripts.

If you provide your own string manipulation or tokens resource to replace the default for a particular script, call `ClearIntlResourceCache` at launch to ensure that your supplied resource is used instead of the script system's `'itl2'` or `'itl4'` resource.

The current default ID numbers for a script system's `'itl2'` and `'itl4'` resources are stored in its script variables. You can read and modify these values with the `GetScriptVariable` and `SetScriptVariable` functions using the selectors `smScriptSort` (for the `'itl2'` resource) and `smScriptToken` (for the `'itl4'` resource). Before calling `ClearIntlResourceCache`, you should set the script's default ID number to the ID of the resource that you are supplying.

If the international resources selection flag is `TRUE`, the ID numbers of your supplied resources must be in the system script range. Otherwise, the IDs must be in the range of the current script.

If you use the `SetScriptVariable` function to change the value of the `'itl2'` or `'itl4'` resource ID and then call `ClearIntlResourceCache` to flush the cache, be sure to restore the original resource ID before your application quits.

Special Considerations

The function `ClearIntlResourceCache` may move memory; your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Script.h`

FillParseTable

Helps your application to quickly process a buffer of mixed single-byte and double-byte characters. (Deprecated in Mac OS X v10.4. You should update your application to handle Unicode text. There is no replacement function because Unicode handles encoding in a different manner.)

```
Boolean FillParseTable (
   CharByteTable table,
   ScriptCode script
);
```

Parameters

`table`
A 256-byte table to be filled in by `FillParseTable`.

`script`
A value that specifies the script system the parse table belongs to. Constants for all defined script codes are listed in "Meta Script Codes" (page 25). To specify the font script, pass `smCurrentScript` in this parameter.

Return Value
If you specify `smCurrentScript` for the `script` parameter, the value returned by `FillParseTable` can be affected by the state of the font force flag. It is unaffected by the international resources selection flag.

Discussion
Before calling `FillParseTable`, allocate space for a 256-byte table to pass to the function in the table parameter.

This function returns a 256-byte table that distinguishes the character codes of all possible single-byte characters from the first (high-order) byte values of all possible double-byte characters in the specified script system. The script system associated with the character you wish to examine must be enabled in order for any of these three functions to provide useful information.

For single-byte script systems, the character-type tables reside in the string-manipulation (`'itl2'`) resource and reflect region-specific or language-specific differences in uppercase conventions.

For double-byte script systems, the character-type tables reside in the encoding/rendering (`'itl5'`) resource, not the string-manipulation resource. Whenever you call `FillParseTable`, the necessary character-set encoding information is taken from the encoding/rendering resource. You cannot use the `GetIntlResource` function to access double-byte character-type tables directly.In every script system, double-byte characters have distinctive high-order (first) bytes that allow them to be distinguished from single-byte characters. `FillParseTable` fills a 256-byte table, conceptually equivalent to a single-byte character-set table, with values that indicate, byte-for-byte, whether the character-code value represented by that byte index is the first byte of a double-byte character. An entry in the `CharByteTable` is 0 for a single-byte character and 1 for the first byte of a double-byte character.

If your application is processing mixed characters, it can use the table to identify the locations of the double-byte characters as it makes a single pass through the text, rather than having to call `CharacterByteType` or `CharacterType` for each byte of the text buffer in turn. `CharacterByteType` and `CharacterType` start anew at the beginning of the text buffer each time they are called, tracking character positions up to the offset of the byte to be analyzed.

Special Considerations

`FillParseTable` may move memory; your application should not call this function at interrupt time.

The table defined by `CharByteTable` is not dynamic; it does not get updated when the current font changes. You need to call it separately for each script run in your text.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Script.h`

FontScript

Returns the script code for the current script (usually the font script). (Deprecated in Mac OS X v10.4. Use `ATSFontFamilyGetEncoding` instead.)

```
short FontScript (
   void
);
```

Parameters

Return Value

A script code. All recognized script codes and their defined constants are listed in "Meta Script Codes" (page 25). `FontScript` returns only explicit script codes (□0). If the font of the active graphics port is Roman and the font force flag is `TRUE`, the script code returned is that of the system script and the script-forced result flag is set to `TRUE`. If the font of the active graphics port is non-Roman, the state of the font force flag is ignored. If the script system corresponding to the font of the active graphics port is not installed and enabled, the script code returned is that of the system script and the script-defaulted result flag is set to `TRUE`.

Discussion

The information about the script code is subject to two control flags—the font force flag and the international resources selection flag. You can test and set these flags with the `GetScriptManagerVariable` and `SetScriptManagerVariable` selectors `smFontForce` and `smIntlForce`.

The function starts by initializing two result flags, the script-forced result flag and the script-defaulted result flag, to `FALSE`. These flags are Script Manager variables, accessed through the `GetScriptManagerVariable` function selectors `smForced` and `smDefault`.

Special Considerations

`FontScript` may move memory; your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Script.h`

FontToScript

Translates a font family ID number into its corresponding script code, if that script system is currently enabled. (Deprecated in Mac OS X v10.4. Use `ATSFontFamilyGetEncoding` instead.)

```
short FontToScript (
   short fontNumber
);
```

Parameters

`fontNumber`
A font family ID number.

Return Value
A script code. All recognized script codes and their defined constants are listed in "Meta Script Codes" (page 25). `FontToScript` returns only explicit script codes (□0). If `fontNumber` is in the Roman range and the font force flag is `TRUE`, the script code returned is that of the system script and the script-forced result flag is set to `TRUE`. If `fontNumber` is in the non-Roman range, the state of the font force flag is ignored. If the script system corresponding to `fontNumber` is not enabled, the script code returned is that of the system script and the script-defaulted result flag is set to `TRUE`.

Discussion
The information about the script code is subject to two control flags—the font force flag and the international resources selection flag. You can test and set these flags with the `GetScriptManagerVariable` and `SetScriptManagerVariable` selectors `smFontForce` and `smIntlForce`.

The function starts by initializing two result flags, the script-forced result flag and the script-defaulted result flag, to `FALSE`. These flags are Script Manager variables, accessed through the `GetScriptManagerVariable` function selectors `smForced` and `smDefault`.

Do not use the function `FontToScript` to convert resource IDs to scripts codes.

Special Considerations

`FontToScript` may move memory; your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Script.h`

GetIntlResourceTable

Obtains a specific word-selection, line-break, number-parts, untoken, or whitespace table from the appropriate international resource. (Deprecated in Mac OS X v10.4. There is no replacement because this function is no longer needed in Mac OS X.)

```
void GetIntlResourceTable (
   ScriptCode script,
   short tableCode,
   Handle *itlHandle,
   long *offset,
   long *length
);
```

Parameters

script
: A script code, the value that specifies a particular script system. Constants for all defined script codes are listed in "Meta Script Codes" (page 25).

tableCode
: A number that specifies which table is requested. The constants for `tableCode` are detailed in "Table Selectors" (page 50).

itlHandle
: On return, a handle to the string-manipulation (`'itl2'`) or tokens (`'itl4'`) resource containing the table specified in the `tableCode` parameter. If the script system whose table is requested is not available, `GetIntlResourceTable` returns a `NULL` handle.

offset
: On return, a pointer to the offset (in bytes) to the specified table from the beginning of the resource.

length
: On return, a pointer to the size of the table (in bytes).

Discussion

When you provide a script code in the `script` parameter, and a table code in the `tableCode` parameter, `GetIntlResourceTable` returns a handle to the string-manipulation resource or tokens resource containing that table, the offset of the specified table from the beginning of the resource, and the length of the table.

If you wish to manipulate the contents of the table you have requested, use the size returned in the `length` parameter to allocate a buffer, and perform a block move of the table's contents into that buffer.

Special Considerations

`GetIntlResourceTable` may move memory; your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Script.h`

GetSysDirection

Returns the current value of `SysDirection`, the global variable that determines the system direction (primary line direction). (Deprecated in Mac OS X v10.4. This function does not return anything useful in Mac OS X.)

```
short GetSysDirection (
   void
);
```

Parameters

Return Value

The current value of `SysDirection`: 0 if the system direction is left-to-right; -1 ($FFFF) if the system direction is right-to-left.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Script.h`

IntlScript

Identifies the script system used by the Text Utilities date-formatting, time-formatting, and string-sorting functions. (Deprecated in Mac OS X v10.4. Use `ATSFontFamilyGetEncoding` instead.)

```
short IntlScript (
   void
);
```

Parameters

Return Value

A script code. All recognized script codes and their defined constants are listed in "Meta Script Codes" (page 25). `IntlScript` returns only explicit script codes (□0). If the international resources selection flag is `TRUE`, the script code returned is that of the system script. If the identified script system is not enabled, the script code returned is that of the system script and the script-defaulted result flag is set to `TRUE`.

Discussion

Information about the script system is subject to two control flags—the font force flag and the international resources selection flag. You can test and set these flags with the `GetScriptManagerVariable` and `SetScriptManagerVariable` selectors `smFontForce` and `smIntlForce`.

The function starts by initializing two result flags, the script-forced result flag and the script-defaulted result flag, to `FALSE`. These flags are Script Manager variables, accessed through the `GetScriptManagerVariable` function selectors `smForced` and `smDefault`.

The function also identifies the script system whose resources are returned by the Script Manager function `GetIntlResource`. It is either the font script—the script system corresponding to the current font of the active graphics port—or the system script.

Special Considerations

`IntlScript` may move memory; your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Script.h`

IntlTokenize

Allows your application to convert text into a sequence of language-independent tokens. (Deprecated in Mac OS X v10.4. There is no replacement because this function is no longer needed in Mac OS X.)

```
TokenResults IntlTokenize (
   TokenBlockPtr tokenParam
);
```

Parameters

`tokenParam`
: A pointer to a token block structure. The structure specifies the text to be converted to tokens, the destination of the token list, a handle to the tokens (`'itl4'`) resource, and a set of options. See the `TokenBlock` (page 9)data structure for information on what you need to pass in this structure and what you obtain on return.

Return Value
A `TokenResults` value that specifies whether the function executed with errors. See "Token Results" (page 90) for a list of the values that can be returned.

Discussion
Before calling the `IntlTokenize` function, allocate memory for and set up the following data structures:

- A token block structure (data type `TokenBlock`). The token block structure is a parameter block that holds both input and output parameters for the `IntlTokenize` function.
- A token list to hold the results of the tokenizing operation. To set up the token list, estimate how many tokens will be generated from your text, multiply that by the size of a token structure, and allocate a memory block of that size in bytes. An upper limit to the possible number of tokens is the number of characters in the source text.
- A string list, if you want the `IntlTokenize` function to generate character strings for all the tokens. To set up the string list, multiply the estimated number of tokens by the expected average size of a string, and allocate a memory block of that size in bytes. An upper limit is twice the number of tokens plus the number of bytes in the source text.

The function `IntlTokenize` creates tokens based on information in the tokens (`'itl4'`) resource of the script system under which the source text was created. You must load the tokens resource and place its handle in the token block structure before calling the `IntlTokenize` function.

The token block structure contains both input and output values. At input, you must provide values for the fields that specify the source text location, the token list location, the size of the token list, the tokens (`'itl4'`) resource to use, and several options that affect the operation. You must set reserved locations to 0 before calling `IntlTokenize`.

On output, the token block structure specifies how many tokens have been generated and the size of the string list (if you have selected the option to generate strings).

The results of the tokenizing operation are contained in the token list, an array of token structures (data type `TokenRec` (page 11)).

Pascal strings are generated if the `doString` parameter in the token block structure is set to `TRUE`. The string is a normalized version of the source text that generated the token; alternate digits are replaced with ASCII numerals, the decimal point is always an ASCII period, and double-byte Roman letters are replaced with low-ASCII equivalents.

To make a series of calls to `IntlTokenize` and append the results of each call to the results of previous calls, set `doAppend` to `FALSE` and initialize `tokenCount` and `stringCount` to 0 before making the first call to `IntlTokenize`. (You can ignore `stringCount` if you set `doString` to `FALSE`.) Upon completion of the call, `tokenCount` and `stringCount` will contain the number of tokens and the length in bytes of the string list, respectively, generated by the call. On subsequent calls, set `doAppend` to `TRUE`, reset the `source` and `sourceLength` parameters (and any other parameters as appropriate) for the new source text, but maintain the output values for `tokenCount` and `stringCount` from each call as input values to the next call. At the end of your sequence of calls, the token list and string list will contain, in order, all the tokens and strings generated from the calls to `IntlTokenize`.

If you are making tokens from text that was created under more than one script system, you must load the proper tokens resource and place its handle in the token block structure separately for each script run in the text, appending the results each time.

Delimiters for quoted literals are passed to `IntlTokenize` in a two-integer array.

The individual delimiters, as specified in the `leftDelims` and `rightDelims` parameters, are paired by position. The first (in storage order) opening delimiter in `leftDelims` is paired with the first closing delimiter in `rightDelims`.

Comment delimiters may be 1 or 2 tokens each and there may be two sets of opening and closing pairs. They are passed to `IntlTokenize` in a `commentType` array.

If only one token is needed for a delimiter, the second token must be specified to be `delimPad`. If only one delimiter of an opening-closing pair is needed, then both of the tokens allocated for the other symbol must be `delimPad`. The first token of a two-token sequence is at the higher position in the `leftComment` or `rightComment` array.

When `IntlTokenize` encounters an escape character within a quoted literal, it places the portion of the literal before the escape character into a single token (of type `tokenLiteral`), places the escape character into another token (`tokenEscape`), places the character following the escape character into another token (whatever token type it corresponds to), and places the portion of the literal following the escape sequence into another token (`tokenLiteral`). Outside of a quoted literal, the escape character has no special significance.

`IntlTokenize` considers the character specified in the `decimalCode` parameter to be a decimal character only when it is flanked by numeric or alternate numeric characters, or when it follows them.

Special Considerations

`IntlTokenize` may move memory; your application should not call this function at interrupt time.

Because each call to `IntlTokenize` must be for a single script run, there can be no change of script within a comment or quoted literal.

Comments and quoted literals must be complete within a single call to `IntlTokenize` in order to avoid syntax errors.

`IntlTokenize` always uses the tokens resource whose handle you pass it in the token block structure. Therefore, it is not directly affected by the state of the font force flag or the international resources selection flag. However, if you use the `GetIntlResource` function to get a handle to the tokens resource to pass to `IntlTokenize`, remember that `GetIntlResource` is affected by the state of the international resources selection flag.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Script.h`

SetSysDirection

Sets the value of `SysDirection`, the global variable that determines the system direction (primary line direction). (Deprecated in Mac OS X v10.4. There is no replacement because this function is no longer needed in Mac OS X.)

```
void SetSysDirection (
   short value
);
```

Parameters

`value`
: The desired value for `SysDirection`:0 if you wish the system direction to be left-to-right and -1 ($FFFF) if you wish the system direction to be right-to-left.

Return Value

Discussion
The value of `SysDirection` is initialized from the system's international configuration resource, and may be controlled by the user. Your application can use the `SetSysDirection` function to change `SysDirection` while drawing, but should restore it when appropriate (such as when your application becomes inactive).

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`Script.h`

TransliterateText

Converts characters from one subscript to the closest possible approximation in a different subscript within the same double-byte script system. (Deprecated in Mac OS X v10.4. Use `CFStringUppercase` instead.)

```
OSErr TransliterateText (
   Handle srcHandle,
   Handle dstHandle,
   short target,
   long srcMask,
   ScriptCode script
);
```

Parameters

srcHandle

A handle to the source text to be transliterated. The `TransliterateText` function converts all of the text that you pass it in this parameter. It determines the length of the source text (in bytes) from the handle size.

dstHandle

A handle to a buffer that, upon completion of the call, contains the transliterated text.

Before calling `TransliterateText`, allocate a handle (of any size) to pass in the `dstHandle` parameter. The length of the transliterated text may be different (as when converting between single-byte and double-byte characters), and `TransliterateText` sets the size of the destination handle as required. It is your responsibility to dispose of the destination handle when you no longer need it.

target

A value that specifies what kind of text the source text is to be transliterated into.

The low byte of the target is the format to convert to (the target format). It determines what form the text should be transliterated to. In all script systems, there are two currently supported values for target format: `smTransAscii` and `smTransNative`. In double-byte script systems, additional values are recognized.

The high byte is the target modifier; it contains modifiers, whose meanings depend on the script code, providing additional formatting instructions. In all script systems, there are two values for target modifier: `smTransLower` and `smTransUpper`.

srcMask

A bit array that specifies which parts of the source text are to be transliterated. A bit is set for each script system or subscript that should be converted to the target format. In all script systems, the `srcMask` parameter may have the following values: `smMaskAscii`, `smMaskNative`, and `smMaskAll`. In double-byte script systems, additional values are recognized.

script

A value that specifies the script system of the text to be transliterated. Constants for all defined script codes are listed in "Meta Script Codes" (page 25). To specify the font script, pass `smCurrentScript` in this parameter.

Return Value

A result code. See "Script Manager Result Codes" (page 93).

Discussion

Transliteration is the conversion of text from one form or subscript to another within a single script system. In the Roman script system, transliteration means case conversion. In double-byte script systems, it is the automatic conversion of characters from one subscript to another. One common use for transliteration is as an initial stage of text conversion for an input method.

`TransliterateText` also performs uppercasing and lowercasing, with consideration for regional variants, in the Roman script system and on Roman text within double-byte script systems.

Because the low-ASCII character set (character codes $20–$7F) is present in all script systems, you could theoretically use the `TransliterateText` function to convert characters from one script system into another completely different script system. You could transliterate from a native subscript into ASCII under one script system, and then transliterate from that ASCII into a native subscript under a different script system. Such a function is not recommended, however, because of the imperfect nature of phonetic translation. Furthermore, many script systems do not support transliteration from native subscripts to ASCII.

Special Considerations

`TransliterateText` may move memory; your application should not call this function at interrupt time.

If you pass `smCurrentScript` in the `script` parameter, the conversion performed by `TransliterateText` can be affected by the state of the font force flag. It is unaffected by the international resources selection flag.

Transliteration of a block of text does not work across script-run boundaries. Because the `TransliterateText` function requires transliteration tables that are in a script system's international resources, you need to call it anew for each script run in your text.

Currently, the Roman version of `TransliterateText` checks the source mask only to ensure that at least one of the bits corresponding to the `smMaskAscii` and `smMaskNative` constants is set.

The Arabic and Hebrew versions of `TransliterateText` perform case conversion only. They allow the target values `smTransAscii` and `smTransNative` only; otherwise, they behave like the Roman version.

The `TransliterateText` tables for single-byte script systems reside in the script's string-manipulation (`'itl2'`) resource, so they can reflect region-specific or language-specific differences in uppercase conventions. If the string-manipulation resource does not include these tables, `TransliterateText` exits without doing anything.

The `TransliterateText` tables for double-byte script systems reside in the script's transliteration (`'trsl'`) resource. If the `'trsl'` resource does not include these tables, `TransliterateText` exits without doing anything.

The Japanese, Traditional Chinese, and Simplified Chinese versions of `TransliterateText` have two modes of operation. If either `smMaskAscii` or `smMaskNative` is specified in the source mask, and if the target is `smTransAscii`, and if either of the target modifiers is specified, `TransliterateText` performs the specified case conversion on both single-byte and double-byte Roman letters. Otherwise, `TransliterateText` performs conversions according to the target format values. Any combination of source masks and target format is permitted.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Script.h`

Deprecated in Mac OS X v10.5

GetIntlResource

Returns a handle to one of the international resources. (Deprecated in Mac OS X v10.5. The replacement for this function depends on the purpose for which it is used, as described in the Special Considerations section.)

```
Handle GetIntlResource (
   short theID
);
```

Parameters

theID

Contains an integer (0, 1, 2, 4, or 5 respectively for the `'itl0'`, `'itl1'`, `'itl2'`, `'itl4'`, and `'itl5'` resources) to identify the type of the desired international resource.

Return Value

A handle to the correct resource of the requested type. The resource returned is that of the current script, which is either the font script or the system script. The resource is of one of the following types: numeric-format (`'itl0'`), long-date-format (`'itl1'`), string-manipulation (`'itl2'`), tokens (`'itl4'`), or encoding/rendering (`'itl5'`). If `GetIntlResource` cannot return the requested resource, it returns a `NULL` handle and sets the global variable `resErr` to the appropriate error code.

Special Considerations

Depending on the information that this function was called to obtain, it can be replaced by the use of `CFLocaleCopyCurrent` to get an appropriate `CFLocaleRef` followed by one of the following:

- `CFLocaleGetValue` with keys such as `kCFLocaleUsesMetricSystem`, `kCFLocaleDecimalSeparator`, `kCFLocaleCurrencySymbol`.
- `CFDateFormatterCreate` to get an appropriate `CFDateFormatterRef` object, followed by `CFDateFormatterCopyProperty` with keys such as `kCFDateFormatterMonthSymbols`, `kCFDateFormatterWeekdaySymbols`, and `kCFDateFormatterAMSymbol`.
- `CFNumberFormatterCreate` to get an appropriate `CFNumberFormatterRef` object, followed by `CFNumberFormatterCopyProperty` with keys such as `kCFNumberFormatterCurrencyDecimalSeparator`, `kCFNumberFormatterMinusSign`, `kCFNumberFormatterPercentSymbol`, and `kCFNumberFormatterNegativePrefix`.

`GetIntlResource` may move memory; your application should not call this function at interrupt time.

Carbon Porting Notes

While the return type of this function remains a Handle, in Mac OS X it returns an ordinary memory handle instead of a resource handle.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Script.h`

GetScriptManagerVariable

Retrieves the value of the specified Script Manager variable. (Deprecated in Mac OS X v10.5. The replacement for this function depends on the selector used with it, as described in the Special Considerations section.)

```
long GetScriptManagerVariable (
   short selector
);
```

Parameters

selector

A value that specifies a particular Script Manager variable. To specify the Script Manager variable whose value you need, use one of the selector constants listed in "Script Manager Selectors" (page 40).

Return Value

The current value of the specified Script Manager variable or 0 if the selector is invalid. For some valid selectors, 0 may also be a valid return value. For example, when you call `GetScriptManagerVariable` with a selector value of `smRegionCode` on a version of Macintosh system software that has been localized for the United States, it returns 0. Although `GetScriptManagerVariable` always returns a long integer, the actual value may be a long integer, standard integer, or signed byte. If the value is not a long integer, it is stored in the low-order word or byte of the long integer returned by `GetScriptManagerVariable`; the remaining bytes are set to 0.

Discussion

The Script Manager maintains a set of variables that control general settings of the text environment, including the identity of the system script and the keyboard script, and the settings of the font force flag and the international resources selection flag.

You may want access to the Script Manager variables in order to understand the current environment or to modify it.

Special Considerations

The replacement for this function depends on the selector used with it. Many of the selectors refer to information that is not meaningful on a Unicode system or refer to details of the Script Manager itself; in general there are no replacements for these. Selectors that have meaningful replacements are shown in the following list. These are not direct replacements; they provide analogous but more modern functionality.

`smSysScript`. To obtain a text encoding for the legacy Mac OS encoding associated with the user's preferred user interface language or with the application's default text encoding, use `CFStringGetSystemEncoding` or `GetApplicationTextEncoding`. Sometimes `smSysScript` is just used to get a script code to pass to `GetScriptVariable` (page 111); in this case the replacements for `GetScriptVariable` (page 111) selectors may provide more information.

`smKeyScript`. To obtain the intended language associated with the user's current keyboard input source (plus other languages that can be input using it), use `TISCopyCurrentKeyboardInputSource` to get that input source, then pass it to `TISGetInputSourceProperty` with the `kTISPropertyInputSourceLanguages` key.

`smKCHRCache`. To obtain the key layout data for the keyboard layout currently in use, use `TISCopyCurrentKeyboardLayoutInputSource` to get that input source, then pass it to `TISGetInputSourceProperty` with the `kTISPropertyUnicodeKeyLayoutData` key (this returns `'uchr'` Unicode ayout data only; it will not return any data for keyboard layouts that only have `'KCHR'` data).

`smRegionCode`. To obtain the locale associated with the user's preferred formats (for dates, times, numbers, and so on) use the following code:

```
CFStringRef curLocaleStringRef = NULL;
```

```
    localeRef = CFLocaleCopyCurrent();
    if (localeRef) {
        curLocaleStringRef = CFLocaleGetIdentifier(localeRef);
        CFRelease(localeRef);
    }
```

To obtain the user's preferred user interface language, use the following line of code:

```
CFArrayRef langArray = (CFArrayRef)CFPreferencesCopyAppValue(CFSTR("AppleLanguages"),
kCFPreferencesCurrentApplication);
```

The first entry in `langArray` indicates the preferred language. See also `CFLocaleCopyPreferredLanguages`.

Selectors that have no meaningful replacement on a Unicode system include `smEnabled`, `smBidirect`, and `smDoubleByte`. Selectors that pertain to internal operation of the Script Manager itself and thus have no meaningful replacement include `smVersion`, `smMunged`, `smPrint`, and `smSysRef`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`Script.h`

GetScriptVariable

Retrieves the value of the specified script variable from the specified script system. (Deprecated in Mac OS X v10.5. The replacement for this function depends on the selector used with it, as described in the Special Considerations section.)

```
long GetScriptVariable (
   short script,
   short selector
);
```

Parameters

`script`

A value that specifies the script system whose variable you are accessing. Use one of the script-code constants listed in "Meta Script Codes" (page 25).

`selector`

A value that specifies a particular script variable. Use one of the selector constants listed in "Script Variable Selectors" (page 45). Valid selector values are defined by each script system.

Return Value

0 if the selector value is invalid or if the specified script system is not installed. For some valid selectors, 0 may also be a valid return value. For example, calling `GetScriptVariable` with a selector of `smScriptLang` on a version of Macintosh system software that has been localized for the United States returns 0. Although `GetScriptVariable` always returns a long integer, the actual value may be a long integer, standard integer, or signed byte. If the value is not a long integer, it is stored in the low-order word or byte of the long integer returned by `GetScriptVariable`; the remaining bytes are set to 0.

Discussion

Each enabled script system maintains a set of variables that control the current settings of that script system, including the ID numbers of its international resources, its preferred fonts and font sizes, and its primary line direction.

Special Considerations

The replacement for this function depends on the selector used with it. Many of the selectors refer to information that is not meaningful on a Unicode system or refer to details of the Script Manager itself; in general there are no replacements for these. Selectors that have meaningful replacements are shown in the following list. These are not direct replacements; they provide analogous but more modern functionality.

`smScriptLang`. This was typically used with the system script to determine the system language. Instead, to obtain the user's preferred user interface language, use the following line of code:

```
CFArrayRef langArray = (CFArrayRef)CFPreferencesCopyAppValue(CFSTR("AppleLanguages"),
kCFPreferencesCurrentApplication);
```

The first entry in `langArray` indicates the preferred language. See also `CFLocaleCopyPreferredLanguages`.

Font selectors `smScriptSysFond`, `smScriptSysFondSize`, `smScriptAppFond`, `smScriptAppFondSize`, `smScriptMonoFondSize`, `smScriptPrefFondSize`, `smScriptSmallFondSize`, and `smScriptHelpFondSize`. On Mac OS X you generally do not need to worry about setting an appropriate font based on character script to ensure that characters are displayed correctly; Unicode encoding and font fallbacks (to automatically find a font that can display a character) take care of this. However, for cases where you do need to do this (such as Carbon applications that handle non-Unicode text), the Core Text function `CTFontCreateUIFontForLanguage` (available in Mac OS X v10.5 and later) provides a way to get a `CTFontRef` object for a specified language and user interface use.

Script resource ID selectors `smScriptNumber`, `smScriptDate`, `smScriptSort`, and `smScriptToken`. These were used in several ways. Sometimes they were used to get a resource ID so specific fields in the resource could be examined (for example, to determine the appropriate decimal separator or time format). For this use `CFLocaleGetValue` can now be used with an appropriate key (for example, `kCFLocaleDecimalSeparator`) to get similar information (much of the information associated with the resource specified by `smScriptToken` is not relevant for a Unicode system). Another use was to get a resource ID (or a handle) to pass to some other system function. For text sorting, this is replaced by the collation functionality in CFString. For formatting of times, dates, and numbers, this is replaced by functionality in CFLocale, CFDateFormatter, CFNumberFormatter.

`smScriptKeys`. To determine an appropriate keyboard input source for a particular language, use `TISCopyInputSourceForLanguage`.

`smScriptIcon`. To obtain an icon for a particular keyboard input source, use `TISGetInputSourceProperty` with the `kTISPropertyIconRef` or `kTISPropertyIconImageURL` key.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Script.h`

SetScriptManagerVariable

Sets the specified Script Manager variable to the value of the input parameter. (Deprecated in Mac OS X v10.5. This is mainly used to set the value of variables that control the internal operation of the Script Manager (selectors `smIntlForce` and `smGenFlags`), and therefore there is no modern replacement.)

```
OSErr SetScriptManagerVariable (
   short selector,
   long param
);
```

Parameters

`selector`

A value that specifies a particular Script Manager variable. To specify the Script Manager variable whose value you wish to change, use one of the selector constants listed in "Script Manager Selectors" (page 40).

`param`

The new value for the specified Script Manager variable.

The actual values to be assigned may be long integers, standard integers, or signed bytes. If the value is other than a long integer, you must store it in the low-order word or byte of the `param` parameter and set the unused bytes to 0.

Return Value

A result code. See "Script Manager Result Codes" (page 93). The value `smBadVerb` if the selector is not valid. Otherwise, the function returns 0 (`noErr`).

Discussion

The Script Manager maintains a set of variables that control general settings of the text environment, including the identity of the system script and the keyboard script, and the settings of the font force flag and the international resources selection flag.

You may want access to the Script Manager variables in order to understand the current environment or to modify it.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`Script.h`

SetScriptVariable

Sets the specified script variable for the specified script system to the value of the input parameter. (Deprecated in Mac OS X v10.5. The replacement for this function depends on the purpose for which it is used, as described in the Special Considerations section.)

```
OSErr SetScriptVariable (
   short script,
   short selector,
   long param
);
```

Parameters

script

A value that specifies the script system whose variable you are setting. Use one of the script-code constants listed in "Meta Script Codes" (page 25).

selector

A value that specifies a particular script variable. Use one of the selector constants listed in "Script Variable Selectors" (page 45).

param

The new value for the specified script variable. The actual value to be assigned may be a long integer, standard integer, or signed byte. If the value is not a long integer, you must store it in the low-order word or byte of the `param` parameter and set the unused bytes to 0.

Return Value

A result code. See "Script Manager Result Codes" (page 93). The value `smBadVerb` if the selector is not valid, and `smBadScript` if the script is invalid. Otherwise, 0 (`noErr`).

Discussion

Each enabled script system maintains a set of variables that control the current settings of that script system, including the ID numbers of its international resources, its preferred fonts and font sizes, and its primary line direction.

Special Considerations

The replacement for this function depends on whether the goal is to set the keyboard layout globally or for a specific TSM document. To set it globally, use `TISSelectInputSource`. To set it for a specific document, use the TSM document property `kTSMDocumentInputSourceOverridePropertyTag`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Script.h`

Document Revision History

This table describes the changes to *Script Manager Reference*.

Date	Notes
2007-12-11	Deprecated document. Added information about replacement technologies and workarounds for functions deprecated in Mac OS X v10.5.
2006-07-12	Made minor formatting changes.
2006-07-24	Added deprecation information.
2005-09-08	Fixed a linking problem.
2005-07-07	Fixed broken links and added cross references to the KeyScript function.
2003-04-10	Fixed an error regarding return values for the function `GetSysDirection` (page 102) and `SetSysDirection` (page 106).
2003-02-12	Updated formatting. Added documentation for data types and constants.

Index