Text Encoding Conversion Manager Reference

Carbon > Text & Fonts

2005-07-07

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Contents

Tables

Text Encoding Conversion Manager Reference

Framework:	CoreServices/CoreServices.h
Declared in	TextCommon.h TextEncodingConverter.h TextEncodingPlugin.h UnicodeConverter.h

Overview

The Text Encoding Conversion (TEC) Manager provides two facilities—the Text Encoding Converter and the Unicode Converter—that your application can use to handle text encoding conversion on the Mac OS. You will find the Text Encoding Conversion Manager helpful if you develop Internet applications, such as Web browsers or e-mail applications, applications that transfer text across different platforms, or applications based in Unicode.

Functions by Task

Creating a Text Encoding Specification

`CreateTextEncoding` (page 30)
Creates and returns a text encoding specification.

Obtaining Information From a Text Encoding Specification

`GetTextEncodingBase` (page 39)
Returns the base encoding of the specified text encoding.

`GetTextEncodingFormat` (page 39)
Returns the format value of the specified text encoding.

`GetTextEncodingName` (page 39)
Returns the localized name for a specified text encoding.

`GetTextEncodingVariant` (page 41)
Returns the variant from the specified text encoding.

`ResolveDefaultTextEncoding` (page 46)
Returns a text encoding specification in which any meta-values have been resolved to real values. Currently, this affects only the base encoding values packed into the text encoding specification.

Converting Between Script Manager Values and Text Encodings

`RevertTextEncodingToScriptInfo` (page 47)
Converts the given Mac OS text encoding specification to the corresponding script code and, if possible, language code and font name.

`UpgradeScriptInfoToTextEncoding` (page 77)
Converts any combination of a Mac OS script code, a language code, a region code, and a font name to a text encoding.

Obtaining Information About Available Text Encodings

`TECCountAvailableTextEncodings` (page 54)
Counts and returns the number of text encodings currently configured in the Text Encoding Converter.

`TECCountSubTextEncodings` (page 57)
Counts and returns the number of subencodings a text encoding supports.

`TECGetAvailableTextEncodings` (page 66)
Returns the text encoding specifications currently configured in the Text Encoding Converter.

`TECGetSubTextEncodings` (page 70)
Returns the text encoding specifications for the subencodings the encoding scheme supports.

`NearestMacTextEncodings` (page 42)
Obtains the best and alternate Mac text encoding.

Identifying Direct Encoding Conversions

`TECCountDirectTextEncodingConversions` (page 56)
Counts and returns the number of direct conversions currently configured in the Text Encoding Converter.

`TECGetDirectTextEncodingConversions` (page 67)
Returns the types of direct conversions currently configured in the Text Encoding Converter.

Identifying Possible Destination Encodings

`TECCountDestinationTextEncodings` (page 55)
Counts and returns the number of destination encodings to which a specified source encoding can be converted in one step.

`TECGetDestinationTextEncodings` (page 66)
Returns the encoding specifications for all the destination text encodings to which the Text Encoding Converter can directly convert the specified source encoding.

Obtaining Converter Information

`TECGetInfo` (page 69)
Allocates a converter information structure of type `TECInfo` in the application heap using `NewHandle`, fills it out, and returns a handle.

Creating and Deleting Converter Objects

`TECCreateConverter` (page 58)
: Determines a conversion path for a source and destination encoding, then creates a text encoding converter object and returns a pointer to it.

`TECCreateConverterFromPath` (page 59)
: Creates a converter object for a specific conversion path—from a source encoding through intermediate encodings to a destination encoding—and returns a pointer to it.

`TECClearConverterContextInfo` (page 50)
: Resets a converter object to its initial state so you can reuse it.

`TECDisposeConverter` (page 61)
: Disposes of a converter object.

Converting Text Between Encodings

`TECConvertText` (page 51)
: Converts a stream of text from a source encoding to a destination encoding. It uses the conversion path specified by the converter object you supply.

`TECFlushText` (page 64)
: Flushes out any data in a converter object's temporary buffers and resets the converter object.

Converting to Multiple Encoding Runs

`TECConvertTextToMultipleEncodings` (page 52)
: Converts text in the source encoding to runs of text in multiple destination encodings. It uses the conversion path specified in the converter object you supply.

`TECCreateOneToManyConverter` (page 60)
: Determines a conversion path for the source encoding and destinations encodings you specify, creates a text encoding converter object, and returns a reference to it.

`TECFlushMultipleEncodings` (page 62)
: Flushes out any encodings that may be stored in a converter object's temporary buffers and shifts encodings back to their default state, if any.

`TECGetEncodingList` (page 68)
: Gets the list of destination encodings from a converter object.

Using Sniffers to Investigate Encodings

`TECCreateSniffer` (page 60)
: Creates a sniffer object and returns a reference to it.

`TECClearSnifferContextInfo` (page 50)
: Resets a sniffer object to its initial settings so you can reuse it.

`TECDisposeSniffer` (page 62)
: Disposes of a sniffer object.

`TECCountAvailableSniffers` (page 54)
Counts and returns the number of sniffers available in all installed plug-ins.

`TECGetAvailableSniffers` (page 65)
Returns the list of sniffers available in all installed plug-ins.

`TECSniffTextEncoding` (page 73)
Analyzes a text stream and returns the probable encodings in a ranked list, based on an array of possible encodings you supply. It also returns the number of errors and features for each encoding.

Getting Information About Internet and Regional Text Encoding Names

`TECCountMailTextEncodings` (page 56)
Counts and returns the number of currently supported e-mail encodings for a specified region.

`TECCountWebTextEncodings` (page 58)
Counts and returns the number of currently supported text encodings for a region code.

`TECGetMailTextEncodings` (page 69)
Returns the currently supported mail encoding specifications for a region code.

`TECGetTextEncodingFromInternetName` (page 71)
Returns the Mac OS text encoding specification that corresponds to an Internet encoding name.

`TECGetTextEncodingInternetName` (page 71)
Returns the Internet encoding name that corresponds to a Mac OS text encoding.

`TECGetWebTextEncodings` (page 72)
Returns the currently supported text encoding specifications for a region code.

Converting to Unicode

`ChangeTextToUnicodeInfo` (page 15)
Changes the mapping information for the specified Unicode converter object used to convert text to Unicode to the new mapping you provide.

`ConvertFromTextToUnicode` (page 17)
Converts a string from any encoding to Unicode.

`CreateTextToUnicodeInfo` (page 30)
Creates and returns a Unicode converter object containing information required for converting strings from a non-Unicode encoding to Unicode.

`CreateTextToUnicodeInfoByEncoding` (page 31)
Based on the given text encoding specification, creates and returns a Unicode converter object containing information required for converting strings from the specified non-Unicode encoding to Unicode.

`DisposeTextToUnicodeInfo` (page 37)
Releases the memory allocated for the specified Unicode converter object.

`ResetTextToUnicodeInfo` (page 45)
Reinitializes all state information kept by the context objects.

Converting From Unicode

`ChangeUnicodeToTextInfo` (page 15)
Changes the mapping information contained in the specified Unicode converter object used to convert Unicode text to a non-Unicode encoding.

`ConvertFromUnicodeToText` (page 23)
Converts a Unicode text string to the destination encoding you specify.

`CreateUnicodeToTextInfo` (page 32)
Creates and returns a Unicode converter object containing information required for converting strings from Unicode to a non-Unicode encoding.

`CreateUnicodeToTextInfoByEncoding` (page 33)
Based on the given text encoding specification for the converted text, creates and returns a Unicode converter object containing information required for converting strings from Unicode to the specified non-Unicode encoding.

`DisposeUnicodeToTextInfo` (page 38)
Releases the memory allocated for the specified Unicode converter object.

`ResetUnicodeToTextInfo` (page 45)
Reinitializes all state information kept by a Unicode converter object.

Converting From Unicode to Multiple Encodings

`ConvertFromUnicodeToTextRun` (page 25)
Converts a string from Unicode to one or more encodings.

`ConvertFromUnicodeToScriptCodeRun` (page 20)
Converts a string from Unicode to one or more scripts.

`CreateUnicodeToTextRunInfo` (page 34)
Creates and returns a Unicode converter object containing the information required for converting a Unicode text string to strings in one or more non-Unicode encodings.

`CreateUnicodeToTextRunInfoByEncoding` (page 35)
Based on the given text encoding specifications for the converted text runs, creates and returns a Unicode converter object containing information required for converting strings from Unicode to one or more specified non-Unicode encodings.

`CreateUnicodeToTextRunInfoByScriptCode` (page 36)
Based on the given script codes for the converted text runs, creates and returns a Unicode converter object containing information required for converting strings from Unicode to one or more specified non-Unicode encodings.

`DisposeUnicodeToTextRunInfo` (page 38)
Releases the memory allocated for the specified Unicode converter object.

`ResetUnicodeToTextRunInfo` (page 46)
Reinitializes all state information kept by the context objects in TextRun conversions.

Converting Between Unicode and Pascal Strings

`ConvertFromPStringToUnicode` (page 16)
Converts a Pascal string in a Mac OS text encoding to a Unicode string.

`ConvertFromUnicodeToPString` (page 19)
Converts a Unicode string to Pascal in a Mac OS text encoding.

Obtaining Unicode Mapping Information

`CountUnicodeMappings` (page 29)
Counts available mappings that meet the specified matching criteria.

`QueryUnicodeMappings` (page 43)
Returns a list of the conversion mappings available on the system that meet specified matching criteria and returns the number of mappings found.

Truncating Strings Before Converting Them

`TruncateForTextToUnicode` (page 74)
Identifies where your application can safely break a multibyte string to be converted to Unicode so that the string is not broken in the middle of a multibyte character.

`TruncateForUnicodeToText` (page 75)
Identifies where your application can safely break a Unicode string to be converted to any encoding so that the string is broken in a way that preserves the text element integrity.

Setting the Fallback Handler

`SetFallbackUnicodeToText` (page 48)
Specifies a fallback handler to be used for converting a Unicode text segment to another encoding when the Unicode Converter cannot convert the text using the mapping table specified by the Unicode converter object.

`SetFallbackUnicodeToTextRun` (page 49)
Specifies a fallback handler to be used for converting a Unicode text segment to another encoding when the Unicode Converter cannot convert the text using the mapping table specified by a Unicode converter object.

Working With Universal Procedure Pointers

`NewUnicodeToTextFallbackUPP` (page 43)
Creates a new universal procedure pointer (UPP) to a Unicode-to-text fallback callback.

`DisposeUnicodeToTextFallbackUPP` (page 37)
Disposes of a a new universal procedure pointer (UPP) to a Unicode-to-text fallback callback.

`InvokeUnicodeToTextFallbackUPP` (page 41)
Calls your Unicode-to-text fallback callback.

Getting UniChar Property Values

`UCGetCharProperty` (page 76)
Obtains the value associated with a property type for the specified `UniChar` characters.

Functions

ChangeTextToUnicodeInfo

Changes the mapping information for the specified Unicode converter object used to convert text to Unicode to the new mapping you provide.

```
OSStatus ChangeTextToUnicodeInfo (
   TextToUnicodeInfo ioTextToUnicodeInfo,
   ConstUnicodeMappingPtr iUnicodeMapping
);
```

Parameters

`ioTextToUnicodeInfo`
The Unicode converter object of type `TextToUnicodeInfo` (page 106) containing the mapping to be modified. You use the function `CreateTextToUnicodeInfo` (page 30) to obtain one.

`iUnicodeMapping`
A structure of type `UnicodeMapping` (page 107) identifying the new mapping to be used. This is the mapping that replaces the existing mapping in the Unicode converter object.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
The function replaces the mapping table information that currently exists in the Unicode converter object pointed to by the `ioTextToUnicodeInfo` parameter with the information contained in the `UnicodeMapping` structure you supply as the `iUnicodeMapping` parameter.

`ChangeTextToUnicodeInfo` resets the Unicode converter object's fields as necessary.

If an error is returned, the Unicode converter object is invalid.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

ChangeUnicodeToTextInfo

Changes the mapping information contained in the specified Unicode converter object used to convert Unicode text to a non-Unicode encoding.

```
OSStatus ChangeUnicodeToTextInfo (
   UnicodeToTextInfo ioUnicodeToTextInfo,
   ConstUnicodeMappingPtr iUnicodeMapping
);
```

Parameters

ioUnicodeToTextInfo

The Unicode converter object of type `UnicodeToTextInfo` (page 109) to be modified. You use the function `CreateUnicodeToTextInfo` (page 32) or `CreateUnicodeToTextInfoByEncoding` (page 33) to obtain a Unicode converter object of this type.

iUnicodeMapping

The structure of type `UnicodeMapping` (page 107) to be used. This is the new mapping that replaces the existing mapping in the Unicode converter object.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion

The function replaces the mapping table information that currently exists in the specified Unicode converter object with the information contained in the new Unicode mapping structure you provide.

`ChangeUnicodeToTextInfo` resets the Unicode converter object's fields as necessary. However, it does not initialize or reset the conversion state maintained by the Unicode converter object.

This function is especially useful for converting a string from Unicode if the Unicode string contains characters that require multiple destination encodings and you know the next destination encoding.

For example, you can change the other (destination) encoding of the Unicode mapping structure pointed to by the `iUnicodeMapping` parameter before you call the function `ConvertFromUnicodeToText` (page 23) to convert the next character or sequence of characters that require a different destination encoding.

If an error is returned, the Unicode converter object is invalid.

Availability

Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`UnicodeConverter.h`

ConvertFromPStringToUnicode

Converts a Pascal string in a Mac OS text encoding to a Unicode string.

```
OSStatus ConvertFromPStringToUnicode (
   TextToUnicodeInfo iTextToUnicodeInfo,
   ConstStr255Param iPascalStr,
   ByteCount iOutputBufLen,
   ByteCount *oUnicodeLen,
   UniChar oUnicodeStr[]
);
```

Parameters

iTextToUnicodeInfo

A Unicode converter object of type `TextToUnicodeInfo` (page 106) for the Pascal string to be converted. You can use the function `CreateTextToUnicodeInfo` (page 30) or `CreateTextToUnicodeInfoByEncoding` (page 31) to create the Unicode converter object.

iPascalStr

The Pascal string to be converted to Unicode.

iOutputBufLen

The length in bytes of the output buffer pointed to by the `oUnicodeStr` parameter. Your application supplies this buffer to hold the returned converted string. The `oUnicodeLen` parameter may return a byte count that is less than this value if the converted string is smaller than the buffer size you allocated.

oUnicodeLen

On return, a pointer to the length in bytes of the converted Unicode string returned in the `oUnicodeStr` parameter.

oUnicodeStr

A pointer to a Unicode character array. On return, this array holds the converted Unicode string.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion

The `ConvertFromPStringToUnicode` function provides an easy and efficient way to convert a short Pascal string to a Unicode string without incurring the overhead associated with the function `ConvertFromTextToUnicode` (page 17).

If necessary, this function automatically uses fallback characters to map the text elements of the string.

Availability

Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`UnicodeConverter.h`

ConvertFromTextToUnicode

Converts a string from any encoding to Unicode.

```
OSStatus ConvertFromTextToUnicode (
   TextToUnicodeInfo iTextToUnicodeInfo,
   ByteCount iSourceLen,
   ConstLogicalAddress iSourceStr,
   OptionBits iControlFlags,
   ItemCount iOffsetCount,
   const ByteOffset iOffsetArray[],
   ItemCount *oOffsetCount,
   ByteOffset oOffsetArray[],
   ByteCount iOutputBufLen,
   ByteCount *oSourceRead,
   ByteCount *oUnicodeLen,
   UniChar oUnicodeStr[]
);
```

Parameters

iTextToUnicodeInfo

A Unicode converter object of type `TextToUnicodeInfo` containing mapping and state information used for the conversion. The contents of this Unicode converter object are modified by the function. Your application obtains a Unicode converter object using the function `CreateTextToUnicodeInfo` (page 30).

iSourceLen

The length in bytes of the source string to be converted.

iSourceStr

The address of the source string to be converted.

iControlFlags

Conversion control flags. You can use "Conversion Masks" (page 112) to set the `iControlFlags` parameter.

iOffsetCount

The number of offsets in the `iOffsetArray` parameter. Your application supplies this value. The number of entries in `iOffsetArray` must be fewer than the number of bytes specified in `iSourceLen`. If you don't want offsets returned to you, specify `0` (zero) for this parameter.

iOffsetArray

An array of type `ByteOffset`. On input, you specify the array that contains an ordered list of significant byte offsets pertaining to the source string. These offsets may identify font or style changes, for example, in the source string. All array entries must be less than the length in bytes specified by the `iSourceLen` parameter. If you don't want offsets returned to your application, specify `NULL` for this parameter and `0` (zero) for `iOffsetCount`.

oOffsetCount

On return, a pointer to the number of offsets that were mapped in the output stream.

oOffsetArray

An array of type `ByteOffset`. On return, this array contains the corresponding new offsets for the Unicode string produced by the converter.

iOutputBufLen

The length in bytes of the output buffer pointed to by the `oUnicodeStr` parameter. Your application supplies this buffer to hold the returned converted string. The `oUnicodeLen` parameter may return a byte count that is less than this value if the converted byte string is smaller than the buffer size you allocated. The relationship between the size of the source string and the Unicode string is complex and depends on the source encoding and the contents of the string.

oSourceRead

On return, a pointer to the number of bytes of the source string that were converted. If the function returns a `kTECUnmappableElementErr` result code, this parameter returns the number of bytes that were converted before the error occurred.

oUnicodeLen

On return, a pointer to the length in bytes of the converted stream.

oUnicodeStr

A pointer to an array used to hold a Unicode string. On input, this value points to the beginning of the array for the converted string. On return, this buffer holds the converted Unicode string. (For guidelines on estimating the size of the buffer needed, see the discussion.

Return Value
A result code. See "TEC Manager Result Codes" (page 166). The function returns a `noErr` result code if it has completely converted the input string to Unicode without using fallback characters.

Discussion
You specify the source string's encoding in the Unicode mapping structure that you pass to the function `CreateTextToUnicodeInfo` (page 30) to obtain a Unicode converter object for the conversion. You pass the Unicode converter object returned by `CreateTextToUnicodeInfo` to `ConvertFromTextToUnicode` as the `iTextToUnicodeInfo` parameter.

In addition to converting a text string in any encoding to Unicode, the `ConvertFromTextToUnicode` function can map offsets for style or font information from the source text string to the returned converted string. The converter reads the application-supplied offsets, which apply to the source string, and returns the corresponding new offsets in the converted string. If you do not want the offsets at which font or style information occurs mapped to the resulting string, you should pass `NULL` for `iOffsetArray` and `0` (zero) for `iOffsetCount`.

Your application must allocate a buffer to hold the resulting converted string and pass a pointer to the buffer in the `oUnicodeStr` parameter. To determine the size of the output buffer to allocate, you should consider the size of the source string, its encoding type, and its content in relation to the resulting Unicode string.

For example, for 1-byte encodings, such as MacRoman, the Unicode string will be at least double the size (more if it uses noncomposed Unicode) for MacArabic and MacHebrew, the corresponding Unicode string could be up to six times as big. For most 2-byte encodings, for example Shift-JIS, the Unicode string will be less than double the size. For international robustness, your application should allocate a buffer three to four times larger than the source string. If the output Unicode text is actually UTF-8—which could occur beginning with the current release of the Text Encoding Conversion Manager, version 1.2.1—the UTF-8 buffer pointer must be cast to `UniCharArrayPtr` before it can be passed as the `oUnicodeStr` parameter. Also, the output buffer length will have a wider range of variation than for UTF-16; for ASCII input, the output will be the same size; for Han input, the output will be twice as big, and so on.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

ConvertFromUnicodeToPString

Converts a Unicode string to Pascal in a Mac OS text encoding.

```
OSStatus ConvertFromUnicodeToPString (
   UnicodeToTextInfo iUnicodeToTextInfo,
   ByteCount iUnicodeLen,
   const UniChar iUnicodeStr[],
   Str255 oPascalStr
);
```

Parameters

iUnicodeToTextInfo

A Unicode converter object. You use the `CreateUnicodeToTextInfo` or `CreateUnicodeToTextInfoByEncoding` function to obtain the Unicode converter object for the conversion.

iUnicodeLen

The length in bytes of the Unicode string to be converted. This is the string your application provides in the `iUnicodeStr` parameter.

iUnicodeStr

A pointer to an array containing the Unicode string to be converted.

oPascalStr

A buffer. On return, the converted Pascal string returned by the function.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion

The `ConvertFromUnicodeToPString` function provides an easy and efficient way to convert a Unicode string to a Pascal string in a Mac OS text encoding without incurring the overhead associated with use of the function `ConvertFromUnicodeToText` (page 23) or `ConvertFromUnicodeToScriptCodeRun` (page 20).

If necessary, this function uses the loose mapping and fallback characters to map the text elements of the string. For fallback mappings, it uses the handler associated with the Unicode converter object.

Availability

Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`UnicodeConverter.h`

ConvertFromUnicodeToScriptCodeRun

Converts a string from Unicode to one or more scripts.

```
OSStatus ConvertFromUnicodeToScriptCodeRun (
   UnicodeToTextRunInfo iUnicodeToTextInfo,
   ByteCount iUnicodeLen,
   const UniChar iUnicodeStr[],
   OptionBits iControlFlags,
   ItemCount iOffsetCount,
   const ByteOffset iOffsetArray[],
   ItemCount *oOffsetCount,
   ByteOffset oOffsetArray[],
   ByteCount iOutputBufLen,
   ByteCount *oInputRead,
   ByteCount *oOutputLen,
   LogicalAddress oOutputStr,
   ItemCount iScriptRunBufLen,
   ItemCount *oScriptRunOutLen,
   ScriptCodeRun oScriptCodeRuns[]
);
```

Parameters

iUnicodeToTextInfo

You use the function `CreateUnicodeToTextRunInfoByScriptCode` (page 36) to obtain a Unicode converter object to specify for this parameter.

iUnicodeLen

The length in bytes of the Unicode string to be converted.

iUnicodeStr

A pointer to the Unicode string to be converted.

iControlFlags

Conversion control flags. The following constants define the masks for control flags valid for this parameter. You can use "Conversion Masks" (page 112) and "Directionality Masks" (page 116) to set the `iControlFlags` parameter.

If the text-run control flag is clear, `ConvertFromUnicodeToScriptCodeRun` attempts to convert the Unicode text to the single script from the list of scripts in the Unicode converter object that produces the best result, that is, that provides for the greatest amount of source text conversion. If the complete source text can be converted into more than one of the scripts specified in the array, then the converter chooses among them based on their order in the array. If this flag is clear, the `oScriptCodeRuns` parameter always points to a value equal to 1.

If you set the use-fallbacks control flag, the converter uses the default fallback characters for the current script. If the converter cannot handle a character using the current encoding, even using fallbacks, the converter attempts to convert the character using the other scripts, beginning with the first one specified in the list and skipping the one where it failed.

If you set the `kUnicodeTextRunBit` control flag, the converter attempts to convert the complete Unicode text string into the first script specified in the Unicode mapping structures array you passed to `CreateUnicodeToTextRunInfo`, `CreateUnicodeToTextRunInfoByEncoding`, or `CreateUnicodeToTextRunInfoByScriptCode` to create the Unicode converter object used for this conversion. If it cannot do this, the converter then attempts to convert the first text element that failed to the remaining scripts, in their specified order in the array. What the converter does with the next text element depends on the setting of the keep-same-encoding control flag:

If the keep-same-encoding control flag is clear, the converter returns to the original script and attempts to continue conversion with that script; this is equivalent to converting each text element to the first one that works, in the order specified.

If the Unicode-keep-same-encoding control flag is set, the converter continues with the new destination script until it encounters a text element that cannot be converted using the new script. This attempts to minimize the number of script code changes in the output text. When the converter cannot convert a text element using any of the scripts in the list and the Unicode-keep-same-encoding control flag is set, the converter uses the fallbacks default characters for the current script.

iOffsetCount

The number of offsets in the array pointed to by the `iOffsetArray` parameter. Your application supplies this value. The number of entries in `iOffsetArray` must be fewer than half the number of bytes specified in `iUnicodeLen`. If you don't want offsets returned to you, specify `0` (zero) for this parameter.

iOffsetArray

An array of type `ByteOffset`. On input, you specify the array that contains an ordered list of significant byte offsets pertaining to the source Unicode string. These offsets may identify font or style changes, for example, in the Unicode string. If you don't want offsets returned to your application, specify `NULL` for this parameter and `0` (zero) for `iOffsetCount`.

oOffsetCount

On return, a pointer to the number of offsets that were mapped in the output stream.

oOffsetArray

An array of type `ByteOffset`. On return, this array contains the corresponding new offsets for the resulting converted string.

iOutputBufLen

The length in bytes of the output buffer pointed to by the `oOutputStr` parameter. Your application supplies this buffer to hold the returned converted string. The `oOutputLen` parameter may return a byte count that is less than this value if the converted byte string is smaller than the buffer size you allocated.

oInputRead

On return, a pointer to the number of bytes of the Unicode source string that were converted. If the function returns a result code other than `noErr`, then this parameter returns the number of bytes that were converted before the error occurred.

oOutputLen

On return, a pointer to the length in bytes of the converted string.

oOutputStr

A buffer address. On input, this value points to the beginning of the buffer for the converted string. On return, this buffer contains the converted string in one or more encodings. When an error occurs, the `ConvertFromUnicodeToScriptCodeRun` function returns the converted string up to the character that caused the error.

iScriptRunBufLen

The number of script code run elements you allocated for the script code run array pointed to by the `oScriptCodeRuns` parameter. The converter returns the number of valid script code runs in the location pointed to by `oScriptRunOutLen`. Each entry in the script code run array specifies the beginning offset in the converted text and its associated script code.

oScriptRunOutLen

A pointer to a value of type `ItemCount`. On output, this value contains the number of valid script code runs returned in the `oScriptCodeRuns` parameter.

oScriptCodeRuns

An array of elements of type `ScriptCodeRun`. Your application should allocate an array with the number of elements you specify in the `iScriptRunBufLen` parameter. On return, this array contains the script code runs for the converted text string. Each entry in the array specifies the beginning offset in the converted text string and the associated script code specification.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion

To use the `ConvertFromUnicodeToScriptCodeRun` function, you must first set up an array of script codes containing in order of precedence the scripts to be used for the conversion. To create a Unicode converter object, you call the function `CreateUnicodeToTextRunInfoByScriptCode` (page 36). You pass the returned Unicode converter object as the `iUnicodeToTextInfo` parameter when you call the `ConvertFromUnicodeToScriptCodeRun` function.

Availability

Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`UnicodeConverter.h`

ConvertFromUnicodeToText

Converts a Unicode text string to the destination encoding you specify.

```
OSStatus ConvertFromUnicodeToText (
   UnicodeToTextInfo iUnicodeToTextInfo,
   ByteCount iUnicodeLen,
   const UniChar iUnicodeStr[],
   OptionBits iControlFlags,
   ItemCount iOffsetCount,
   const ByteOffset iOffsetArray[],
   ItemCount *oOffsetCount,
   ByteOffset oOffsetArray[],
   ByteCount iOutputBufLen,
   ByteCount *oInputRead,
   ByteCount *oOutputLen,
   LogicalAddress oOutputStr
);
```

Parameters

iUnicodeToTextInfo

A Unicode converter object of type `UnicodeToTextInfo` for converting text from Unicode. You use the function `CreateUnicodeToTextInfo` (page 32) or `CreateUnicodeToTextInfoByEncoding` (page 33) to obtain a Unicode converter object to specify for this parameter. This function modifies the contents of the `iUnicodeToTextInfo` parameter.

iUnicodeLen

The length in bytes of the Unicode string to be converted.

iUnicodeStr

A pointer to the Unicode string to be converted. If the input text is UTF-8, which is supported for versions 1.2.1 or later of the converter, you must cast the UTF-8 buffer pointer to `ConstUniCharArrayPtr` before you can pass it as this parameter.

iControlFlags

Conversion control flags. You can use "Conversion Masks" (page 112) and "Directionality Masks" (page 116) to set the `iControlFlags` parameter.

iOffsetCount

The number of offsets contained in the array provided by the `iOffsetArray` parameter. Your application supplies this value. If you don't want offsets returned to you, specify `0` (zero) for this parameter.

iOffsetArray

An array of type `ByteOffset`. On input, you specify the array that gives an ordered list of significant byte offsets pertaining to the Unicode source string to be converted. These offsets may identify font or style changes, for example, in the source string. If you don't want offsets returned to your application, specify `NULL` for this parameter and `0` (zero) for `iOffsetCount`. All offsets must be less than `iUnicodeLen`.

oOffsetCount

On return, a pointer to the number of offsets that were mapped in the output stream.

oOffsetArray

An array of type `ByteOffset`. On return, this array contains the corresponding new offsets for the converted string in the new encoding.

iOutputBufLen

The length in bytes of the output buffer pointed to by the `oOutputStr` parameter. Your application supplies this buffer to hold the returned converted string. The `oOutputLen` parameter may return a byte count that is less than this value if the converted byte string is smaller than the buffer size you allocated.

oInputRead

On return, a pointer to a the number of bytes of the Unicode string that were converted. If the function returns a `kTECUnmappableElementErr` result code, this parameter returns the number of bytes that were converted before the error occurred.

oOutputLen

On return, a pointer to the length in bytes of the converted text stream.

oOutputStr

A value of type `LogicalAddress`. On input, this value points to a buffer for the converted string. On return, the buffer holds the converted text string. (For guidelines on estimating the size of the buffer needed, see the following discussion.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
This function can also map offsets for style or font information from the source text string to the returned converted string. The converter reads the application-supplied offsets and returns the corresponding new offsets in the converted string. If you do not want font or style information offsets mapped to the resulting string, you should pass `NULL` for `iOffsetArray` and `0` (zero) for `iOffsetCount`.

Your application must allocate a buffer to hold the resulting converted string and pass a pointer to the buffer in the `oOutputStr` parameter. To determine the size of the output buffer to allocate, you should consider the size and content of the Unicode source string in relation to the type of encoding to which it will be converted. For example, for many encodings, such as MacRoman and Shift-JIS, the size of the returned string will be between half the size and the same size as the source Unicode string. However, for some encodings that are not Mac OS ones, such as EUC-JP, which has some 3-byte characters for Kanji, the returned string could be larger than the source Unicode string. For MacArabic and MacHebrew, the result will usually be less than half the size of the Unicode string.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

ConvertFromUnicodeToTextRun

Converts a string from Unicode to one or more encodings.

```
OSStatus ConvertFromUnicodeToTextRun (
   UnicodeToTextRunInfo iUnicodeToTextInfo,
   ByteCount iUnicodeLen,
   const UniChar iUnicodeStr[],
   OptionBits iControlFlags,
   ItemCount iOffsetCount,
   const ByteOffset iOffsetArray[],
   ItemCount *oOffsetCount,
   ByteOffset oOffsetArray[],
   ByteCount iOutputBufLen,
   ByteCount *oInputRead,
   ByteCount *oOutputLen,
   LogicalAddress oOutputStr,
   ItemCount iEncodingRunBufLen,
   ItemCount *oEncodingRunOutLen,
   TextEncodingRun oEncodingRuns[]
);
```

Parameters

iUnicodeToTextInfo

You use the function `CreateUnicodeToTextRunInfo` (page 34), `CreateUnicodeToTextRunInfoByEncoding` (page 35), or `CreateUnicodeToTextRunInfoByScriptCode` (page 36) to obtain a Unicode converter object to specify for this parameter.

iUnicodeLen

The length in bytes of the Unicode string to be converted.

iUnicodeStr

A pointer to the Unicode string to be converted.

iControlFlags

Conversion control flags. The following constants define the masks for control flags valid for this parameter. You can use "Conversion Masks" (page 112) and "Directionality Masks" (page 116) to set the `iControlFlags` parameter.

If the text-run control flag is clear, `ConvertFromUnicodeToTextRun` attempts to convert the Unicode text to the single encoding it chooses from the list of encodings in the Unicode mapping structures array that you provide when you create the Unicode converter object. This is the encoding that produces the best result, that is, that provides for the greatest amount of source text conversion. If the complete source text can be converted into more than one of the encodings specified in the Unicode mapping structures array, then the converter chooses among them based on their order in the array. If this flag is clear, the `oEncodingRuns` parameter always points to a value equal to 1.

If you set the use-fallbacks control flag, the converter uses the default fallback characters for the current encoding. If the converter cannot handle a character using the current encoding, even using fallbacks, the converter attempts to convert the character using the other encodings, beginning with the first encoding specified in the list and skipping the encoding where it failed.

If you set the `kUnicodeTextRunBit` control flag, the converter attempts to convert the complete Unicode text string into the first encoding specified in the Unicode mapping structures array you passed to `CreateUnicodeToTextRunInfo`, `CreateUnicodeToTextRunInfoByEncoding`, or `CreateUnicodeToTextRunInfoByScriptCod`e when you created the Unicode converter object for this conversion. If it cannot do this, the converter then attempts to convert the first text element that failed to the remaining encodings, in their specified order in the array. What the converter does with the next text element depends on the setting of the keep-same-encoding control flag.

If the keep-same-encoding control flag is clear, the converter returns to the original encoding and attempts to continue conversion with that encoding; this is equivalent to converting each text element to the first encoding that works, in the order specified.

If the keep-same-encoding control flag is set, the converter continues with the new destination encoding until it encounters a text element that cannot be converted using the new encoding. This attempts to minimize the number of encoding changes in the output text. When the converter cannot convert a text element using any of the encodings in the list and the Unicode-keep-same-encoding control flag is set, the converter uses the fallbacks default characters for the current encoding.

iOffsetCount

The number of offsets in the array pointed to by the `iOffsetArray` parameter. Your application supplies this value. If you don't want offsets returned to you, specify `0` (zero) for this parameter.

iOffsetArray

An array of type `ByteOffset`. On input, you specify the array that contains an ordered list of significant byte offsets pertaining to the source Unicode string. These offsets may identify font or style changes, for example, in the Unicode string. If you don't want offsets returned to your application, specify `NULL` for this parameter and `0` (zero) for `iOffsetCount`. All offsets must be less than `iUnicodeLen`.

oOffsetCount

On return, a pointer to the number of offsets that were mapped in the output stream.

oOffsetArray

An array of type `ByteOffset`. On return, this array contains the corresponding new offsets for the resulting converted string.

iOutputBufLen

The length in bytes of the output buffer pointed to by the `oOutputStr` parameter. Your application supplies this buffer to hold the returned converted string. The `oOutputLen` parameter may return a byte count that is less than this value if the converted byte string is smaller than the buffer size you allocated.

oInputRead

On return, a pointer to the number of bytes of the Unicode source string that were converted. If the function returns a result code other than `noErr`, then this parameter returns the number of bytes that were converted before the error occurred.

oOutputLen

On return, a pointer to the length in bytes of the converted string.

oOutputStr

A value of type `LogicalAddress`. On input, this value points to the start of the buffer for the converted string. On output, this buffer contains the converted string in one or more encodings. When an error occurs, the `ConvertFromUnicodeToTextRun` function returns the converted string up to the character that caused the error. (For guidelines on estimating the size of the buffer needed, see the discussion following the parameter descriptions.

iEncodingRunBufLen

The number of text encoding run elements you allocated for the encoding run array pointed to by the `oEncodingRuns` parameter. The converter returns the number of valid encoding runs in the location pointed to by `oEncodingRunOutLen`. Each entry in the encoding runs array specifies the beginning offset in the converted text and its associated text encoding.

oEncodingRunOutLen

On return, a pointer to a the number of valid encoding runs returned in the `oEncodingRuns` parameter.

oEncodingRuns

On input, an array of structures of type `TextEncodingRun`. Your application should allocate an array with the number of elements you specify in the `iEncodingRunBufLen` parameter. On return, this array contains the encoding runs for the converted text string. Each entry in the encoding run array specifies the beginning offset in the converted text string and the associated encoding specification.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion

To use the `ConvertFromUnicodeToTextRun` function, you must first set up an array of structures of type `UnicodeMapping` (page 107) containing, in order of precedence, the mapping information for the conversion. To create a Unicode converter object, you call the `CreateUnicodeToTextRunInfo` function passing it the Unicode mapping array, or you can the `CreateUnicodeToTextRunInfoByEncoding` or `CreateUnicodeToTextRunInfoByScriptCode` functions, which take arrays of text encodings or script codes instead of an array of Unicode mappings. You pass the returned Unicode converter object as the `iUnicodeToTextInfo` parameter when you call the `ConvertFromUnicodeToTextRun` function.

Two of the control flags that you can set for the `iControlFlags` parameter allow you to control how the Unicode Converter uses the multiple encodings in converting the text string. These flags are explained in the description of the `iControlFlags` parameter. Here is a summary of how to use these two control flags:

- To keep the converted text in a single encoding, clear the text-run control flag.
- To keep as much contiguous converted text as possible in one encoding, set the text-run control flag and clear the keep-same-encoding control flag.
- To minimize the number of resulting encoding runs and the changes of destination encoding, set both the text-run and keep-same-encoding control flags.

The `ConvertFromUnicodeToTextRun` function returns the converted string in the array pointed to by the `oOutputStr` parameter. Beginning with the first text element in the `oOutputStr` array, the elements of the array pointed to by the `oEncodingRuns` parameter identify the encodings of the converted string. The

number of elements in the `oEncodingRuns` array may not correspond to the number of elements in the `oOutputStr` array. This is because the `oEncodingRuns` array includes only elements for the beginning of each new encoding run in the converted string.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

CountUnicodeMappings

Counts available mappings that meet the specified matching criteria.

```
OSStatus CountUnicodeMappings (
   OptionBits iFilter,
   ConstUnicodeMappingPtr iFindMapping,
   ItemCount *oActualCount
);
```

Parameters

iFilter

Filter control flags representing the six subfields of the Unicode mapping structure that this function uses to match against in determining which mappings on the system to return to your application. The filter control enumeration, described in "Unicode Matching Masks" (page 120), define the constants for the subfield's flags and their masks. You can include in the search criteria any of the three text encoding subfields for both the Unicode encoding and the other specified encoding. For any flag not turned on, the subfield value is ignored and the function does not check the corresponding subfield of the mappings on the system.

iFindMapping

A structure of type `UnicodeMapping` (page 107) containing the text encodings whose field values are to be matched.

oActualCount

On return, a pointer to the number of matching mappings found.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
You can filter on any of the three text encoding subfields of the Unicode mapping structure's `unicodeEncoding` specification and on any of the three text encoding subfields of the structure's `otherEncoding` specification. The `iFilter` parameter consists of a set of six control flags that you set to identify which of the corresponding six subfields to include in the match count. No filtering is performed on fields for which you do not set the corresponding filter control flag.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

CreateTextEncoding

Creates and returns a text encoding specification.

```
TextEncoding CreateTextEncoding (
   TextEncodingBase encodingBase,
   TextEncodingVariant encodingVariant,
   TextEncodingFormat encodingFormat
);
```

Parameters

encodingBase

A base text encoding.

encodingVariant

A variant of the base text encoding. To specify the default variant for the base encoding given in the `encodingBase` parameter, you can use the `kTextEncodingDefaultVariant` constant.

encodingFormat

A format for the base text encoding. To specify the default format for the base encoding, you can use the `kTextEncodingDefaultFormat` constant. If you want to obtain a `TextEncoding` value that references UTF-16 or UTF-8, pass `kUnicode16BitFormat` or `kUnicodeUTF8Format`.

Return Value

The text encoding specification that the function creates from the values you pass it.

Discussion

When you create a text encoding specification, the three values that you specify are packed into an unsigned integer, which you can then pass by value to the functions that use text encodings. See the data type `TextEncodingRun` (page 105).

Availability

Available in CarbonLib 1.0 and later when Text Common 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Declared In

`TextCommon.h`

CreateTextToUnicodeInfo

Creates and returns a Unicode converter object containing information required for converting strings from a non-Unicode encoding to Unicode.

```
OSStatus CreateTextToUnicodeInfo (
   ConstUnicodeMappingPtr iUnicodeMapping,
   TextToUnicodeInfo *oTextToUnicodeInfo
);
```

Parameters

iUnicodeMapping

A pointer to a structure of type `UnicodeMapping`. Your application provides this structure to identify the mapping to use for the conversion. You must supply a value of type `TextEncoding` in the `unicodeEncoding` field of this structure. A `TextEncoding` is a triple composed of an encoding base, an encoding variant, and a format. You can obtain a `UnicodeMapping` (page 107) value by calling the function `CreateTextEncoding`.

oTextToUnicodeInfo

On return, the Unicode converter object holds mapping table information you supplied as the `UnicodeMapping` parameter and state information related to the conversion. This information is required for conversion of a text stream in a non-Unicode encoding to Unicode.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion

You pass a Unicode converter object returned from the function CreateTextToUnicodeInfo to the function `ConvertFromTextToUnicode` (page 17) or `ConvertFromPStringToUnicode` (page 16) to identify the information to be used for the conversion. These two functions modify the contents of the object.

You pass a Unicode converter object returned from `CreateTextToUnicodeInfo` to the function `TruncateForTextToUnicode` (page 74) to identify the information to be used to truncate the string. This function does not modify the contents of the Unicode converter object.

If an error is returned, the Unicode converter object is invalid.

Availability

Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`UnicodeConverter.h`

CreateTextToUnicodeInfoByEncoding

Based on the given text encoding specification, creates and returns a Unicode converter object containing information required for converting strings from the specified non-Unicode encoding to Unicode.

```
OSStatus CreateTextToUnicodeInfoByEncoding (
   TextEncoding iEncoding,
   TextToUnicodeInfo *oTextToUnicodeInfo
);
```

Parameters

iEncoding

The text encoding specification for the source text.

oTextToUnicodeInfo

The Unicode converter object of type `TextToUnicodeInfo` (page 106) returned by the function.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
You can use this function instead of the `CreateTextToUnicodeInfo` (page 30) function when you do not need to create a Unicode mapping structure. You simply specify the text encoding of the source text. However, this method is less efficient because the text encoding parameter must be resolved internally into a Unicode mapping.

You cannot specify a version of Unicode. The function uses a 16-bit form of Unicode as the default.

You pass a Unicode converter object returned from `CreateTextToUnicodeInfoByEncoding` to the function `ConvertFromTextToUnicode` (page 17) or `ConvertFromPStringToUnicode` (page 16) to identify the information to be used for the conversion. These two functions modify the contents of the Unicode converter object.

You pass a Unicode converter object returned from `CreateTextToUnicodeInfoByEncoding` to the function `TruncateForTextToUnicode` (page 74) to identify the information to be used to truncate the string. This function does not modify the contents of the Unicode converter object.

If you are converting the text stream to Unicode as an intermediary encoding, and then from Unicode to the final destination encoding, you use the function `CreateUnicodeToTextInfo` (page 32) to create a Unicode converter object for the second part of the process.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Declared In
`UnicodeConverter.h`

CreateUnicodeToTextInfo

Creates and returns a Unicode converter object containing information required for converting strings from Unicode to a non-Unicode encoding.

```
OSStatus CreateUnicodeToTextInfo (
   ConstUnicodeMappingPtr iUnicodeMapping,
   UnicodeToTextInfo *oUnicodeToTextInfo
);
```

Parameters

iUnicodeMapping

A pointer to a structure of type `UnicodeMapping` (page 107). Your application provides this structure to identify the mapping to be used for the conversion. The `unicodeEncoding` field of this structure can specify a Unicode format of `kUnicode16BitFormat` or `kUnicodeUTF8Format`. Note that the versions of the Unicode Converter prior to 1.2.1 do not support `kUnicodeUTF8Format`.

oUnicodeToTextInfo

On return, a pointer to a Unicode converter object that holds the mapping table information you supply as the `iUnicodeMapping` parameter and the state information related to the conversion. The information contained in the Unicode converter object is required for the conversion of a Unicode string to a non-Unicode encoding.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
You pass the Unicode converter object returned from `CreateUnicodeToTextInfo` to the function `ConvertFromUnicodeToText` (page 23) or `ConvertFromUnicodeToPString` (page 19) to identify the information to be used for the conversion. These two functions modify the contents of the Unicode converter object.

If an error is returned, the Unicode converter object is invalid.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

CreateUnicodeToTextInfoByEncoding

Based on the given text encoding specification for the converted text, creates and returns a Unicode converter object containing information required for converting strings from Unicode to the specified non-Unicode encoding.

```
OSStatus CreateUnicodeToTextInfoByEncoding (
   TextEncoding iEncoding,
   UnicodeToTextInfo *oUnicodeToTextInfo
);
```

Parameters

iEncoding

The text encoding specification for the destination, or converted, text.

oUnicodeToTextInfo

A pointer to a Unicode converter object of type `UnicodeToTextInfo` (page 109).

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
You can use this function instead of the `CreateUnicodeToTextInfo` (page 32) function to create a Unicode converter. However, this method is less efficient internally because the destination text encoding you specify must be resolved into a Unicode mapping. Using this function, you cannot specify a version of Unicode, so a default version of Unicode is used; 16-bit format is assumed.

You pass a Unicode converter object returned from the function `CreateUnicodeToTextInfoByEncoding` to the function `ConvertFromUnicodeToText` (page 23) or `ConvertFromUnicodeToPString` (page 19) to identify the information to be used for the conversion. These two functions modify the contents of the Unicode converter object.

You pass a Unicode converter object returned from `CreateUnicodeToTextInfoByEncoding` to the function `TruncateForUnicodeToText` (page 75) to identify the information to be used to truncate the string. This function does not modify the contents of the Unicode converter object.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

CreateUnicodeToTextRunInfo

Creates and returns a Unicode converter object containing the information required for converting a Unicode text string to strings in one or more non-Unicode encodings.

```
OSStatus CreateUnicodeToTextRunInfo (
   ItemCount iNumberOfMappings,
   const UnicodeMapping iUnicodeMappings[],
   UnicodeToTextRunInfo *oUnicodeToTextInfo
);
```

Parameters

`iNumberOfMappings`

The number of mappings specified by your application for converting from Unicode to any other encoding types, including other forms of Unicode. If you pass 0 for this parameter, the converter will use all of the scripts installed in the system. The primary script is the one with highest priority; `ScriptOrder` (`'itlm'` resource) determines the priority of the rest. If you set the high-order bit for this parameter, the Unicode converter assumes that the `iEncodings` parameter contains a single element specifying the preferred encoding. This feature is supported for versions 1.2 or later of the converter.

`iUnicodeMappings`

A pointer to an array of structures of type `UnicodeMapping` (page 107). Your application provides this structure to identify the mappings to be used for the conversion. The order in which you specify the mappings determines the priority of the destination encodings. For this function, the Unicode mapping structure can specify a Unicode format of `kUnicode16BitFormat` or `kUnicodeUTF8Format`. Note that the versions of the Unicode Converter prior to the Text Encoding Conversion Manager 1.2.1 do not support `kUnicodeUTF8Format`. Also, note that the `unicodeEncoding` field should be the same for all of the entries in `iUnicodeMappings`. If you pass `NULL` for the `iUnicodeMappings` parameter, the converter uses all of the scripts installed in the system, assuming the default version of Unicode with 16-bit format. The primary script is the one with the highest priority and `ScriptOrder`(`'itlm'` resource) determines the priority of the rest. This is supported beginning with version 1.2 of the Text Encoding Conversion Manager.

`oUnicodeToTextInfo`

A pointer to a Unicode converter object for converting Unicode text strings to strings in one or more non-Unicode encodings. On return, a pointer to a Unicode converter object that holds the mapping table information you supply as the `iUnicodeMappings` parameter and the state information related to the conversion.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
You pass a Unicode converter object returned from the function `CreateUnicodeToTextRunInfo` to the function `ConvertFromUnicodeToTextRun` (page 25) or `ConvertFromUnicodeToScriptCodeRun` (page 20) to identify the information to be used for the conversion. These two functions modify the contents of the Unicode converter object.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

CreateUnicodeToTextRunInfoByEncoding

Based on the given text encoding specifications for the converted text runs, creates and returns a Unicode converter object containing information required for converting strings from Unicode to one or more specified non-Unicode encodings.

```
OSStatus CreateUnicodeToTextRunInfoByEncoding (
   ItemCount iNumberOfEncodings,
   const TextEncoding iEncodings[],
   UnicodeToTextRunInfo *oUnicodeToTextInfo
);
```

Parameters

iNumberOfEncodings

The number of desired encodings. If you pass 0 for this parameter, the converter will use all of the scripts installed in the system. The primary script is the one with highest priority; `ScriptOrder`('`itlm`' resource) determines the priority of the rest. If you set the high-order bit for this parameter, the Unicode converter assumes that the `iEncodings` parameter contains a single element specifying the preferred encoding. This feature is supported for versions 1.2 or later of the converter.

iEncodings

An array of text encoding specifications for the desired encodings. Your application provides this structure to identify the encodings to be used for the conversion. The order in which you specify the encodings determines the priority of the destination encodings. If you pass `NULL` for this parameter, the converter will use all of the scripts installed in the system. The primary script is the one with highest priority and `ScriptOrder`('`itlm`' resource) determines the priority of the rest.This feature is supported for versions 1.2 or later of the converter.

oUnicodeToTextInfo

A pointer to a Unicode converter object for converting Unicode text strings to strings in one or more non-Unicode encodings. On return, a pointer to a Unicode converter object that holds the encodings you supply as the `iEncodings` parameter and the state information related to the conversion.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
You pass a Unicode converter object returned from `CreateUnicodeToTextRunInfoByEncoding` to the function `ConvertFromUnicodeToTextRun` (page 25) or `ConvertFromUnicodeToScriptCodeRun` (page 20) to identify the information to be used for the conversion. These two functions modify the contents of the Unicode converter object.

If an error is returned, the converter object is invalid.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

CreateUnicodeToTextRunInfoByScriptCode

Based on the given script codes for the converted text runs, creates and returns a Unicode converter object containing information required for converting strings from Unicode to one or more specified non-Unicode encodings.

```
OSStatus CreateUnicodeToTextRunInfoByScriptCode (
   ItemCount iNumberOfScriptCodes,
   const ScriptCode iScripts[],
   UnicodeToTextRunInfo *oUnicodeToTextInfo
);
```

Parameters

iNumberOfScriptCodes

The number of desired scripts. If you pass 0 for this parameter, the converter uses all the scripts installed in the system. In this case, the primary script is the one with highest priority; `ScriptOrder` (`'itlm'` resource) determines the priority of the rest. If you set the high-order bit for this parameter, the Unicode converter assumes that the `iScripts` parameter contains a single element specifying the preferred script. This feature is supported beginning with the Text Encoding Conversion Manager 1.2.

iScripts

An array of script codes for the desired scripts. Your application provides this structure to identify the scripts to be used for the conversion. The order in which you specify the scripts determines their priority. If you pass `NULL` for this parameter, the converter uses all of the scripts installed in the system. In this case, the primary script is the one with the highest priority and the priority order of the remaining scripts is defined by the `ScriptOrder(itlm resource)` resource. This feature is supported for versions 1.2 or later of the converter.

oUnicodeToTextInfo

A pointer to a Unicode converter object for converting Unicode text strings to strings in one or more non-Unicode encodings. On return, a pointer to Unicode converter object that holds the scripts you supply as the `iScripts` parameter and the state information related to the conversion.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
You pass a Unicode converter object returned from `CreateUnicodeToTextRunInfoByScriptCode` to the function `ConvertFromUnicodeToTextRun` (page 25) or `ConvertFromUnicodeToScriptCodeRun` (page 20) to identify the information to be used for the conversion. These two functions modify the contents of the Unicode converter object.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Declared In
`UnicodeConverter.h`

DisposeTextToUnicodeInfo

Releases the memory allocated for the specified Unicode converter object.

```
OSStatus DisposeTextToUnicodeInfo (
   TextToUnicodeInfo *ioTextToUnicodeInfo
);
```

Parameters

ioTextToUnicodeInfo

A pointer to a Unicode converter object of type `TextToUnicodeInfo` (page 106), used for converting text to Unicode. On input, you specify the object to dispose. It must be an object which your application created using the function `CreateTextToUnicodeInfo` (page 30) or `CreateTextToUnicodeInfoByEncoding` (page 31). You must not point to any other type of Unicode converter object. Your application should not use this function with the same structure more than once.

Return Value
A result code. See "TEC Manager Result Codes" (page 166). If your application specifies an invalid Unicode converter object, such as `NULL`, the function returns a `paramErr` result code.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

DisposeUnicodeToTextFallbackUPP

Disposes of a a new universal procedure pointer (UPP) to a Unicode-to-text fallback callback.

```
void DisposeUnicodeToTextFallbackUPP (
   UnicodeToTextFallbackUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion
See the callback `UnicodeToTextFallbackProcPtr` (page 93) for more information.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

DisposeUnicodeToTextInfo

Releases the memory allocated for the specified Unicode converter object.

```
OSStatus DisposeUnicodeToTextInfo (
   UnicodeToTextInfo *ioUnicodeToTextInfo
);
```

Parameters

ioUnicodeToTextInfo

A pointer to a Unicode converter object for converting from Unicode to a non-Unicode encoding. You specify a Unicode converter object that your application created using the function `CreateUnicodeToTextInfo` (page 32) or `CreateUnicodeToTextInfoByEncoding` (page 33). You must not point to any other type of Unicode converter object. Your application should not attempt to dispose of the same Unicode converter object more than once.

Return Value

A result code. See "TEC Manager Result Codes" (page 166). The function returns `noErr` if it disposes of the Unicode converter object successfully. If your application specifies an invalid Unicode converter object, such as `NULL`, the function returns a `paramErr` result code.

Availability

Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`UnicodeConverter.h`

DisposeUnicodeToTextRunInfo

Releases the memory allocated for the specified Unicode converter object.

```
OSStatus DisposeUnicodeToTextRunInfo (
   UnicodeToTextRunInfo *ioUnicodeToTextRunInfo
);
```

Parameters

ioUnicodeToTextRunInfo

A pointer to a Unicode converter object. On input, you specify a Unicode converter object that points to the conversion information to dispose. It must be an object which your application created using the function `CreateUnicodeToTextRunInfo` (page 34), `CreateUnicodeToTextRunInfoByEncoding` (page 35), or `CreateUnicodeToTextRunInfoByScriptCode` (page 36). You must point to any other type of Unicode converter object. Your application should not use this function with the same structure more than once.

Return Value

A result code. See "TEC Manager Result Codes" (page 166). If your application specifies an invalid Unicode converter object, such as `NULL`, the function returns `paramErr`.

Availability

Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
UnicodeConverter.h

GetTextEncodingBase

Returns the base encoding of the specified text encoding.

```
TextEncodingBase GetTextEncodingBase (
   TextEncoding encoding
);
```

Parameters

encoding

A text encoding specification whose base encoding you want to obtain.

Return Value
The base encoding portion of the specified text encoding.

Discussion
See the data type TextEncodingRun (page 105)

Availability
Available in CarbonLib 1.0 and later when Text Common 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
TextCommon.h

GetTextEncodingFormat

Returns the format value of the specified text encoding.

```
TextEncodingFormat GetTextEncodingFormat (
   TextEncoding encoding
);
```

Parameters

encoding

A text encoding specification.

Return Value
The text encoding format value contained in the text encoding you specified.

Availability
Available in CarbonLib 1.0 and later when Text Common 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
TextCommon.h

GetTextEncodingName

Returns the localized name for a specified text encoding.

```
OSStatus GetTextEncodingName (
   TextEncoding iEncoding,
   TextEncodingNameSelector iNamePartSelector,
   RegionCode iPreferredRegion,
   TextEncoding iPreferredEncoding,
   ByteCount iOutputBufLen,
   ByteCount *oNameLength,
   RegionCode *oActualRegion,
   TextEncoding *oActualEncoding,
   TextPtr oEncodingName
);
```

Parameters

iEncoding

A text encoding specification whose name you want to obtain.

iNamePartSelector

The portion of the encoding name you want to obtain. See "Text Encoding Name Selectors" (page 152) for a list of possible values.

iPreferredRegion

The preferred region to use for the name. You can specify a Mac OS region code (which also implies a language) for this parameter. If the function cannot return the name for the preferred region, it returns the name using a region code with the same language or in a default language (for example, English).

iPreferredEncoding

The preferred encoding to use for the name. For example, ASCII, Mac OS Roman, or Shift-JIS. If the function cannot return the name using the preferred encoding, it returns the name using another encoding, such as Unicode or ASCII.

iOutputBufLen

The length in bytes of the output buffer that your application provides for the returned encoding name.

oNameLength

A pointer to a value of type `ByteCount`. On return, this parameter holds the actual length, in bytes, of the text encoding name. The value represents the full length of the name, which might be greater than the size of the output buffer, specified by the `iOutputBufLen` parameter. The length of the portion of the name actually contained in the output buffer is the smaller of `oNameLength` and `iOutputBufLen`.

oActualRegion

A pointer to a value of type `RegionCode`. On return, this parameter holds the actual region associated with the returned encoding name.

oActualEncoding

A pointer to a value of type `TextEncoding`. On return, this parameter holds the actual encoding associated with the returned encoding name.

oEncodingName

A pointer to a buffer you provide. On return, this parameter holds the text encoding name.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion
Names returned by `GetTextEncodingName` (in the buffer referred to by `oEncodingName`) can contain parentheses and other menu item meta characters, and so cannot be used with `AppendMenu` or `InsertMenuItem`. You can use them with `SetMenuItemText`.

This function can return resources and memory errors, and the following result codes:

- `kTextUnsupportedEncodingErr`, which indicates that the encoding whose name you want to obtain is not supported.
- `kTECMissingTableErr`, which indicates the name resource associated with the encoding is missing.
- `kTECTableFormatErr` or `kTECTableCheckSumErr`, which indicates that the name resource associated with that encoding is invalid.

Availability
Available in CarbonLib 1.0 and later when Text Common 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextCommon.h`

GetTextEncodingVariant

Returns the variant from the specified text encoding.

```
TextEncodingVariant GetTextEncodingVariant (
   TextEncoding encoding
);
```

Parameters
encoding
A text encoding specification.

Return Value
The text encoding variant portion of the specified text encoding.

Availability
Available in CarbonLib 1.0 and later when Text Common 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Declared In
`TextCommon.h`

InvokeUnicodeToTextFallbackUPP

Calls your Unicode-to-text fallback callback.

```
OSStatus InvokeUnicodeToTextFallbackUPP (
   UniChar *iSrcUniStr,
   ByteCount iSrcUniStrLen,
   ByteCount *oSrcConvLen,
   TextPtr oDestStr,
   ByteCount iDestStrLen,
   ByteCount *oDestConvLen,
   LogicalAddress iInfoPtr,
   ConstUnicodeMappingPtr iUnicodeMappingPtr,
   UnicodeToTextFallbackUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeUnicodeToTextFallbackUPP`, as the system calls your Unicode-to-text fallback callback for you. See the callback `UnicodeToTextFallbackProcPtr` (page 93) for more information.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

NearestMacTextEncodings

Obtains the best and alternate Mac text encoding.

```
OSStatus NearestMacTextEncodings (
   TextEncoding generalEncoding,
   TextEncoding *bestMacEncoding,
   TextEncoding *alternateMacEncoding
);
```

Parameters

`generalEncoding`
The text encoding for which you want to obtain a Mac text encoding.

`bestMacEncoding`
On return, the Mac text encoding that best matches the encoding specified by the `generalEncoding` parameter.

`alternateMacEncoding`
On return, the Mac text encoding that is the second best match for the encoding specified by the `generalEncoding` parameter.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Availability
Available in CarbonLib 1.0 and later when Text Common 1.5 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextCommon.h`

NewUnicodeToTextFallbackUPP

Creates a new universal procedure pointer (UPP) to a Unicode-to-text fallback callback.

```
UnicodeToTextFallbackUPP NewUnicodeToTextFallbackUPP (
   UnicodeToTextFallbackProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your Unicode-to-text fallback callback.

Return Value

On return, a UPP to the Unicode-to-text fallback callback.

Discussion

See the callback `UnicodeToTextFallbackProcPtr` (page 93) for more information.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Declared In

`UnicodeConverter.h`

QueryUnicodeMappings

Returns a list of the conversion mappings available on the system that meet specified matching criteria and returns the number of mappings found.

```
OSStatus QueryUnicodeMappings (
   OptionBits iFilter,
   ConstUnicodeMappingPtr iFindMapping,
   ItemCount iMaxCount,
   ItemCount *oActualCount,
   UnicodeMapping oReturnedMappings[]
);
```

Parameters

iFilter

Filter control flags representing the six values given in the Unicode mapping structure that this function uses to match against in determining which mappings on the system to return to your application. The filter control flag enumerations, described in "Unicode Matching Masks" (page 120), define the constants for the flags and their masks. You can include in the search criteria any of the three text encoding values—base, variant, and format—for both the Unicode encoding and the other specified encoding. For any flag not turned on, the value is ignored the function does not check the corresponding value of the mapping tables on the system.

iFindMapping

A structure of type `UnicodeMapping` (page 107) containing the text encodings whose values are to be matched.

iMaxCount

The maximum number of mappings that can be returned. You provide this value to identify the number of elements in the array pointed to by the `oReturnedMappings` parameter that your application allocated. If the function identifies more matching mappings than the array can hold, it returns as many of them as fit. The function also returns a `kTECArrayFullErr` in this case.

oActualCount

On return, a pointer to the number of matching mappings found. This number may be greater than the number of mappings specified by `iMaxCount` if more matching mappings are found than can fit in the `oReturnedMappings` array.

oReturnedMappings

A pointer to an array of structures of type `UnicodeMapping` (page 107). On input, this pointer refers to an array for the matching mappings returned by the function. To allocate sufficient elements for the array, you can use the function `CountUnicodeMappings` (page 29) to determine the number of mappings returned for given values of the `iFilter` and `iFindMapping` parameters. On return, a pointer to an array that holds the matching mappings. If there are more matches than the array can hold, the function returns as many of them as will fit and a `kTECBufferBelowMinimumSizeErr` error result. The `oActualCount` parameter identifies the number of matching mappings actually found, which may be greater than the number returned.

Return Value

A result code. See "TEC Manager Result Codes" (page 166). If the function returns a `noErr` result code, the value retuned in the `oActualCount` parameter is less than or equal to the value returned in the `iMaxCount` parameter and the `oReturnedMappings` parameter contains all of the matching mappings found. If the function returns a `kTECArrayFullErr`, the function found more mappings than your `oReturnedMappings` array could accommodate.

Discussion

You can use the `QueryUnicodeMappings` function to obtain all mappings on the system up to the number allowed by your `oReturnedMappings` array by specifying a value of zero for the `iFilter` field.

You can use the function to obtain very specific mappings by setting individual filter control flags. You can filter on any of the three text encoding subfields of the Unicode mapping structure's `unicodeEncoding` specification and on any of the three text encoding subfields of the mapping's `otherEncoding` specification. The `iFilter` parameter consists of a set of six control flags that you set to identify which of the corresponding six subfields to include in the match. The list provided in the `oReturnedMappings` parameter will contain only mappings that match the fields of the Unicode mapping structure whose text encodings subfields you identify in the filter control flags. No filtering is performed on subfields for which you do not set the corresponding filter control flag.

For example, to obtain a list of all mappings in which one of the encodings is the default variant and default format of the Unicode 1.1 base encoding and the other encoding is the default variant and default format of a base encoding other than Unicode, you would set up the `iFilter` and `iFindMappings` parameter as follows. To set up these parameters, you use the constants defined for the text encoding bases, the text encoding default variants, the text encoding default formats, and the filter control flag bitmasks. In this example, the text encoding base field of the Unicode mapping structure's `otherEncoding` field is ignored, so you can specify any value for it. When you call `QueryUnicodeMappings`, passing it these parameters, the function will return a list of mappings between the Unicode encoding you specified and every other available encoding in which each non-Unicode base encoding shows up once because you specified its default variant and default format.

```
iFindMapping.unicodeMapping = CreateTextEncoding(
kTextEncodingUnicodeV1_1,
kTextEncodingDefaultVariant,
kTextEncodingDefaultFormat);
```

```
iFindMapping.otherEncoding = CreateTextEncoding(
kTextEncodingMacRoman,
kTextEncodingDefaultVariant,
kTextEncodingDefaultFormat);
iFilter = kUnicodeMatchUnicodeBaseMask |
kUnicodeMatchUnicodeVariantMask |
kUnicodeMatchUnicodeFormatMask |
kUnicodeMatchOtherVariantMask |
kUnicodeMatchOtherFormatMask;
```

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

ResetTextToUnicodeInfo

Reinitializes all state information kept by the context objects.

```
OSStatus ResetTextToUnicodeInfo (
   TextToUnicodeInfo ioTextToUnicodeInfo
);
```

Parameters
`ioTextToUnicodeInfo`

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.3 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

ResetUnicodeToTextInfo

Reinitializes all state information kept by a Unicode converter object.

```
OSStatus ResetUnicodeToTextInfo (
   UnicodeToTextInfo ioUnicodeToTextInfo
);
```

Parameters
`ioUnicodeToTextInfo`

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Declared In
`UnicodeConverter.h`

ResetUnicodeToTextRunInfo

Reinitializes all state information kept by the context objects in TextRun conversions.

```
OSStatus ResetUnicodeToTextRunInfo (
   UnicodeToTextRunInfo ioUnicodeToTextRunInfo
);
```

Parameters

ioUnicodeToTextRunInfo

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

ResolveDefaultTextEncoding

Returns a text encoding specification in which any meta-values have been resolved to real values. Currently, this affects only the base encoding values packed into the text encoding specification.

```
TextEncoding ResolveDefaultTextEncoding (
   TextEncoding encoding
);
```

Parameters

encoding

A text encoding specification possibly containing meta-values that you want to resolve to a text encoding specification containing only real values.

Return Value
A text encoding specification containing only real base encoding values.

Discussion
This function is useful for application developers who are providing APIs that take text encoding specifications as parameters. All APIs in the Unicode Converter and Text Encoding Converter perform this translation automatically.

Availability
Available in CarbonLib 1.0 and later when Text Common 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextCommon.h`

RevertTextEncodingToScriptInfo

Converts the given Mac OS text encoding specification to the corresponding script code and, if possible, language code and font name.

```
OSStatus RevertTextEncodingToScriptInfo (
   TextEncoding iEncoding,
   ScriptCode *oTextScriptID,
   LangCode *oTextLanguageID,
   Str255 oTextFontname
);
```

Parameters

iEncoding

The text encoding specification to be converted.

oTextScriptID

A pointer to a value of type `ScriptCode`. On return, a Mac OS script code that corresponds to the text encoding specification you identified in the `iEncoding` parameter. If you do not pass a pointer for this parameter, the function returns a `paramErr` result code.

oTextLanguageID

A pointer to a value of type `LangCode`. On input, if you do not want the function to return the language code, specify `NULL` as the value of this parameter. On return, the appropriate language code, if the language can be unambiguously derived from the text encoding specification, for example, Japanese, and you did not set the parameter to `NULL`.

If you do not specify `NULL` on input and the language is ambiguous—that is, the function cannot accurately derive it from the text encoding specification—the function returns a value of `kTextLanguageDontCare`.

oTextFontname

A Pascal string. On input, if you do not want the function to return the font name, specify `NULL` as the value of this parameter. On return, the name of the appropriate font if the font can be unambiguously derived from the text encoding specification, for example, Symbol, and you did not set the parameter to `NULL`.

If you do not specify `NULL` on input and the font is ambiguous—that is, the function cannot accurately derive it from the text encoding specification—the function returns a zero-length string.

Return Value

A result code. See "TEC Manager Result Codes" (page 166). The function returns `paramErr` if the text encoding specification input parameter value is invalid. The function returns a `kTECTableFormatErr` result code if the internal mapping tables used for translation are invalid. For a list of other possible result codes, see "Data Types".

Discussion

If you have applications that use Mac OS Script Manager and Font Manager functions, you can use the `RevertTextEncodingToScriptInfo` function to convert information in a text encoding specification into the appropriate Mac OS script code, language code, and font name, if they can be unambiguously derived. Your application can then use this information to display text to a user on the screen.

For more information see the `UpgradeScriptInfoToTextEncoding` (page 77) function and "Base Text Encodings" (page 122).

Availability

Available in CarbonLib 1.0 and later when Text Common 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
TextCommon.h

SetFallbackUnicodeToText

Specifies a fallback handler to be used for converting a Unicode text segment to another encoding when the Unicode Converter cannot convert the text using the mapping table specified by the Unicode converter object.

```
OSStatus SetFallbackUnicodeToText (
   UnicodeToTextInfo iUnicodeToTextInfo,
   UnicodeToTextFallbackUPP iFallback,
   OptionBits iControlFlags,
   LogicalAddress iInfoPtr
);
```

Parameters

iUnicodeToTextInfo

\The Unicode converter object to which the fallback handler is to be associated. You use the function CreateUnicodeToTextInfo (page 32) or CreateUnicodeToTextInfoByEncoding (page 33) to obtain a Unicode converter object of this type.

iFallback

A universal procedure pointer to the application-defined fallback routine. For a description of the function prototype that your fallback handler must adhere to and how to create your own fallback handler, see UnicodeToTextFallbackProcPtr (page 93). You should use the NewUnicodeToTextFallbackProc macro to convert a pointer to your fallback handler into a UnicodeToTextFallbackUPP.

iControlFlags

Control flags that stipulate which fallback handler the Unicode Converter should call—the application-defined fallback handler or the default handler—if a fallback handler is required, and the sequence in which the Unicode Converter should call the fallback handlers if either can be used when the other fails or is unavailable. See "Fallback Handler Selectors" (page 122).

iInfoPtr

A point to a block of memory to be passed to the application-defined fallback handler. The Unicode Converter passes this pointer to the application-defined fallback handler as the last parameter when it calls the fallback handler. Your application can use this memory block to store data required by your fallback handler whenever it is called. This is similar in use to a reference constant (refcon). If you don't need to use a memory block, specify NULL for this parameter.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
You can define multiple fallback handlers and associate them with different Unicode converter objects, depending on your requirements. See UnicodeToTextFallbackProcPtr (page 93) for a description of how to create and install an application-defined fallback handler.

You can use a fallback handler when one of the Unicode conversion functions, ConvertFromUnicodeToText (page 23) , ConvertFromUnicodeToTextRun (page 25) , ConvertFromUnicodeToPString (page 19) , and ConvertFromUnicodeToScriptCodeRun (page 20) , cannot convert the text using the mapping table specified by the Unicode converter object passed to the function.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

SetFallbackUnicodeToTextRun

Specifies a fallback handler to be used for converting a Unicode text segment to another encoding when the Unicode Converter cannot convert the text using the mapping table specified by a Unicode converter object.

```
OSStatus SetFallbackUnicodeToTextRun (
   UnicodeToTextRunInfo iUnicodeToTextRunInfo,
   UnicodeToTextFallbackUPP iFallback,
   OptionBits iControlFlags,
   LogicalAddress iInfoPtr
);
```

Parameters

iUnicodeToTextRunInfo

The Unicode converter object to which the fallback handler is to be associated. You use the function `CreateUnicodeToTextRunInfo` (page 34), `CreateUnicodeToTextRunInfoByEncoding` (page 35), or `CreateUnicodeToTextRunInfoByScriptCode` (page 36) to obtain a Unicode converter object to specify for this parameter.

iFallback

A universal procedure pointer to the application-defined fallback routine. For a description of the function prototype to which your fallback handler must adhere and how to create your own fallback handler, see `UnicodeToTextFallbackProcPtr` (page 93). You should use the `NewUnicodeToTextFallbackProc` macro described in the discussion of the function `SetFallbackUnicodeToText` (page 48).

iControlFlags

Control flags that stipulate which fallback handler the Unicode Converter should call—the application-defined fallback handler or the default handler—if a fallback handler is required, and the sequence in which the Unicode Converter should call the fallback handlers if either can be used when the other fails or is unavailable. See "Fallback Handler Selectors" (page 122).

iInfoPtr

A pointer to a block of memory to be passed to the application-defined fallback handler. The Unicode Converter passes this pointer to the application-defined fallback handler as the last parameter when it calls the fallback handler. Your application can use this block to store data required by your fallback handler whenever it is called. This is similar in use to a reference constant (refcon). If you don't need to use a memory block, specify `NULL` for this parameter.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
You can define multiple fallback handlers and associate them with different Unicode converter objects, depending on your requirements. See `UnicodeToTextFallbackProcPtr` (page 93) for a description of how to create and install an application-defined fallback handler.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

TECClearConverterContextInfo

Resets a converter object to its initial state so you can reuse it.

```
OSStatus TECClearConverterContextInfo (
   TECObjectRef encodingConverter
);
```

Parameters

encodingConverter

A reference to the text encoding converter object you want to reset. It can be a reference returned by the `TECCreateConverter` (page 58), `TECCreateOneToManyConverter` (page 60), or `TECCreateConverterFromPath` (page 59) functions.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
It is more efficient to reuse an existing converter object than to create a new one that contains the same conversion information. This function clears the text string, but does not alter the source and destination encodings.

If you are converting multiple segments of a text string, you should not clear the converter object until you have converted all the text segments.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECClearSnifferContextInfo

Resets a sniffer object to its initial settings so you can reuse it.

```
OSStatus TECClearSnifferContextInfo (
   TECSnifferObjectRef encodingSniffer
);
```

Parameters

encodingSniffer

A pointer to the sniffer object you want to reset.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
Sniffers maintain state information about the input encoding buffer and the number of errors and features found for each encoding; this information allows a caller to progressively sniff an input buffer in sequential chunks. Before sniffing a buffer that contains completely new information you must clear any state information by calling `TECClearSnifferContextInfo`.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECConvertText

Converts a stream of text from a source encoding to a destination encoding. It uses the conversion path specified by the converter object you supply.

```
OSStatus TECConvertText (
   TECObjectRef encodingConverter,
   ConstTextPtr inputBuffer,
   ByteCount inputBufferLength,
   ByteCount *actualInputLength,
   TextPtr outputBuffer,
   ByteCount outputBufferLength,
   ByteCount *actualOutputLength
);
```

Parameters

encodingConverter
A reference to the text encoding converter object you want to use for the conversion. It can be a reference returned by the `TECCreateConverter` (page 58) or `TECCreateConverterFromPath` (page 59) functions.

inputBuffer
The stream of text you want to convert.

inputBufferLength
The length in bytes (`UInt8` or `unsigned char`) of the stream of text.

actualInputLength
On return, a pointer to the number of source text bytes that were converted from the input buffer.

outputBuffer
A pointer to a buffer for a byte stream. On output, the buffer holds the converted text.

outputBufferLength
The length in bytes of the `outputBuffer` parameter.

actualOutputLength
On return, a pointer to the number of bytes of converted text returned in the `outputBuffer` parameter.

Return Value
A result code. See "TEC Manager Result Codes" (page 166). If there is not enough memory available for `TECConvertText` to convert the text when allocating internal buffers, the function returns the appropriate Memory Manager result code.

Discussion
If the output buffer you allocate is too small to accommodate any of the converted text, the function fails. For best results, you should follow these guidelines when you allocate an output buffer:

- Base the buffer length on an estimate of the byte requirements of the destination encoding. Make sure you account for additional bytes needed by the destination encoding (for example, an escape sequence) in addition to the actual text.
- Always allocate a buffer at least 32 bytes long.
- If size is a concern, make sure the output buffer is at least large enough to hold a portion of the converted text. You can convert part of the text, then use the value of the `actualInputLength` parameter to identify the next byte to be taken and to determine how many bytes remain. To convert the remaining text, you simply call the function again with the remaining text and a new output buffer.
- If the destination encoding is a character encoding scheme—such as ISO-2022-JP, which begins in ASCII and switches to other coded character sets through limited combinations of escape sequences—then you need to allocate enough space to accommodate escape sequences that signal switches. ISO-2022-JP requires 3 to 5 bytes for an escape sequence preceding the 1-byte or 2-byte character it introduces. If you allocate a buffer that is less than 5 bytes, the TECConvertText function could fail, depending on the text being converted.

To make sure that you receive all of the converted text, you should call the function `TECFlushText` (page 64) when you are finished converting all the text in a text stream.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECConvertTextToMultipleEncodings

Converts text in the source encoding to runs of text in multiple destination encodings. It uses the conversion path specified in the converter object you supply.

```
OSStatus TECConvertTextToMultipleEncodings (
   TECObjectRef encodingConverter,
   ConstTextPtr inputBuffer,
   ByteCount inputBufferLength,
   ByteCount *actualInputLength,
   TextPtr outputBuffer,
   ByteCount outputBufferLength,
   ByteCount *actualOutputLength,
   TextEncodingRun outEncodingsBuffer[],
   ItemCount maxOutEncodingRuns,
   ItemCount *actualOutEncodingRuns
);
```

Parameters

encodingConverter

The reference to the text encoding converter object to be used for the conversion. This is the reference returned by the function `TECCreateOneToManyConverter` (page 60).

inputBuffer
The stream of text to be converted.

inputBufferLength
The length in bytes of the stream of text specified in the `inputBuffer` parameter.

actualInputLength
On return, a pointer to a the number of source text bytes that were converted.

outputBuffer
On return, a pointer to a buffer that holds the converted text.

outputBufferLength
The length in bytes of the `outputBuffer` parameter.

actualOutputLength
On return, a pointer to the number of bytes of the converted text returned in the `outputBuffer` parameter.

outEncodingsBuffer
An array of text encoding runs for output. Note that the actual byte size of this buffer should be `actualOutEncodingRuns * sizeof(TextEncodingRun)`.

maxOutEncodingRuns
The maximum number of runs that can fit in the `outEncodingsBuffer` array.

actualOutEncodingRuns
On return, a pointer to the number of runs in `outEncodingsBuffer` array.

Return Value
A result code. See "TEC Manager Result Codes" (page 166). If there is not enough memory available to convert the text when allocating internal buffers, the function returns the appropriate Memory Manager result code.

Discussion
For the function to return successfully, the output buffer you allocate must be large enough to accommodate the converted text. If the output buffer is too small to accommodate any converted text, the function will fail. For best results, you should follow these guidelines when you allocate an output buffer:

- Base the buffer length on an estimate of the byte requirements of the destination encoding. Make sure you account for additional bytes needed by the destination encoding (for example, an escape sequence) in addition to the actual text.
- Always allocate a buffer at least 32 bytes long.
- If size is a concern, make sure the output buffer is at least large enough to hold a portion of the converted text. You can convert part of the text, then use the value of the `actualInputLength` parameter to identify the next byte to be taken and to determine how many bytes remain. To convert the remaining text, you simply call the function again with the remaining text and a new output buffer.
- If the destination encoding is a character encoding scheme—such as ISO-2022-JP, which begins in ASCII and switches to other coded character sets through limited combinations of escape sequences—then you need to allocate enough space to accommodate escape sequences that signal switches. ISO-2022-JP requires 3 to 5 bytes for an escape sequence preceding the 1-byte or 2-byte character it introduces. If you allocate a buffer that is less than 5 bytes, the `TECConvertText` function could fail, depending on the text being converted.

The Text Encoding Converter creates internal buffers that hold intermediate results for indirect conversions

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
TextEncodingConverter.h

TECCountAvailableSniffers

Counts and returns the number of sniffers available in all installed plug-ins.

```
OSStatus TECCountAvailableSniffers (
   ItemCount *numberOfEncodings
);
```

Parameters

numberOfEncodings

On return, a pointer to the number of sniffers in all installed plug-ins. You can use this number to determine what size array to allocate for a parameter of the `TECGetAvailableSniffers` (page 65) function.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
This function counts every instance of a sniffer. If different conversion plug-ins support a sniffer for the same encoding, the sniffer is counted more than once. Since the `TECGetAvailableSniffers` function ignores duplicate sniffers, `TECCountAvailableSniffers` may return a number greater than the number of array elements needed for the `availableSniffers[]` parameter of the `TECGetAvailableSniffers` function.

Availability
Supported in Carbon. Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
TextEncodingConverter.h

TECCountAvailableTextEncodings

Counts and returns the number of text encodings currently configured in the Text Encoding Converter.

```
OSStatus TECCountAvailableTextEncodings (
   ItemCount *numberEncodings
);
```

Parameters

numberEncodings

On return, a pointer to the number of currently supported text encodings. You use this value to determine the array size for a parameter of the `TECGetAvailableTextEncodings` (page 66) function.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
The number of text encodings includes every instance of a text encoding. If different conversion plug-ins support the same text encoding, the text encoding will be counted more than once. For example, the Japanese Encodings plug-in supports Mac OS Japanese, and so does the Unicode Encodings plug-in. Since the `TECGetAvailableTextEncodings` function ignores duplicate text encoding specifications, `TECCountAvailableTextEncodings` may return a number greater than the number of array elements needed for the `availableEncodings []` parameter.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECCountDestinationTextEncodings

Counts and returns the number of destination encodings to which a specified source encoding can be converted in one step.

```
OSStatus TECCountDestinationTextEncodings (
   TextEncoding inputEncoding,
   ItemCount *numberOfEncodings
);
```

Parameters

`inputEncoding`
The text encoding specification describing the source text.

`numberOfEncodings`
On return, a pointer to the number of text encodings to which the source encoding can be converted in one step. You should use this to determine how large to make the array you pass to the `TECGetDestinationTextEncodings` (page 66) function.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
This function counts every instance of an encoding. If different conversion plug-ins support the same direct text encoding, the direct text encoding is counted more than once.

Since the `TECGetDestinationTextEncodings` function ignores duplicate text encoding specifications, `TECCountDestinationTextEncodings` may return a number greater than the number of array elements needed for the `destinationEncodings[]` parameter.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Declared In
`TextEncodingConverter.h`

TECCountDirectTextEncodingConversions

Counts and returns the number of direct conversions currently configured in the Text Encoding Converter.

```
OSStatus TECCountDirectTextEncodingConversions (
   ItemCount *numberOfEncodings
);
```

Parameters

numberOfEncodings

On return, a pointer to the number of direct conversions. You should use this value to determine the array size for a parameter of the `TECGetDirectTextEncodingConversions` (page 67) function.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion

The number of direct conversions includes every instance of a conversion. If different conversion plug-ins support the same direct conversion, the direct conversion is counted more than once.

Since the `TECGetDirectTextEncodingConversions` (page 67) function ignores duplicate direct conversions, `TECCountDirectTextEncodingConversions` may return a number greater than the number of array elements needed for the `directConversions` parameter.

Availability

Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`TextEncodingConverter.h`

TECCountMailTextEncodings

Counts and returns the number of currently supported e-mail encodings for a specified region.

```
OSStatus TECCountMailTextEncodings (
   RegionCode locale,
   ItemCount *numberEncodings
);
```

Parameters

locale

A Mac OS region code. A region code designates a combination of language, writing system, and geographic region; the region may not correspond to a particular country (for example, Swiss French or Arabic).

numberEncodings

On return, a pointer to the number of currently supported e-mail encodings for the region code. You use this number to determine what size array to allocate for a parameter of the `TECGetMailTextEncodings` (page 69) function.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion
This function counts every instance of an encoding. If different conversion plug-ins support the same direct text encoding, the direct text encoding is counted more than once. Since the `TECGetMailTextEncodings` function ignores duplicate text encoding specifications, `TECCountMailTextEncodings` may return a number greater than the number of array elements needed.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECCountSubTextEncodings

Counts and returns the number of subencodings a text encoding supports.

```
OSStatus TECCountSubTextEncodings (
   TextEncoding inputEncoding,
   ItemCount *numberOfEncodings
);
```

Parameters

inputEncoding
The text encoding specification that contains the subencodings.

numberOfEncodings
On return, a pointer to the number of currently supported subencodings. You use this value to determine the array size for a parameter of the `TECGetSubTextEncodings` (page 70) function.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
Subencodings are text encodings that are embedded as part of a larger text encoding specification. For example, EUC-JP contains JIS Roman or ASCII, JIS X0208, JIS X0212, and half-width Katakana from JIS X0201. Not every encoding that can be broken into multiple encodings necessarily supports this routine. It's up to the plug-in developer to decide which encodings might be useful to break up. Subencodings are not the same as text encoding variants.

If an encoding can be converted to multiple runs of encodings (as indicated by a destination base encoding of `kTextEncodingMultiRun`), you can call the `TECGetSubTextEncodings` function to get the list of output encodings. See the `TECCreateOneToManyConverter` (page 60) and `TECGetDestinationTextEncodings` (page 66) functions for information.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECCountWebTextEncodings

Counts and returns the number of currently supported text encodings for a region code.

```
OSStatus TECCountWebTextEncodings (
   RegionCode locale,
   ItemCount *numberEncodings
);
```

Parameters

locale

A Mac OS region code indicating the locale for which you want to count encodings. A region code designates a combination of language, writing system, and geographic region; the region may not correspond to a particular country (for example, Swiss French or Arabic).

numberEncodings

On return, a pointer to the number of currently supported text encodings for a region code. You should use this number to determine how large to make the array you pass to the `TECGetWebTextEncodings` (page 72) function.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion

This function counts every instance of the same encoding. That is, if different conversion plug-ins support the same text encoding for a conversion process, the text encoding is counted more than once. Since the `TECGetWebTextEncodings` function ignores duplicate text encoding specifications, `TECCountWebTextEncodings` may return a number greater than the number of array elements needed for the `availableEncodings[]` parameter.

Availability

Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`TextEncodingConverter.h`

TECCreateConverter

Determines a conversion path for a source and destination encoding, then creates a text encoding converter object and returns a pointer to it.

```
OSStatus TECCreateConverter (
   TECObjectRef *newEncodingConverter,
   TextEncoding inputEncoding,
   TextEncoding outputEncoding
);
```

Parameters

newEncodingConverter

A pointer to a converter object. On return, this reference points to a newly created text converter object.

inputEncoding

The text encoding specification for the source text encoding.

outputEncoding

The text encoding specification for the destination text encoding.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
You use this converter object reference with conversion functions such as `TECConvertText` (page 51) to convert text. This converter object describes the source, destination, and intermediate encodings; state information; and references to required plug-ins.

If the function does not find a direct conversion path, it creates an indirect conversion path. You can use the function `TECCreateConverterFromPath` (page 59) to specify an explicit conversion path.

You must use the `TECDisposeConverter` (page 61) function to remove a converter object.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECCreateConverterFromPath

Creates a converter object for a specific conversion path—from a source encoding through intermediate encodings to a destination encoding—and returns a pointer to it.

```
OSStatus TECCreateConverterFromPath (
   TECObjectRef *newEncodingConverter,
   const TextEncoding inPath[],
   ItemCount inEncodings
);
```

Parameters

newEncodingConverter

A pointer to a converter object reference. On return, the reference points to a newly created text converter object.

inPath

An ordered array of text encoding specifications, beginning with the source encoding specification and ending with the destination encoding specification. Each adjacent pair of text encodings must represent a conversion that is supported by the Text Encoding Converter.

inEncodings

The number of text encoding specifications in the `inPath` array.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
This function is faster than the function `TECCreateConverter` (page 58) since it does not need to search for a conversion path. You can use the `TECGetDestinationTextEncodings` (page 66) function to determine each step in the sequence from the source to the destination encoding.

To remove a converter object, you must call the function `TECDisposeConverter` (page 61).

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECCreateOneToManyConverter

Determines a conversion path for the source encoding and destinations encodings you specify, creates a text encoding converter object, and returns a reference to it.

```
OSStatus TECCreateOneToManyConverter (
   TECObjectRef *newEncodingConverter,
   TextEncoding inputEncoding,
   ItemCount numOutputEncodings,
   const TextEncoding outputEncodings[]
);
```

Parameters

newEncodingConverter
A pointer to a converter object. On return, this points to a newly created one-to-many converter object.

inputEncoding
The text encoding specification for the source text encoding.

numOutputEncodings
The number of text encoding specifications in the `outputEncoding` array.

outputEncodings
An ordered array of text encoding specifications for the destination text encodings.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
You use this converter object reference with conversion functions such as `TECConvertTextToMultipleEncodings` (page 52). The converter object describes the source, destination, and intermediate encodings; state information; and references to required plug-ins.

To remove a converter object, you must call the function `TECDisposeConverter` (page 61).

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECCreateSniffer

Creates a sniffer object and returns a reference to it.

```
OSStatus TECCreateSniffer (
   TECSnifferObjectRef *encodingSniffer,
   TextEncoding testEncodings[],
   ItemCount numTextEncodings
);
```

Parameters

encodingSniffer

A pointer to a sniffer object reference, which is of type `TECSnifferObjectRef` (page 104). On return, the reference pertains to the newly created sniffer object.

testEncodings

An array of text encoding specifications supplied by the caller; `TECCreateSniffer` creates a sniffer that can detect each of these encodings.

numTextEncodings

The number of text encoding specifications in the `testEncodings[]` array.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion

The `TECCreateSniffer` function polls plug-ins for available sniffers, creates a sniffer object capable of sniffing each of the specified encodings that it can find a sniffer function for, and returns a reference to it. You use this sniffer object reference with sniffer functions such as `TECSniffTextEncoding` (page 73). If no sniffer function is available for an encoding, no error is returned and `TECSniffTextEncoding` indicates later that the encoding was not examined.

To remove a sniffer object, you must call the function `TECDisposeSniffer` (page 62).

Availability

Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`TextEncodingConverter.h`

TECDisposeConverter

Disposes of a converter object.

```
OSStatus TECDisposeConverter (
   TECObjectRef newEncodingConverter
);
```

Parameters

newEncodingConverter

A reference to the text encoding converter object you want to remove. This can be the reference returned by the `TECCreateConverter` (page 58), `TECCreateConverterFromPath` (page 59), or `TECCreateOneToManyConverter` (page 60) functions.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion
If you want to reuse the converter object for a different text stream with the same source and destination encoding, you should clear the converter object using the `TECClearConverterContextInfo` (page 50) function rather than disposing of it and then creating a new converter object.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECDisposeSniffer

Disposes of a sniffer object.

```
OSStatus TECDisposeSniffer (
   TECSnifferObjectRef encodingSniffer
);
```

Parameters

encodingSniffer
The sniffer object reference you want to remove.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
This function releases all memory allocated to the sniffer object created by the `TECCreateSniffer` (page 60) function.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECFlushMultipleEncodings

Flushes out any encodings that may be stored in a converter object's temporary buffers and shifts encodings back to their default state, if any.

```
OSStatus TECFlushMultipleEncodings (
   TECObjectRef encodingConverter,
   TextPtr outputBuffer,
   ByteCount outputBufferLength,
   ByteCount *actualOutputLength,
   TextEncodingRun outEncodingsBuffer[],
   ItemCount maxOutEncodingRuns,
   ItemCount *actualOutEncodingRuns
);
```

Parameters

encodingConverter

The reference to the text encoding converter object whose contents are to be flushed. This is the reference returned by the function `TECCreateOneToManyConverter` (page 60).

outputBuffer

On return, a pointer to a buffer that holds the converted text. An error is returned is the buffer is not large enough to hold the entire converted text stream.

outputBufferLength

The length in bytes of the `outputBuffer` parameter.

actualOutputLength

On return, a pointer to a the actual number of bytes of the converted text returned in the `outputBuffer` parameter.

outEncodingsBuffer

An ordered array of text encoding runs for the destination text encoding. Note that the actual byte size of this buffer should be `actualOutEncodingRuns * sizeof(TextEncodingRun)`.

maxOutEncodingRuns

The maximum number of encoding runs that can fit in `outEncodingsBuffer`[].

actualOutEncodingRuns

On return, a pointer to a the number of runs in the buffer during conversion.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion

You should always call `TECFlushMultipleEncodings` at the end of the conversion process to flush out any data that may be stored in the temporary buffers of the text encoding converter object or to perform other end-of-encoding conversion tasks. Encodings such as ISO-2022-JP are reset to a default state when you use this function.

For the function to return successfully, the output buffer you allocate must be large enough to accommodate the converted text. If the output buffer is too small to accommodate any converted text, the function will fail. For best results, you should follow these guidelines when you allocate an output buffer:

- Base the buffer length on an estimate of the byte requirements of the destination encoding. Make sure you account for additional bytes needed by the destination encoding (for example, an escape sequence) in addition to the actual text.
- Always allocate a buffer at least 32 bytes long.
- If size is a concern, make sure the output buffer is at least large enough to hold a portion of the converted text. You can convert part of the text, then use the value of the `actualInputLength` parameter to identify the next byte to be taken and to determine how many bytes remain. To convert the remaining text, you simply call the function again with the remaining text and a new output buffer.

- If the destination encoding is a character encoding scheme—such as ISO-2022-JP, which begins in ASCII and switches to other coded character sets through limited combinations of escape sequences—then you need to allocate enough space to accommodate escape sequences that signal switches. ISO-2022-JP requires 3 to 5 bytes for an escape sequence preceding the 1-byte or 2-byte character it introduces. If you allocate a buffer that is less than 5 bytes, the `TECConvertText` function could fail, depending on the text being converted.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECFlushText

Flushes out any data in a converter object's temporary buffers and resets the converter object.

```
OSStatus TECFlushText (
   TECObjectRef encodingConverter,
   TextPtr outputBuffer,
   ByteCount outputBufferLength,
   ByteCount *actualOutputLength
);
```

Parameters

`encodingConverter`
A reference to the text converter object whose contents are to be flushed. This can be a reference returned by the `TECCreateConverter` (page 58) or `TECCreateConverterFromPath` (page 59) functions.

`outputBuffer`
On return, a pointer to a buffer that holds the converted text.

`outputBufferLength`
The length in bytes of the buffer provided by the `outputBuffer` parameter.

`actualOutputLength`
On return, a pointer to the number of bytes of converted text returned in the buffer specified by the `outputBuffer` parameter.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
You should always call `TECFlushText` when you finish converting a text stream. If you are converting a single stream in multiple chunks using multiple calls to `TECConvertText`, you only need to call `TECFlushText` after the last call to `TECConvertText` for that stream. The function uses the conversion path specified in the converter object you supply.

For the function to return successfully, the output buffer you allocate must be large enough to accommodate the flushed text. If the output buffer is too small to accommodate any flushed text, the function will fail. For best results, you should follow these guidelines when you allocate an output buffer:

- Base the buffer length on an estimate of the byte requirements of the destination encoding. Make sure you account for additional bytes needed by the destination encoding (for example, an escape sequence) in addition to the actual text.
- Always allocate a buffer at least 32 bytes long.

Encodings such as ISO-2022 that need to shift back to a certain default state at the end of a conversion can do so when this function is called.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECGetAvailableSniffers

Returns the list of sniffers available in all installed plug-ins.

```
OSStatus TECGetAvailableSniffers (
   TextEncoding availableSniffers[],
   ItemCount maxAvailableSniffers,
   ItemCount *actualAvailableSniffers
);
```

Parameters

availableSniffers
: On return, an array of text encoding specifications that the available sniffers currently support. You should use the `TECCountAvailableSniffers` (page 54) function to determine what size array to allocate.

maxAvailableSniffers
: The number of text encoding specifications the `availableSniffers` array can contain.

actualAvailableSniffers
: On return, a pointer to the number of text encodings in the `availableSniffers` array.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
This function ignores duplicate text encoding specifications. If you used the `TECCountAvailableSniffers` (page 54) function to determine the size of the `TECGetAvailableSniffers` array, the number of available encodings may be fewer than the number of array elements, because `TECCountAvailableSniffers` includes duplicate text encoding specifications in its count.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECGetAvailableTextEncodings

Returns the text encoding specifications currently configured in the Text Encoding Converter.

```
OSStatus TECGetAvailableTextEncodings (
   TextEncoding availableEncodings[],
   ItemCount maxAvailableEncodings,
   ItemCount *actualAvailableEncodings
);
```

Parameters

availableEncodings

On return, an array of text encoding specifications. You should use the `TECCountAvailableTextEncodings` (page 54) function to determine what size array to allocate.

maxAvailableEncodings

The number of text encoding specifications the `availableEncodings` array can contain.

actualAvailableEncodings

On return, a pointer to the number of text encodings returned in the `availableEncodings` array.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion

This function ignores duplicate text encoding specifications. If you used the `TECCountAvailableTextEncodings` (page 54) function to determine the size of the `availableEncodings []` array, the number of encodings may be fewer than the number of array elements, because `TECCountAvailableTextEncodings` includes duplicate text encodings in its count.

Availability

Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`TextEncodingConverter.h`

TECGetDestinationTextEncodings

Returns the encoding specifications for all the destination text encodings to which the Text Encoding Converter can directly convert the specified source encoding.

```
OSStatus TECGetDestinationTextEncodings (
   TextEncoding inputEncoding,
   TextEncoding destinationEncodings[],
   ItemCount maxDestinationEncodings,
   ItemCount *actualDestinationEncodings
);
```

Parameters

inputEncoding

The text encoding specification describing the source text.

destinationEncodings

On return, an array of specifications for the destination encodings to which the converter can directly convert the source encoding. You should use the `TECCountDestinationTextEncodings` (page 55) function to determine how large an array to allocate.

maxDestinationEncodings

The maximum number of destination text encodings that the array can contain.

actualDestinationEncodings

On return, a pointer to the number of text encoding specifications in the destination encodings array.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion

This function ignores duplicate direct text encoding specifications. If you used the `TECCountDestinationTextEncodings` (page 55) function to determine the size of the `destinationEncodings[]` array, the number of available encodings may be fewer than the number of array elements, because `TECCountDestinationTextEncodings` includes duplicates in its count.

You can display the names of these destination encodings to the user.

Availability

Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In

`TextEncodingConverter.h`

TECGetDirectTextEncodingConversions

Returns the types of direct conversions currently configured in the Text Encoding Converter.

```
OSStatus TECGetDirectTextEncodingConversions (
   TECConversionInfo availableConversions[],
   ItemCount maxAvailableConversions,
   ItemCount *actualAvailableConversions
);
```

Parameters

availableConversions

An array composed of text encoding conversion information structures, each of which specifies a set of source and destination encodings for a type of conversion. See `TECConversionInfo` (page 99) for more information. You should use the `TECGetDirectTextEncodingConversions` (page 67) function to determine how large to make the array.

maxAvailableConversions

The maximum number of text encoding conversion information structures that the `directConversions` array can contain.

actualAvailableConversions

On return, a pointer to the number of text encoding conversion information structures returned in the `directConversions` array.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion
This function ignores duplicate text encoding conversion information structures. If you used the `TECCountDirectTextEncodingConversions` (page 56) function to determine the size of the `directConversions[]` array, the number of text encoding conversion information structures may be fewer than the number of array elements, because `TECCountDirectTextEncodingConversions` counts duplicate text encoding conversion information structures.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECGetEncodingList

Gets the list of destination encodings from a converter object.

```
OSStatus TECGetEncodingList (
   TECObjectRef encodingConverter,
   ItemCount *numEncodings,
   Handle *encodingList
);
```

Parameters

`encodingConverter`
A reference to the text encoding conversion object returned by the `TECCreateOneToManyConverter` (page 60) function.

`numEncodings`
On return, a pointer to the number of encodings specified by the `encodingList` handle.

`encodingList`
A handle to an array of text encoding specifications. On return, it contains an array of text encoding specifications to which the converter object can convert. The memory for the array is allocated automatically by the Text Encoding Converter.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
The `TECDisposeConverter` function automatically disposes of the pointer for you. This means you should not reference the pointer after you have disposed of the converter object.

Plug-ins that perform one-to-many conversions use the `TECGetEncodingList` function to get the output encoding list from the converter object reference.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Declared In
`TextEncodingConverter.h`

TECGetInfo

Allocates a converter information structure of type `TECInfo` in the application heap using `NewHandle`, fills it out, and returns a handle.

```
OSStatus TECGetInfo (
   TECInfoHandle *tecInfo
);
```

Parameters

tecInfo

A handle to a structure of type `TECInfo` (page 101) containing information about the converter.

Return Value

A result code. See "TEC Manager Result Codes" (page 166). This function can return memory errors.

Discussion

When you are finished with the handle, your application must dispose of it using `DisposeHandle`. You must also perform any required preflighting or memory rearrangement before calling `TECGetInfo`.

Availability

Available in CarbonLib 1.0 and later when Text Common 1.2.1 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`TextCommon.h`

TECGetMailTextEncodings

Returns the currently supported mail encoding specifications for a region code.

```
OSStatus TECGetMailTextEncodings (
   RegionCode locale,
   TextEncoding availableEncodings[],
   ItemCount maxAvailableEncodings,
   ItemCount *actualAvailableEncodings
);
```

Parameters

locale

A Mac OS region code. A region code designates a combination of language, writing system, and geographic region; the region may not correspond to a particular country (for example, Swiss French or Arabic).

availableEncodings

An array of text encoding specifications. On return, the array contains specifications for the e-mail text encodings for a region code. You should use the function `TECCountMailTextEncodings` (page 56) function to determine what size array to allocate.

maxAvailableEncodings

The number of text encoding specifications the `availableEncodings` array can contain.

actualAvailableEncodings

On return, a pointer to the number of text encodings in the `availableEncodings` array.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion
This function ignores duplicate text encoding specifications. If you used the `TECCountMailTextEncodings` (page 56) function to determine the size of the `availableEncodings[]` array the number of available encodings may be fewer than the number of array elements, because `TECCountMailTextEncodings` includes duplicate text encoding specifications in its count.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECGetSubTextEncodings

Returns the text encoding specifications for the subencodings the encoding scheme supports.

```
OSStatus TECGetSubTextEncodings (
   TextEncoding inputEncoding,
   TextEncoding subEncodings[],
   ItemCount maxSubEncodings,
   ItemCount *actualSubEncodings
);
```

Parameters

`inputEncoding`
A text encoding specification.

`subEncodings`
On return, the array contains the specifications for the subencodings of the `inputEncoding` parameter. You should use the function `TECCountSubTextEncodings` (page 57) function to determine what size an array to allocate.

`maxSubEncodings`
The number of text encoding specifications the `subEncodings` array can contain.

`actualSubEncodings`
On return, a pointer to number of subencodings in the `subEncodings` array.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
Subencodings are text encodings that are embedded as part of a larger text encoding specification. For example, EUC-JP contains JIS Roman or ASCII, JIS X0208, JIS X0212, and half-width Katakana from JIS X0201. Not every encoding that can be broken into multiple encodings necessarily supports this routine. It's up to the plug-in developer to decide which encodings might be useful to break up. Subencodings are not the same as text encoding variants

If an encoding can be converted to multiple runs of encodings (as indicated by a destination base encoding of `kTextEncodingMultiRun`), you can call the `TECGetSubTextEncodings` (page 70) function to get the list of output encodings. See the `TECCreateOneToManyConverter` (page 60) and `TECGetDestinationTextEncodings` (page 66) functions for information about multiple output encoding run conversions.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECGetTextEncodingFromInternetName

Returns the Mac OS text encoding specification that corresponds to an Internet encoding name.

```
OSStatus TECGetTextEncodingFromInternetName (
   TextEncoding *textEncoding,
   ConstStr255Param encodingName
);
```

Parameters

textEncoding
On return, a pointer to a structure that contains a Mac OS text encoding specification.

encodingName
An Internet encoding name, in 7-bit US ASCII.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
Internet encoding names are stored as strings, while the Text Encoding Converter uses numeric values.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECGetTextEncodingInternetName

Returns the Internet encoding name that corresponds to a Mac OS text encoding.

```
OSStatus TECGetTextEncodingInternetName (
   TextEncoding textEncoding,
   Str255 encodingName
);
```

Parameters

textEncoding
A Mac OS text encoding specification.

encodingName
On return, the Internet encoding name, in 7-bit US ASCII. If there are several Internet encoding names for the same text encoding, the `encodingName` parameter contains the preferred name.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECGetWebTextEncodings

Returns the currently supported text encoding specifications for a region code.

```
OSStatus TECGetWebTextEncodings (
   RegionCode locale,
   TextEncoding availableEncodings[],
   ItemCount maxAvailableEncodings,
   ItemCount *actualAvailableEncodings
);
```

Parameters

locale

A Mac OS region code. A region code designates a combination of language, writing system, and geographic region and may not correspond to a particular country (for example, Swiss French or Arabic).

availableEncodings

On return, an array that contains specifications for the currently supported text encodings in the specified region. You should use the `TECCountWebTextEncodings` (page 58) function to determine how large an array to allocate.

maxAvailableEncodings

The number of text encodings specifications the `availableEncodings` array can contain.

actualAvailableEncodings

On return, a pointer to the number of text encodings specifications in the `availableEncodings` array.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
This function ignores duplicate text encoding specifications. If you used the `TECCountWebTextEncodings` (page 58) function to determine the size of the `availableEncodings[]` array the number of available encodings may be fewer than the number of array elements, because `TECCountWebTextEncodings` includes duplicate text encoding specifications in its count.

You can use the list of available encodings to create an encoding selection menu for a Web browser.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextEncodingConverter.h`

TECSniffTextEncoding

Analyzes a text stream and returns the probable encodings in a ranked list, based on an array of possible encodings you supply. It also returns the number of errors and features for each encoding.

```
OSStatus TECSniffTextEncoding (
   TECSnifferObjectRef encodingSniffer,
   ConstTextPtr inputBuffer,
   ByteCount inputBufferLength,
   TextEncoding testEncodings[],
   ItemCount numTextEncodings,
   ItemCount numErrsArray[],
   ItemCount maxErrs,
   ItemCount numFeaturesArray[],
   ItemCount maxFeatures
);
```

Parameters

encodingSniffer

A reference to a sniffer object.

inputBuffer

The text to be sniffed.

inputBufferLength

The length of the input buffer.

testEncodings

An array of text encoding specifications. You must fill the array with the text encodings for which you want to sniff. On output, the array elements are reordered from the most likely to the least likely text encodings.

numTextEncodings

The number of entries in the `testEncodings[]` parameter.

numErrsArray

An array that must contain at least `numTextEncodings` elements. On return, an array of the number of errors found for each possible text encoding. The array elements are in the same order as the `testEncodings[]` array elements at output.

maxErrs

The maximum number of errors a sniffer can encounter. The sniffer stops looking for an encoding after this number is reached.

numFeaturesArray

An array of that must contain at least `numTextEncodings` elements. On return, an array of the number of features found for each possible text encoding. The array elements are in the same order as the `testEncodings[]` array elements at output.

maxFeatures

The maximum number of features a sniffer can encounter. The sniffer stops looking for a features after this number is reached.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Discussion

An error indicates a code point or sequence that is illegal in the specified encoding. A feature indicates the presence of a sequence that is characteristic of that encoding.

For example, the byte sequence which is interpreted in Mac OS Roman as "é" could legally be interpreted either as Mac OS Roman text or as Mac OS Japanese text. Both sniffers would return zero errors, but the Mac OS Japanese sniffer would also return two features of Mac OS Japanese (representing two legal 2-byte characters.)

The arrays are returned in a ranked list with the most likely text encodings first. The results are sorted first by number of errors (fewest to most), then by number of features (most to fewest), and then by the original order in the list. On return, the most likely encoding is in `testEncodings[0]` or `testEncodings[1]`.

If an encoding is not examined, its number of errors and features are set to 0xFFFFFFFF, and the encoding is sorted to the end of the list.

Availability
Available in CarbonLib 1.0 and later when Text Encoding Convertor 1.2 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Declared In
`TextEncodingConverter.h`

TruncateForTextToUnicode

Identifies where your application can safely break a multibyte string to be converted to Unicode so that the string is not broken in the middle of a multibyte character.

```
OSStatus TruncateForTextToUnicode (
   ConstTextToUnicodeInfo iTextToUnicodeInfo,
   ByteCount iSourceLen,
   ConstLogicalAddress iSourceStr,
   ByteCount iMaxLen,
   ByteCount *oTruncatedLen
);
```

Parameters

iTextToUnicodeInfo
The Unicode converter object of type `TextToUnicodeInfo` (page 106) for the text string to be divided up with each segment properly truncated. The `TruncateForTextToUnicode` function does not modify the object's contents.

iSourceLen
The length in bytes of the multibyte string to be divided up.

iSourceStr
The address of the multibyte string to be divided up.

iMaxLen
The maximum allowable length of the string to be truncated. This must be less than or equal to `iSourceLen`.

oTruncatedLen
A pointer to a value of type `ByteCount`. On return, this value contains the length of the longest portion of the multibyte string, pointed to by `iSourceStr`, that is less than or equal to the length specified by `iMaxLen`. This identifies the byte after which you can break the string.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
Your application can use this function to break a string properly before you call the function `ConvertFromTextToUnicode` (page 17) so that the string you pass it is terminated with complete characters. You can call this function repeatedly to properly divide up a text segment, each time identifying the new beginning of the string, until the last portion of the text is less than or equal to the maximum allowable length. Each time you use the function, you get a properly terminated string within the allowable length range.

Because the `TruncateForTextToUnicode` function does not modify the contents of the Unicode converter object, you can call this function safely between calls to the function `ConvertFromTextToUnicode` (page 17).

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

TruncateForUnicodeToText

Identifies where your application can safely break a Unicode string to be converted to any encoding so that the string is broken in a way that preserves the text element integrity.

```
OSStatus TruncateForUnicodeToText (
   ConstUnicodeToTextInfo iUnicodeToTextInfo,
   ByteCount iSourceLen,
   const UniChar iSourceStr[],
   OptionBits iControlFlags,
   ByteCount iMaxLen,
   ByteCount *oTruncatedLen
);
```

Parameters

iUnicodeToTextInfo
A Unicode converter object `UnicodeToTextInfo` (page 109) for the Unicode string to be divided up. The `TruncateForUnicodeToText` function does not modify the contents of this private structure.

iSourceLen
The length in bytes of the Unicode string to be divided up.

iSourceStr
A pointer to the Unicode string to be divided up.

iControlFlags
Truncation control flags. Specify the flag `kUnicodeStringUnterminatedMask` if truncating a buffer of text that belongs to a longer stream containing a subsequent buffer of text that could have characters belonging to a text element that begins at the end of the current buffer. If you set this flag, typically you would set the `iMaxLen` parameter equal to `iSourceLen`.

iMaxLen
The maximum allowable length of the string to be truncated. This must be less than or equal to `iSourceLen`.

oTruncatedLen

A pointer to a value of type `ByteCount`. On return, this value contains the length of the longest portion of the Unicode source string, pointed to by the `iSourceStr` parameter, that is less than or equal to the value of the `iMaxLen` parameter. This returned parameter identifies the byte after which you can truncate the string.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Discussion
Your application can use this function to divide up a Unicode string properly truncating each portion before you call `ConvertFromUnicodeToText` or `ConvertFromUnicodeToScriptCodeRun` to convert the string. You can call this function repeatedly to properly truncate a text segment, each time identifying the new beginning of the string, until the last portion of the text is less than or equal to the maximum allowable length. Each time you use the function, you get a properly terminated string within the allowable length range.

Because this function does not modify the contents of the Unicode converter object, you can call this function between conversion calls.

Availability
Available in CarbonLib 1.0 and later when Unicode Utilities 1.1 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`UnicodeConverter.h`

UCGetCharProperty

Obtains the value associated with a property type for the specified `UniChar` characters.

```
OSStatus UCGetCharProperty (
   const UniChar *charPtr,
   UniCharCount textLength,
   UCCharPropertyType propType,
   UCCharPropertyValue *propValue
);
```

Parameters

charPtr

A pointer to the Unicode text whose property value you want to obtain.

textLength

The length of the text pointed to by `charPtr`.

propType

The property type for the `UniChar` character whose value you want to obtain. See "Unicode Character Property Types" (page 160) for a list of the constants you can supply.

propValue

On return, the value associated with the property type specified by the `propType` parameter. See "Unicode Character Property Values" (page 160) for a list of the constants that can be returned.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

Availability
Available in CarbonLib 1.0 and later when Text Common 1.5 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextCommon.h`

UpgradeScriptInfoToTextEncoding

Converts any combination of a Mac OS script code, a language code, a region code, and a font name to a text encoding.

```
OSStatus UpgradeScriptInfoToTextEncoding (
   ScriptCode iTextScriptID,
   LangCode iTextLanguageID,
   RegionCode iRegionID,
   ConstStr255Param iTextFontname,
   TextEncoding *oEncoding
);
```

Parameters

iTextScriptID

A valid Script Manager script code. The Mac OS Script Manager defines constants for script codes using this format: `sm`Xxx. To designate the system script, specify the meta-value of `smSystemScript`. To designate the current script based on the font specified in the graphics port (`grafPort`), specify the metavalue of `smCurrentScript`. To indicate that you do not want to provide a script code for this parameter, specify the constant `kTextScriptDontCare`.

iTextLanguageID

A valid Script Manager language code. The Mac OS Script Manager defines constants for language codes using this format: `lang`Xxx. To indicate that you do not want to provide a language code for this parameter, specify the constant `kTextLanguageDontCare`.

iRegionID

A valid Script Manager region code. The Mac OS Script Manager defines constants for region codes using this format: `ver`Xxx. To indicate that you do not want to provide a region code for this parameter, specify the constant `kTextRegionDontCare`.

iTextFontname

The name of a font associated with a particular text encoding specification, such as Symbol or Zapf Dingbats, or the name of any font that is currently installed on the system. To indicate that you do not want to provide a font name, specify a value of `NULL`.

oEncoding

A pointer to a value of type `TextEncoding`. On return, this value holds the text encoding specification that the function created from the other values you provided.

Return Value
A result code. See "TEC Manager Result Codes" (page 166). This function returns `paramErr` if two or more of the input parameter values conflict in some way—for example, the Mac OS language code does not belong to the script whose script code you specified, or if the input parameter values are invalid. The function returns a `kTECTableFormatErr` result code if the internal mapping tables used for translation are invalid.

Discussion
The `UpgradeScriptInfoToTextEncoding` function allows you to derive a text encoding specification from script codes, language codes, region codes, and font names. A one-to-one correspondence exists between many of the Script Manager's script codes and a particular Mac OS text encoding base value. However, because text encodings are a superset of script codes, some combinations of script code, language code, region code, and font name might result in a different text encoding base value than would be the case if the translation were based on the script code alone.

When you call the `UpgradeScriptInfoToTextEncoding` function, you can specify any combination of its parameters, but you must specify at least one.

If you don't specify an explicit value for a script, language, or region code parameter, you must pass the do-not-care constant appropriate to that parameter. If you do not specify an explicit value for `iTextFontName`, you must pass `NULL`. `UpgradeScriptInfoToTextEncoding` uses as much information as you supply to determine the equivalent text encoding or the closest approximation. If you provide more than one parameter, all parameters are checked against one another to ensure that they are valid in combination.

A font name, such as `'Symbol'` or `'Zapf Dingbats,'` can indicate a particular text encoding base. Other font names can indicate particular variants associated with a particular text encoding base. Otherwise, the font name is used to obtain a script code, and this script code will be checked against any script code you supply (in this case, the font must be installed; if it is not, the function returns a `paramErr` result code). If you do not supply either a language code or a region code and the script code you supply or the one that is derived matches the system script, then the system's localization is used to determine the appropriate region and language code. This is used for deriving text encoding base values that depend on region and language, such as `kTextEncodingMacTurkish`.

For more information see the `RevertTextEncodingToScriptInfo` (page 47) function and "Base Text Encodings" (page 122).

Availability
Available in CarbonLib 1.0 and later when Text Common 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`TextCommon.h`

Callbacks by Task

Setting Up a Fallback Handler

`UnicodeToTextFallbackProcPtr` (page 93)
Defines a pointer to a function that converts a Unicode text element for which there is no destination encoding equivalent in the appropriate mapping table to the fallback character sequence defined by your fallback handler, and returns the converted character sequence to the Unicode Converter.

Setting Up a TEC Plug-in

`TECPluginGetPluginDispatchTablePtr` (page 89)
Defines a pointer to a function that returns a pointer to a plug-in dispatch table.

`TECPluginNewEncodingConverterPtr` (page 91)
Defines a pointer to a function that determines a conversion path for a source and destination encoding, then creates a text encoding converter object and returns a pointer to it.

`TECPluginClearContextInfoPtr` (page 80)
Defines a pointer to a function that resets a converter object to its initial state.

`TECPluginConvertTextEncodingPtr` (page 81)
Defines a pointer to a function that converts stream of text from a source encoding to a destination encoding, using the conversion path specified by the converter object you supply.

`TECPluginFlushConversionPtr` (page 83)
Defines a pointer to a function that flushes out any data in a converter object's temporary buffers and resets the converter object.

`TECPluginDisposeEncodingConverterPtr` (page 81)
Defines a pointer to a function that disposes of a converter object.

`TECPluginNewEncodingSnifferPtr` (page 92)
Defines a pointer to a function that creates a sniffer object and returns a reference to it.

`TECPluginClearSnifferContextInfoPtr` (page 80)
Defines a pointer to a function that resets a sniffer object to its initial settings.

`TECPluginSniffTextEncodingPtr` (page 92)
Defines a pointer to a function that analyzes a text stream and returns the probable encodings in a ranked list, based on an array of possible encodings you supply; it also returns the number of errors and features for each encoding.

`TECPluginDisposeEncodingSnifferPtr` (page 82)
Defines a pointer to a function that disposes of a sniffer object.

`TECPluginGetCountAvailableTextEncodingsPtr` (page 85)
Defines a pointer to a function that obtains the available text encodings.

`TECPluginGetCountAvailableTextEncodingPairsPtr` (page 84)
Defines a pointer to a function that obtains the available text encoding pairs.

`TECPluginGetCountDestinationTextEncodingsPtr` (page 86)
Defines a pointer to a function that counts and returns the number of destination encodings to which a specified source encoding can be converted in one step.

`TECPluginGetCountSubTextEncodingsPtr` (page 87)
Defines a pointer to a function that obtains the text encoding specifications for the subencodings the encoding scheme supports.

`TECPluginGetCountAvailableSniffersPtr` (page 83)
Defines a pointer to a function that counts and returns the number of sniffers available in all installed plug-ins.

`TECPluginGetCountWebEncodingsPtr` (page 88)
Defines a pointer to a function that obtains the available web text encodings.

`TECPluginGetCountMailEncodingsPtr` (page 87)
Defines a pointer to a function that obtains the text encodings available for email.

`TECPluginGetTextEncodingInternetNamePtr` (page 90)
Defines a pointer to a function that obtains the Internet text encoding name for a text encoding specification.

`TECPluginGetTextEncodingFromInternetNamePtr` (page 89)
Defines a pointer to a function that obtains the text encoding for an Internet text encoding name.

Callbacks

TECPluginClearContextInfoPtr

Defines a pointer to a function that resets a converter object to its initial state.

```
typedef OSStatus (*TECPluginClearContextInfoPtr)
(
    TECObjectRef encodingConverter,
    TECConverterContextRec * plugContext
);
```

If you name your function `MyTECPluginClearContextInfo`, you would declare it like this:

```
OSStatus MyTECPluginClearContextInfoPtr
(
    TECObjectRef encodingConverter,
    TECConverterContextRec * plugContext
);
```

Parameters

encodingConverter

A reference to the text encoding converter object that needs to be reset.

plugContext

A pointer to a TEC converter context record.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginClearSnifferContextInfoPtr

Defines a pointer to a function that resets a sniffer object to its initial settings.

```
typedef OSStatus (*TECPluginClearSnifferContextInfoPtr)
(
    TECSnifferObjectRef encodingSniffer,
    TECSnifferContextRec * snifContext
);
```

If you name your function `MyTECPluginClearSnifferContextInfo`, you would declare it like this:

```
OSStatus MyTECPluginClearSnifferContextInfoPtr
(
    TECSnifferObjectRef encodingSniffer,
    TECSnifferContextRec * snifContext
);
```

Parameters

encodingSniffer

A reference to the sniffer object that needs to be reset.

snifContext

A pointer to a TEC sniffer context record.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginConvertTextEncodingPtr

Defines a pointer to a function that converts stream of text from a source encoding to a destination encoding, using the conversion path specified by the converter object you supply.

```
typedef OSStatus (*TECPluginConvertTextEncodingPtr)
(
    TECObjectRef encodingConverter,
    TECConverterContextRec * plugContext
);
```

If you name your function `MyTECPluginConvertTextEncoding`, you would declare it like this:

```
OSStatus MyTECPluginConvertTextEncodingPtr
(
    TECObjectRef encodingConverter,
    TECConverterContextRec * plugContext
);
```

Parameters

encodingConverter

A reference to the text encoding converter object to use for the conversion.

plugContext

A pointer to a TEC converter context record that contains the text and other information needed for the conversion.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginDisposeEncodingConverterPtr

Defines a pointer to a function that disposes of a converter object.

```
typedef OSStatus (*TECPluginDisposeEncodingConverterPtr)
(
    TECObjectRef newEncodingConverter,
    TECConverterContextRec * plugContext
);
```

If you name your function `MyTECPluginDisposeEncodingConverter`, you would declare it like this:

```
OSStatus MyTECPluginDisposeEncodingConverterPtr
(
    TECObjectRef newEncodingConverter,
    TECConverterContextRec * plugContext
);
```

Parameters

newEncodingConverter

A reference to the converter object to dispose of.

plugContext

A pointer to a TEC converter context record.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginDisposeEncodingSnifferPtr

Defines a pointer to a function that disposes of a sniffer object.

```
typedef OSStatus (*TECPluginDisposeEncodingSnifferPtr)
(
    TECSnifferObjectRef encodingSniffer,
    TECSnifferContextRec * snifContext
);
```

If you name your function `MyTECPluginDisposeEncodingSniffer`, you would declare it like this:

```
OSStatus MyTECPluginDisposeEncodingSnifferPtr
(
    TECSnifferObjectRef encodingSniffer,
    TECSnifferContextRec * snifContext
);
```

Parameters

encodingSniffer

A reference to the sniffer object you want to dispose.

snifContext

A pointer to a TEC sniffer context record.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

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

Declared In
`TextEncodingPlugin.h`

TECPluginFlushConversionPtr

Defines a pointer to a function that flushes out any data in a converter object's temporary buffers and resets the converter object.

```
typedef OSStatus (*TECPluginFlushConversionPtr)
(
    TECObjectRef encodingConverter,
    TECConverterContextRec * plugContext
);
```

If you name your function `MyTECPluginFlushConversion`, you would declare it like this:

```
OSStatus MyTECPluginFlushConversionPtr
(
    TECObjectRef encodingConverter,
    TECConverterContextRec * plugContext
);
```

Parameters

`encodingConverter`
A reference to the text converter object whose contents are to be flushed.

`plugContext`
A pointer to a TEC converter context record.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

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

Declared In
`TextEncodingPlugin.h`

TECPluginGetCountAvailableSniffersPtr

Defines a pointer to a function that counts and returns the number of sniffers available in all installed plug-ins.

```
typedef OSStatus (*TECPluginGetCountAvailableSniffersPtr)
(
    TextEncoding * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

If you name your function `MyTECPluginGetCountAvailableSniffers`, you would declare it like this:

```
OSStatus MyTECPluginGetCountAvailableSniffersPtr
(
    TextEncoding * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

Parameters

availableEncodings

On return, a pointer to the currently available sniffer text encoding specifications.

maxAvailableEncodings

The number of text encoding specifications the `availableEncodings` array can contain.

actualAvailableEncodings

On the return, the number of text encoding specifications the `availableEncodings` array actually contains.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginGetCountAvailableTextEncodingPairsPtr

Defines a pointer to a function that obtains the available text encoding pairs.

```
typedef OSStatus (*TECPluginGetCountAvailableTextEncodingPairsPtr)
(
    TECConversionInfo * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

If you name your function `MyTECPluginGetCountAvailableTextEncodingPairs`, you would declare it like this:

```
OSStatus MyTECPluginGetCountAvailableTextEncodingPairsPtr
(
    TECConversionInfo * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
```

```
);
```

Parameters

availableEncodings

On return, an array of text encoding conversion information structures, each of which specifies a set of source and destination encodings for a type of conversion.

maxAvailableEncodings

The number of text encoding information structures the `availableEncodings` array can contain.

actualAvailableEncodings

On the return, the number of text encoding information structures the `availableEncodings` array actually contains.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginGetCountAvailableTextEncodingsPtr

Defines a pointer to a function that obtains the available text encodings.

```
typedef OSStatus (*TECPluginGetCountAvailableTextEncodingsPtr)
(
    TextEncoding * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

If you name your function `MyTECPluginGetCountAvailableTextEncodings`, you would declare it like this:

```
OSStatus MyTECPluginGetCountAvailableTextEncodingsPtr
(
    TextEncoding * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

Parameters

availableEncodings

On return, a pointer to the currently available text encoding specifications.

maxAvailableEncodings

The number of text encoding specifications the `availableEncodings` array can contain.

actualAvailableEncodings

On the return, the number of text encoding specifications the `availableEncodings` array actually contains.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

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

Declared In
`TextEncodingPlugin.h`

TECPluginGetCountDestinationTextEncodingsPtr

Defines a pointer to a function that counts and returns the number of destination encodings to which a specified source encoding can be converted in one step.

```
typedef OSStatus (*TECPluginGetCountDestinationTextEncodingsPtr)
(
    TextEncoding inputEncoding,
    TextEncoding * destinationEncodings,
    ItemCount maxDestinationEncodings,
    ItemCount * actualDestinationEncodings
);
```

If you name your function `MyTECPluginGetCountDestinationTextEncodings`, you would declare it like this:

```
OSStatus MyTECPluginGetCountDestinationTextEncodingsPtr
(
    TextEncoding inputEncoding,
    TextEncoding * destinationEncodings,
    ItemCount maxDestinationEncodings,
    ItemCount * actualDestinationEncodings
);
```

Parameters

`inputEncoding`
The text encoding specification describing the source text.

`destinationEncodings`
On return, a pointer to text encodings to which the source encoding can be converted in one step.

`maxDestinationEncodings`
The maximum number of text encodings that can be specified by the `destinationEncodings` parameter.

`actualDestinationEncodings`
On return, the actual number of text encodings specified by the `destinationEncodings` parameter.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

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

Declared In
`TextEncodingPlugin.h`

TECPluginGetCountMailEncodingsPtr

Defines a pointer to a function that obtains the text encodings available for email.

```
typedef OSStatus (*TECPluginGetCountMailEncodingsPtr)
(
    TextEncoding * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

If you name your function `MyTECPluginGetCountMailEncodings`, you would declare it like this:

```
OSStatus MyTECPluginGetCountMailEncodingsPtr
(
    TextEncoding * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

Parameters

availableEncodings

On return, a pointer to the text encodings available for email.

maxAvailableEncodings

The maximum number of text encodings that can be specified by the `availableEncodings` parameter.

actualAvailableEncodings

On return, the number of text encoding specifications `availableEncodings` actually contains.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginGetCountSubTextEncodingsPtr

Defines a pointer to a function that obtains the text encoding specifications for the subencodings the encoding scheme supports.

```
typedef OSStatus (*TECPluginGetCountSubTextEncodingsPtr)
(
    TextEncoding inputEncoding,
    TextEncoding subEncodings[],
    ItemCount maxSubEncodings,
    ItemCount * actualSubEncodings
);
```

If you name your function `MyTECPluginGetCountSubTextEncodings`, you would declare it like this:

```
OSStatus MyTECPluginGetCountSubTextEncodingsPtr
(
```

```
    TextEncoding inputEncoding,
    TextEncoding subEncodings[],
    ItemCount maxSubEncodings,
    ItemCount * actualSubEncodings
);
```

Parameters

inputEncoding

A text encoding specification.

subEncodings

On return, the array contains the specifications for the subencodings of the `inputEncoding` parameter.

maxSubEncodings

The number of text encoding specifications the `subEncodings` array can contain.

actualSubEncodings

On return, a pointer to number of subencodings in the `subEncodings` array.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginGetCountWebEncodingsPtr

Defines a pointer to a function that obtains the available web text encodings.

```
typedef OSStatus (*TECPluginGetCountWebEncodingsPtr)
(
    TextEncoding * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

If you name your function `MyTECPluginGetCountWebEncodings`, you would declare it like this:

```
OSStatus MyTECPluginGetCountWebEncodingsPtr
(
    TextEncoding * availableEncodings,
    ItemCount maxAvailableEncodings,
    ItemCount * actualAvailableEncodings
);
```

Parameters

availableEncodings

On return, points to the currently supported text encodings available for the web.

maxAvailableEncodings

The number of text encodings specifications that `availableEncodings` can specify.

actualAvailableEncodings
On return, the number of text encodings specifications `availableEncodings` actually contains.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

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

Declared In
`TextEncodingPlugin.h`

TECPluginGetPluginDispatchTablePtr

Defines a pointer to a function that returns a pointer to a plug-in dispatch table.

```
typedef TECPluginDispatchTable * (*TECPluginGetPluginDispatchTablePtr)
(
);
```

If you name your function `ConverterPluginGetPluginDispatchTable`, you would declare it like this:

```
TECPluginDispatchTable * ConverterPluginGetPluginDispatchTable();
```

Parameters

Return Value
A pointer to the function dispatch table for the plug-in.

Discussion
You need this callback only for Mac OS X plug-ins. When you create a TEC plug-in in Mac OS X you must export a function named `ConverterPluginGetPluginDispatchTable` with the following prototype:

```
extern TECPluginDispatchTable *ConverterPluginGetPluginDispatchTable (void)
```

This function must return a pointer to the function dispatch table for the plug-in. It is important you name the function `ConverterPluginGetPluginDispatchTable` because `TECPluginGetPluginDispatchTablePtr` is a function pointer to a function of this exact name.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`TextEncodingPlugin.h`

TECPluginGetTextEncodingFromInternetNamePtr

Defines a pointer to a function that obtains the text encoding for an Internet text encoding name.

```
typedef OSStatus (*TECPluginGetTextEncodingFromInternetNamePtr)
(
    TextEncoding * textEncoding,
    ConstStr255Param encodingName
);
```

If you name your function `MyTECPluginGetTextEncodingFromInternetName`, you would declare it like this:

```
OSStatus MyTECPluginGetTextEncodingFromInternetNamePtr
(
    TextEncoding * textEncoding,
    ConstStr255Param encodingName
);
```

Parameters

`textEncoding`

On return, a pointer to a structure that contains a text encoding specification for the text encoding name specified by the `encodingName` parameter.

`encodingName`

An Internet encoding name, in 7-bit US ASCII.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginGetTextEncodingInternetNamePtr

Defines a pointer to a function that obtains the Internet text encoding name for a text encoding specification.

```
typedef OSStatus (*TECPluginGetTextEncodingInternetNamePtr)
(
    TextEncoding textEncoding,
    Str255 encodingName
);
```

If you name your function `MyTECPluginGetTextEncodingInternetName`, you would declare it like this:

```
OSStatus MyTECPluginGetTextEncodingInternetNamePtr
(
    TextEncoding textEncoding,
    Str255 encodingName
);
```

Parameters

`textEncoding`

A text encoding specification.

encodingName
On return, the Internet encoding name, in 7-bit US ASCII. If there are several Internet encoding names for the same text encoding, the `encodingName` parameter contains the preferred name.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

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

Declared In
`TextEncodingPlugin.h`

TECPluginNewEncodingConverterPtr

Defines a pointer to a function that determines a conversion path for a source and destination encoding, then creates a text encoding converter object and returns a pointer to it.

```
typedef OSStatus (*TECPluginNewEncodingConverterPtr)
(
    TECObjectRef * newEncodingConverter,
    TECConverterContextRec * plugContext,
    TextEncoding inputEncoding,
    TextEncoding outputEncoding
);
```

If you name your function `MyTECPluginNewEncodingConverter`, you would declare it like this:

```
OSStatus MyTECPluginNewEncodingConverterPtr
(
    TECObjectRef * newEncodingConverter,
    TECConverterContextRec * plugContext,
    TextEncoding inputEncoding,
    TextEncoding outputEncoding
);
```

Parameters

newEncodingConverter
A pointer to a converter object. On return, this points to a newly created text converter object.

plugContext
A pointer to a TEC converter context record.

inputEncoding
The text encoding specification for the source text.

outputEncoding
The text encoding specification for the destination text.

Return Value
A result code. See "TEC Manager Result Codes" (page 166).

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

Declared In
`TextEncodingPlugin.h`

TECPluginNewEncodingSnifferPtr

Defines a pointer to a function that creates a sniffer object and returns a reference to it.

```
typedef OSStatus (*TECPluginNewEncodingSnifferPtr)
(
    TECSnifferObjectRef * encodingSniffer,
    TECSnifferContextRec * snifContext,
    TextEncoding inputEncoding
);
```

If you name your function `MyTECPluginNewEncodingSniffer`, you would declare it like this:

```
OSStatus MyTECPluginNewEncodingSnifferPtr
(
    TECSnifferObjectRef * encodingSniffer,
    TECSnifferContextRec * snifContext,
    TextEncoding inputEncoding
);
```

Parameters

encodingSniffer

A pointer to a sniffer object reference, which is of type `TECSnifferObjectRef` (page 104). On return, the reference pertains to the newly created sniffer object.

snifContext

A pointer to a TEC sniffer context record.

inputEncoding

The text encoding specification to be detected by the sniffer object.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECPluginSniffTextEncodingPtr

Defines a pointer to a function that analyzes a text stream and returns the probable encodings in a ranked list, based on an array of possible encodings you supply; it also returns the number of errors and features for each encoding.

```
typedef OSStatus (*TECPluginSniffTextEncodingPtr)
(
    TECSnifferObjectRef encodingSniffer,
    TECSnifferContextRec * snifContext
);
```

If you name your function `MyTECPluginSniffTextEncoding`, you would declare it like this:

```
OSStatus MyTECPluginSniffTextEncodingPtr
(
    TECSnifferObjectRef encodingSniffer,
```

```
    TECSnifferContextRec * snifContext
);
```

Parameters

encodingSniffer

A reference to a sniffer object.

snifContext

A pointer to a TEC sniffer context record.

Return Value

A result code. See "TEC Manager Result Codes" (page 166).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

UnicodeToTextFallbackProcPtr

Defines a pointer to a function that converts a Unicode text element for which there is no destination encoding equivalent in the appropriate mapping table to the fallback character sequence defined by your fallback handler, and returns the converted character sequence to the Unicode Converter.

```
typedef OSStatus (*UnicodeToTextFallbackProcPtr)
(
    UniChar * iSrcUniStr,
    ByteCount iSrcUniStrLen,
    ByteCount * oSrcConvLen,
    TextPtr oDestStr,
    ByteCount iDestStrLen,
    ByteCount * oDestConvLen,
    LogicalAddress iInfoPtr,
    ConstUnicodeMappingPtr iUnicodeMappingPtr
);
```

If you name your function `MyUnicodeToTextFallbackProc`, you would declare it like this:

```
OSStatus MyUnicodeToTextFallbackProcPtr
(
    UniChar * iSrcUniStr,
    ByteCount iSrcUniStrLen,
    ByteCount * oSrcConvLen,
    TextPtr oDestStr,
    ByteCount iDestStrLen,
    ByteCount * oDestConvLen,
    LogicalAddress iInfoPtr,
    ConstUnicodeMappingPtr iUnicodeMappingPtr
);
```

Parameters

iSrcUniStr

A pointer to a single UTF-16 character to be mapped by the fallback handler.

iSrcUniStrLen

The length in bytes of the UTF-16 character indicated by the `iSrcUniStr` parameter. Usually this is 2 bytes, but it could be 4 bytes for a non-BMP character.

oSrcConvLen

On return, a pointer to the length in bytes of the portion of the Unicode character that was actually processed by your fallback handler. Your fallback handler returns this value. It should set this to 0 if none of the text was handled, or 2 or 4 if the Unicode character was handled. This value is initialized to 0 before the fallback handler is called.

oDestStr

A pointer to the output buffer where your handler should place any converted text.

iDestStrLen

The maximum size in bytes of the buffer provided by the `oDestStr` parameter.

oDestConvLen

On return, a pointer to the length in bytes of the fallback character sequence generated by your fallback handler. Your handler should return this length. It is initialized to 0 (zero) before the fallback handler is called.

iInfoPtr

A pointer to a block of memory allocated by your application, which can be used by your fallback handler in any way that you like. This is the same pointer passed as the last parameter of `SetFallbackUnicodeToText` or `SetFallbackUnicodeToTextRun`. How you use the data passed to you in this memory block is particular to your handler. This is similar in use to a reference constant (refcon).

iUnicodeMappingPtr

A constant pointer to a structure of type `UnicodeMapping` (page 107). This structure identifies a Unicode encoding specification and a particular base encoding specification.

Return Value

A result code. See "TEC Manager Result Codes" (page 166). Your handler should return `noErr` if it can handle the fallback, or `kTECUnmappableElementErr` if it cannot. It can return other errors for exceptional conditions, such as when the output buffer is too small. If your handler returns `kTECUnmappableElementErr`, then `oSrcConvLen` and `oDestConvLen` are ignored because either the default handler will be called or the default fallback sequence will be used.

Discussion

The Unicode Converter calls your fallback handler when it cannot convert a text string using the mapping table specified by the Unicode converter object passed to either `ConvertFromUnicodeToText` or `ConvertFromUnicodeToPString`. The control flags you set for the `controlFlags` parameter of the function `SetFallbackUnicodeToText` (page 48) or the `SetFallbackUnicodeToTextRun` (page 49) stipulate which fallback handler the Unicode Converter should call and which one to try first if both can be used.

When the Unicode Converter calls your handler, it passes to it the Unicode character to be converted and its length, a buffer for the converted string you return and the buffer length, and a pointer to a block of memory containing the data your application supplied to be passed on to your fallback handler.

After you convert the Unicode text segment to fallback characters, you return the fallback character sequence of the converted text in the buffer provided to you and the length in bytes of this fallback character sequence. You also return the length in bytes of the portion of the source Unicode text element that your handler actually processed.

You provide a fallback-handler function for use with the function `CreateUnicodeToTextInfoByEncoding` (page 33), `ConvertFromUnicodeToPString` (page 19), `ConvertFromUnicodeToTextRun` (page 25), or `ConvertFromUnicodeToScriptCodeRun` (page 20). You associate an application-defined fallback handler with a particular Unicode converter object you intend to pass to the conversion function when you call it.

Text converted from UTF-8 will already have been converted to UTF-16 before the fallback handler is called to process it. Your fallback handler should do all of its processing on text encoded in UTF-16.

Your application-defined fallback handler should not move memory or call any toolbox function that would move memory. If it needs memory, the memory should be allocated before the call to `SetFallbackUnicodeToText` or `SetFallbackUnicodeToTextRun`, and a memory reference should be passed either directly as `iInfoPtr` or in the data referenced by `iInfoPtr`.

To associate a fallback-handler function with a Unicode converter object you use the `SetFallbackUnicodeToText` (page 48) and `SetFallbackUnicodeToTextRun` (page 49) functions. For these functions, you must pass a universal procedure pointer (`UniversalProcPtr`). This is derived from a pointer to your function by using the predefined macro `NewUnicodeToTextFallbackProc`.

For versions of the Unicode Converter prior to 1.2, the fallback handler may receive a multiple character text element, so the source string length value could be greater than 2 and the fallback handler may set `srcConvLen` to a value greater than 2. In versions earlier than 1.2.1, the `srcConvLen` and `destConvLen` variables are not initialized to 0; both values are ignored unless the fallback handler returns `noErr`.

The following example shows how to install an application-defined fallback handler. You can name your application-defined fallback handler anything you choose. The name, `MyUnicodeToTextFallbackProc`, used in this example is not significant. However, you must adhere to the parameters, the return type, and the calling convention as expressed in this example, which follows the prototype, because a pointer to this function must be of type `UnicodeToTextFallbackProcPtr` as defined in the `UnicodeConverter.h` header file.

The `UnicodeConverter.h` header file also defines the `UnicodeToTextFallbackUPP` type and the `NewUnicodeToTextFallbackProc` macro.

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

Declared In
`UnicodeConverter.h`

Data Types

ConstScriptCodeRunPtr

Defines a constant script code run pointer.

```
typedef const ScriptCodeRun * ConstScriptCodeRunPtr;
```

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

Declared In
TextCommon.h

ConstTextEncodingRunPtr

Defines a constant text encoding run pointer.

```
typedef const TextEncodingRun * ConstTextEncodingRunPtr;
```

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

Declared In
TextCommon.h

ConstTextPtr

Defines a constant text pointer.

```
typedef const UInt8 * ConstTextPtr;
```

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

Declared In
TextCommon.h

ConstTextToUnicodeInfo

Defines a constant text to Unicode converter object.

```
typedef TextToUnicodeInfo ConstTextToUnicodeInfo;
```

Discussion
The TruncateForTextToUnicode (page 74) function requires a Unicode converter object as a parameter. This function does not modify the contents of the private structure to which the Unicode converter object refers, so it uses the constant Unicode converter object defined by the ConstTextToUnicodeInfo data type.

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

Declared In
UnicodeConverter.h

ConstUniCharArrayPtr

Defines a constant Unicode character array pointer.

```
typedef const UniChar * ConstUniCharArrayPtr;
```

Discussion
You specify a constant Unicode character array pointer for Unicode strings used within the scope of a function whose contents are not modified by that function.

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

Declared In
`TextCommon.h`

ConstUnicodeMappingPtr

Defines a constant Unicode mapping pointer.

```
typedef const UnicodeMapping *     ConstUnicodeMappingPtr;
```

Discussion
Many Unicode Converter functions take a pointer to a Unicode mapping structure as a parameter. For functions that do not modify the Unicode mapping contents, the Unicode Converter provides a constant pointer to a Unicode mapping structure defined by the `ConstUnicodeMappingPtr` data type.

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

Declared In
`UnicodeConverter.h`

ConstUnicodeToTextInfo

Defines a constant Unicode to text converter object.

```
typedef UnicodeToTextInfo ConstUnicodeToTextInfo;
```

Discussion
The `TruncateForUnicodeToText` (page 75) function requires a Unicode converter object as a parameter. This function does not modify the contents of the private structure to which the Unicode converter object refers, so it uses the constant Unicode converter object defined by the `ConstUnicodeToTextInfo` data type.

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

Declared In
`UnicodeConverter.h`

ScriptCodeRun

Contains script code information for a text run.

```
struct ScriptCodeRun {
    ByteOffset offset;
    ScriptCode script;
};
typedef struct ScriptCodeRun ScriptCodeRun;
typedef ScriptCodeRun * ScriptCodeRunPtr;
```

Fields

`offset`

The beginning character position of a text run and its script code in the converted text.

`script`

The script code for the text that begins at the position specified.

Discussion

To return the result of a multiple encoding conversion, the `ConvertFromUnicodeToScriptCodeRun` (page 20) function uses a script code run structure.

The script code run structure uses an extended script code with values in the range 0–254, which are the text encoding base equivalents to Mac OS encodings. Values 0–32 correspond directly to traditional script codes. This allows a script code run to distinguish Icelandic, Turkish, Symbol, Zapf Dingbats, and so on.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextCommon.h`

TECBufferContextRec

Contains buffers for text and text encoding runs.

```
struct TECBufferContextRec {
    TextPtr textInputBuffer;
    TextPtr textInputBufferEnd;
    TextPtr textOutputBuffer;
    TextPtr textOutputBufferEnd;
    TextEncodingRunPtr encodingInputBuffer;
    TextEncodingRunPtr encodingInputBufferEnd;
    TextEncodingRunPtr encodingOutputBuffer;
    TextEncodingRunPtr encodingOutputBufferEnd;
};
typedef struct TECBufferContextRec TECBufferContextRec;
```

Discussion

This structure is used in the `TECConverterContextRec` (page 99) data structure that is used for a TEC plug-in.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECConversionInfo

Contains text encoding conversion information.

```
struct TECConversionInfo {
    TextEncoding sourceEncoding;
    TextEncoding destinationEncoding;
    UInt16 reserved1;
    UInt16 reserved2;
};
typedef struct TECConversionInfo TECConversionInfo;
```

Fields

`sourceEncoding`
: The text encoding specification for the source text.

`destinationEncoding`
: The text encoding specification for the destination text.

`reserved1`
: Reserved.

`reserved2`
: Reserved.

Discussion

When you call the function `TECGetDirectTextEncodingConversions` (page 67), you pass an array of text encoding conversion information structures. The function fills these structures with information about each type of supported conversion.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingConverter.h`

TECConverterContextRec

Contains converter information used by a Text Encoding Converter plug-in.

```
struct TECConverterContextRec {
    Ptr pluginRec;
    TextEncoding sourceEncoding;
    TextEncoding destEncoding;
    UInt32 reserved1;
    UInt32 reserved2;
    TECBufferContextRec bufferContext;
    UInt32 contextRefCon;
    ProcPtr conversionProc;
    ProcPtr flushProc;
    ProcPtr clearContextInfoProc;
    UInt32 options1;
    UInt32 options2;
    TECPluginStateRec pluginState;
};
typedef struct TECConverterContextRec TECConverterContextRec;
```

Fields

`pluginRec`

`sourceEncoding`

The text encoding specification for the source text.

`destEncoding`

The text encoding specification for the destination text.

`reserved1`

Reserved.

`reserved2`

Reserved.

`bufferContext`

`contextRefCon`

A 32-bit value containing or referring to plug-in-specific data.

`conversionProc`

A pointer to a callback for your conversion procedure.

`flushProc`

A pointer to a callback for your reset procedure.

`clearContextInfoProc`

A pointer to a callback for our clear procedure.

`options1`

A 32-bit value that specfies options needed by your plug-in.

`options2`

A 32-bit value that specfies options needed by your plug-in.

`pluginState`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEncodingPlugin.h`

TECInfo

Contains information about the Unicode Converter, the Text Encoding Converter, and Basic Text Types.

```
struct TECInfo {
    UInt16 format;
    UInt16 tecVersion;
    UInt32 tecTextConverterFeatures;
    UInt32 tecUnicodeConverterFeatures;
    UInt32 tecTextCommonFeatures;
    Str31 tecTextEncodingsFolderName;
    Str31 tecExtensionFileName;
    UInt16 tecLowestTEFileVersion;
    UInt16 tecHighestTEFileVersion;
};
typedef struct TECInfo TECInfo;
typedef TECInfo * TECInfoPtr;
```

Fields

`format`

The current format of the returned structure. The format of the structure is indicated by the `kTECInfoCurrentFormat` constant. Any future changes to the format will always be backwardly compatible; any new fields will be added to the end of the structure.

`tecVersion`

The current version of the Text Encoding Conversion Manager extension in BCD (binary coded decimal), with the first byte indicating the major version; for example, 0x0121 for 1.2.1.

`tecTextConverterFeatures`

New features or bug fixes in the Text Encoding Converter. No bits are currently defined.

`tecUnicodeConverterFeatures`

Bit flags indicating new features or bug fixes in the Unicode Converter. See "Unicode Converter Flags" (page 117) for the currently defined bit flags.

`tecTextCommonFeatures`

Bit flags indicating new features or bug fixes in Basic Text Types (the Text Common static library). No bits are currently defined.

`tecTextEncodingsFolderName`

A Pascal string with the (possibly localized) name of the Text Encodings folder.

`tecExtensionFileName`

A Pascal string with the (possibly localized) name of the Text Encoding Conversion Manager extension file.

`tecLowestTEFileVersion`

`tecHighestTEFileVersion`

Discussion

The converter information structure is used by the function `TECGetInfo` (page 69) to hold returned information about the Unicode Converter, the Text Encoding Converter, and Basic Text Types.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextCommon.h`

TECObjectRef

Defines an opaque reference to a converter object.

```
typedef struct OpaqueTECObjectRef * TECObjectRef;
```

Discussion
When making a text conversion, the Text Encoding Converter requires a reference to a converter object that indicates how to accomplish the conversion. Functions, such as `TECCreateConverter` (page 58), that create a converter object return this reference, which you can then pass to other functions when converting text. A converter object reference is defined by the `TECObjectRef` data type.

The structure of the `OpaqueTECObjectRef` data type is private, and a converter object is not accessible directly.

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

Declared In
`TextEncodingConverter.h`

TECPluginDispatchTable

Contains version and signature information and pointers to the callback functions used by a text encoding converter plug-in.

```
struct TECPluginDispatchTable {
    TECPluginVersion version;
    TECPluginVersion compatibleVersion;
    TECPluginSignature PluginID;
    TECPluginNewEncodingConverterPtr PluginNewEncodingConverter;
    TECPluginClearContextInfoPtr PluginClearContextInfo;
    TECPluginConvertTextEncodingPtr PluginConvertTextEncoding;
    TECPluginFlushConversionPtr PluginFlushConversion;
    TECPluginDisposeEncodingConverterPtr PluginDisposeEncodingConverter;
    TECPluginNewEncodingSnifferPtr PluginNewEncodingSniffer;
    TECPluginClearSnifferContextInfoPtr PluginClearSnifferContextInfo;
    TECPluginSniffTextEncodingPtr PluginSniffTextEncoding;
    TECPluginDisposeEncodingSnifferPtr PluginDisposeEncodingSniffer;
   TECPluginGetCountAvailableTextEncodingsPtr PluginGetCountAvailableTextEncodings;
    TECPluginGetCountAvailableTextEncodingPairsPtr
PluginGetCountAvailableTextEncodingPairs;
    TECPluginGetCountDestinationTextEncodingsPtr
PluginGetCountDestinationTextEncodings;
    TECPluginGetCountSubTextEncodingsPtr PluginGetCountSubTextEncodings;
    TECPluginGetCountAvailableSniffersPtr PluginGetCountAvailableSniffers;
    TECPluginGetCountWebEncodingsPtr PluginGetCountWebTextEncodings;
    TECPluginGetCountMailEncodingsPtr PluginGetCountMailTextEncodings;
    TECPluginGetTextEncodingInternetNamePtr PluginGetTextEncodingInternetName;
    TECPluginGetTextEncodingFromInternetNamePtr
PluginGetTextEncodingFromInternetName;
};
typedef struct TECPluginDispatchTable TECPluginDispatchTable;
```

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

Declared In
TextEncodingPlugin.h

TECPluginSig

Defines a data type for a Text Encoding Converter plug-in signature.

```
typedef OSType TECPluginSig;
```

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

Declared In
TextEncodingConverter.h

TECPluginSignature

Defines a data type for a Text Encoding Converter plug-in signature.

```
typedef OSType TECPluginSignature;
```

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

Declared In
TextEncodingConverter.h

TECPluginStateRec

Contains state information for a Text Encoding Converter plug-in.

```
struct TECPluginStateRec {
    UInt8 state1;
    UInt8 state2;
    UInt8 state3;
    UInt8 state4;
    UInt32 longState1;
    UInt32 longState2;
    UInt32 longState3;
    UInt32 longState4;
};
typedef struct TECPluginStateRec TECPluginStateRec;
```

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

Declared In
TextEncodingPlugin.h

TECPluginVersion

Defines a data type for Text Encoding Converter plug-in version.

```
typedef UInt32 TECPluginVersion;
```

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

Declared In
`TextEncodingConverter.h`

TECSnifferContextRec

Contains infomation used by a sniffer object.

```
struct TECSnifferContextRec {
    Ptr pluginRec;
    TextEncoding encoding;
    ItemCount maxErrors;
    ItemCount maxFeatures;
    TextPtr textInputBuffer;
    TextPtr textInputBufferEnd;
    ItemCount numFeatures;
    ItemCount numErrors;
    UInt32 contextRefCon;
    ProcPtr sniffProc;
    ProcPtr clearContextInfoProc;
    TECPluginStateRec pluginState;
};
typedef struct TECSnifferContextRec TECSnifferContextRec;
```

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

Declared In
`TextEncodingPlugin.h`

TECSnifferObjectRef

Defines a reference to an opaque sniffer object.

```
typedef struct OpaqueTECSnifferObjectRef * TECSnifferObjectRef;
```

Discussion
When analyzing text for possible encodings, the Text Encoding Converter requires a reference to a sniffer object that specifies what types of encodings can be detected. You receive this reference when calling the function `TECCreateSniffer` (page 60). A sniffer object reference is defined by the `TECSnifferObjectRef` data type. The structure of the `OpaqueTECObjectRef` data type is private, and a sniffer object is not accessible directly.

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

Declared In
`TextEncodingConverter.h`

TextEncoding

Defines a data type for a text encoding value.

```
typedef UInt32 TextEncoding;
```

Discussion
A `TextEncoding` value is specified by a text encoding base, a text encoding variant, and a text encoding format. You can obtain a `TextEncoding` value by calling the function `CreateTextEncoding` (page 30). When you call this function, you can provide the `TextEncodingBase`, `TextEncodingVariant`, and `TextEncodingFormat` data types.

A `TextEncoding` value is used, for example, to identify the encoding of text passed to a text converter. Two `TextEncoding` values are needed—for source and destination encoding—when calling the Text Encoding Converter or the Unicode Converter to convert text.

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

Declared In
`TextCommon.h`

TextEncodingRun

Contains text encoding information for a text run.

```
struct TextEncodingRun {
    ByteOffset offset;
    TextEncoding textEncoding;
};
typedef struct TextEncodingRun TextEncodingRun;
typedef TextEncodingRun * TextEncodingRunPtr;
```

Fields

`offset`
The beginning character position of a run of text in the converted text string.

`textEncoding`
The encoding of the text run that begins at the position specified.

Discussion
It is not always possible to convert text expressed in Unicode to another single encoding because no other single encoding encompasses the Unicode character encoding range. To adjust for this, you can create a Unicode mapping structure array that specifies the target encodings the Unicode text should be converted to when multiple encodings must be used.

If the `kUnicodeTextRunMask` flag is set, `ConvertFromUnicodeToTextRun` (page 25) and `ConvertFromUnicodeToScriptCodeRun` (page 20) may convert Unicode text to a string of text containing multiple text encoding runs. Each run contains text expressed in a different encoding from that of the preceding or following text segment. For each text encoding run in the string, a `TextEncodingRun` structure indicates the beginning offset and the text encoding for that run.

Functions that convert text from Unicode to a text run return the converted text in an array of text encoding run structures. A text encoding run structure is defined by the `TextEncodingRun` data type.

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

Declared In
`TextCommon.h`

TextEncodingVariant

Defines a data type for a text encoding variant.

```
typedef UInt32 TextEncodingVariant;
```

Discussion
The following enumerations define text encoding variant constants: "Encoding Variants for Big-5" (page 128), "Encoding Variants for MacArabic" (page 129), "Encoding Variants for MacCroatian" (page 130), "Encoding Variants for MacCyrillic" (page 131), "Encoding Variants for MacFarsi" (page 131), "Encoding Variants for MacHebrew" (page 132), "Encoding Variants for MacIcelandic" (page 132), "Encoding Variants for MacJapanese" (page 133), "Encoding Variants for MacRoman Related to Currency" (page 136), "Encoding Variants for MacRomanian" (page 137), "Encoding Variants for MacRomanLatin1" (page 137), "Encoding Variants for MacRoman" (page 134), and "Encoding Variants for MacVT100" (page 138).

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

Declared In
`TextCommon.h`

TextToUnicodeInfo

Defines reference to an opaque Unicode converter object.

```
typedef struct OpaqueTextToUnicodeInfo * TextToUnicodeInfo;
```

Discussion
A Unicode converter object is a private object containing mapping and state information. Many of the Unicode Converter functions that perform conversions require a Unicode converter object containing information used for the conversion process. There are three types of Unicode converter objects, all serving the same purpose but used for different types of conversions. You use the `TextToUnicodeInfo` type, described here, for converting from non-Unicode text to Unicode text.

Because your application cannot directly create or modify the contents of the private Unicode converter object, the Unicode Converter provides functions to create and dispose of it. To create a Unicode converter object for converting from non-Unicode text to Unicode text, your application must first call either the function `CreateTextToUnicodeInfo` (page 30) or the function `CreateTextToUnicodeInfoByEncoding` (page 31) to provide the mapping information required for the conversion. You can then pass this object to the function `ConvertFromTextToUnicode` (page 17) or `ConvertFromPStringToUnicode` (page 16) to identify the information to be used in performing the actual conversion. After you have finished using the object, you should release the memory allocated for it by calling the function `DisposeTextToUnicodeInfo` (page 37). The `TextToUnicodeInfo` data type defines the Unicode converter object.

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

Declared In
UnicodeConverter.h

UniCharArrayOffset

Represents the boundary between two characters.

```
typedef UInt32 UniCharArrayOffset;
```

Discussion
A UniCharArrayOffset represents the boundary between two characters. For example, the first character in a buffer lies between offsets 0 and 1. So the first character in the buffer can be referred to as either "offset 0, leading" or "offset 1, trailing." This distinction is useful when you deal with caret positions.

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

Declared In
TextCommon.h

UnicodeMapping

Contains information for mapping to or from Unicode encoding.

```
struct UnicodeMapping {
    TextEncoding unicodeEncoding;
    TextEncoding otherEncoding;
    UnicodeMapVersion mappingVersion;
};
typedef struct UnicodeMapping UnicodeMapping;
typedef UnicodeMapping * UnicodeMappingPtr;
```

Fields
unicodeEncoding
: A Unicode text encoding specification of type TextEncoding.

otherEncoding
: A text encoding specification for the text to be converted to or from Unicode.

mappingVersion
: The version of the Unicode mapping table to be used.

Discussion
A Unicode mapping structure contains a complete text encoding specification for a Unicode encoding, a complete non-Unicode text encoding specification giving the encoding for the text to be converted to or from Unicode, and the version of the mapping table to be used for conversion. You use a structure of this type to specify the text encodings to and from which the text string is to be converted. A Unicode mapping structure is defined by the UnicodeMapping data type.

You can specify a variety of normalization options by setting up the Unicode mapping structure as described in the following.

To specify normal canonical decomposition according to Unicode 3.2 rules, with no exclusions ("Canonical decomposition 3.2"), set up the UnicodeMapping structure as follows:

```
mapping.unicodeEncoding (in) = Unicode 2.x-3.x, kUnicodeNoSubset,
kUnicode16BitFormat
mapping.otherEncoding (out) = Unicode 2.x-3.x, kUnicodeCanonicalDecompVariant,
  kUnicode16BitFormat
mapping.mappingVersion = kUnicodeUseLatestMapping
```

Examples:

```
u00E0 -> u0061 + u0300
u0061 + u0300 -> u0061 + u0300
u03AC ->  u03B1 + u0301 (3.2 rules)
uF900 -> u8C48
u00E0 + u0323 -> u0061 + u0323 + u0300  (correct)
```

To specify canonical decomposition according to Unicode 3.2 rules, with HFS+ exclusions ("HFS+ decomposition 3.2"), set up the `UnicodeMapping` structure in one of the following ways. The second method is for compatibility with the old method of using `mappingVersion = kUnicodeUseHFSPlusMapping`.

```
// Method 1
mapping.unicodeEncoding (in) = Unicode 2.x-3.x, kUnicodeNoSubset,
kUnicode16BitFormat
mapping.otherEncoding (out) = Unicode 2.x-3.x, kUnicodeHFSPlusDecompVariant,
kUnicode16BitFormat
mapping.mappingVersion = kUnicodeUseLatestMapping
// Method 2
mapping.unicodeEncoding (in) = Unicode 2.x-3.x, kUnicode16BitFormat,
kUnicode16BitFormat
mapping.otherEncoding (out) = Unicode 2.x, kUnicodeCanonicalDecompVariant,
kUnicode16BitFormat
mapping.mappingVersion = kUnicodeUseHFSPlusMapping
```

Examples:

```
u00E0 -> u0061 + u0300
u0061 + u0300 -> u0061 + u0300
u03AC ->  u03B1 + u0301 (3.2 rules)
uF900 -> uF900 (decomposition excluded for HFS+)
u00E0 + u0323 -> u0061 + u0323 + u0300  (correct)
```

To specify normal canonical composition according to Unicode 3.2 rules, set up the `UnicodeMapping` structure as follows:

```
mapping.unicodeEncoding (in) = Unicode 2.x-3.x, kUnicodeNoSubset,
kUnicode16BitFormat
mapping.otherEncoding (out) = Unicode 2.x-3.x, kUnicodeCanonicalCompVariant,
kUnicode16BitFormat
mapping.mappingVersion = kUnicodeUseLatestMapping
```

Examples:

```
u00E0 -> u00E0
u0061 + u0300 -> u00E0
u03AC ->  u03AC
uF900 -> u8C48
u00E0 + u0323 -> u1EA1 u0300 (correct)
```

To specify canonical composition according to Unicode 3.2 rules, but using the HFS+ decomposition exclusions, set up the `UnicodeMapping` structure as follows. This is the form to use if you want to obtain a composed form that dervide from the decomposed form used for HFS+ filenames.

```
mapping.unicodeEncoding (in) = Unicode 2.x-3.x, kUnicodeNoSubset,
kUnicode16BitFormat
mapping.otherEncoding (out) = Unicode 2.x-3.x, kUnicodeHFSPlusCompVariant,
kUnicode16BitFormat
mapping.mappingVersion = kUnicodeUseLatestMapping
```

Examples:

```
u00E0 -> u00E0
u0061 + u0300 -> u00E0
u03AC ->  u03AC
uF900 -> uF900
u00E0 + u0323 -> u1EA1 u0300 (correct)
```

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

Declared In
`UnicodeConverter.h`

UnicodeToTextFallbackUPP

Defines a universal procedure pointer to a Unicode-to-text-fallback callback function.

```
typedef UnicodeToTextFallbackProcPtr UnicodeToTextFallbackUPP;
```

Discussion
For more information, see the description of the `UnicodeToTextFallbackProcPtr` (page 93) callback function.

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

Declared In
`UnicodeConverter.h`

UnicodeToTextInfo

Defines a reference to an opaque Unicode to text converter object.

```
typedef struct OpaqueUnicodeToTextInfo * UnicodeToTextInfo;
```

Discussion
Many of the Unicode Converter functions that perform conversions require a Unicode converter object containing information used for the conversion process. There are three types of Unicode converter objects used for different types of conversions. You use the `UnicodeToTextInfo` type, described here, for converting from Unicode to text.

Because your application cannot directly create or modify the contents of the private Unicode converter object, the Unicode Converter provides functions to create and dispose of it. To create a Unicode converter object for converting from Unicode to text, your application must first call either the function `CreateUnicodeToTextInfo` (page 32) or `CreateUnicodeToTextInfoByEncoding` (page 33).

You can then pass this object to the function `ConvertFromUnicodeToText` (page 23) or `ConvertFromUnicodeToPString` (page 19) to identify the information used to perform the actual conversion. After you have finished using the object, you should release the memory allocated for it by calling the function `DisposeUnicodeToTextInfo` (page 38).

A Unicode converter object for this purpose is defined by the `UnicodeToTextInfo` data type.

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

Declared In
`UnicodeConverter.h`

UnicodeToTextRunInfo

Defines a reference to an opaque Unicode to text run information converter object.

```
typedef struct OpaqueUnicodeToTextRunInfo * UnicodeToTextRunInfo;
```

Discussion
Many of the Unicode Converter functions that perform conversions require a Unicode converter object containing information used for the conversion process. There are three types of Unicode converter objects used for different types of conversions. You use the `UnicodeToTextRunInfo` type, described here, for converting from Unicode to multiple encodings.

Because your application cannot directly create or modify the contents of the private Unicode converter object, the Unicode Converter provides functions to create and dispose of it. You can use any of three functions to create a Unicode converter object for converting from Unicode to multiple encodings. You can use `CreateUnicodeToTextRunInfo` (page 34), `CreateUnicodeToTextRunInfoByEncoding` (page 35), or `CreateUnicodeToTextRunInfoByScriptCode` (page 36).

You can then pass this object to the function `ConvertFromUnicodeToTextRun` (page 25) or `ConvertFromUnicodeToScriptCodeRun` (page 20) to identify the information used to perform the actual conversion. After you have finished using the object, you should release the memory allocated for it by calling the function `DisposeUnicodeToTextRunInfo` (page 38).

A Unicode converter object for this purpose is defined by the `UnicodeToTextRunInfo` data type.

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

Declared In
`UnicodeConverter.h`

Constants

Feature Selectors

Conversion Flags

Specify how to perform conversion of text from one encoding to another.

```
enum {
    kUnicodeUseFallbacksBit = 0,
    kUnicodeKeepInfoBit = 1,
    kUnicodeDirectionalityBits = 2,
    kUnicodeVerticalFormBit = 4,
    kUnicodeLooseMappingsBit = 5,
    kUnicodeStringUnterminatedBit = 6,
    kUnicodeTextRunBit = 7,
    kUnicodeKeepSameEncodingBit = 8,
    kUnicodeForceASCIIRangeBit = 9,
    kUnicodeNoHalfwidthCharsBit = 10,
    kUnicodeTextRunHeuristicsBit = 11,
    kUnicodeMapLineFeedToReturnBit = 12
};
```

Constants

`kUnicodeUseFallbacksBit`

Enables use of fallback mappings.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeKeepInfoBit`

Sets the keep-information control flag.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeDirectionalityBits`

Sets directionality.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeVerticalFormBit`

Sets the vertical form control flag.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeLooseMappingsBit`

Enables use of the loose-mapping portion of a character mapping table.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeStringUnterminatedBit`
Sets the string-unterminated control flag.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeTextRunBit`
Sets the text-run control flag.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeKeepSameEncodingBit`
Sets the keep-same-encoding control flag.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeForceASCIIRangeBit`
Sets the force ASCII range control flag.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeNoHalfwidthCharsBit`
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeTextRunHeuristicsBit`
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeMapLineFeedToReturnBit`
Available in Mac OS X v10.2 and later.
Declared in `UnicodeConverter.h`.

Conversion Masks

Set or text for conversion flags.

```
enum {
    kUnicodeUseFallbacksMask = 1L << kUnicodeUseFallbacksBit,
    kUnicodeKeepInfoMask = 1L << kUnicodeKeepInfoBit,
    kUnicodeDirectionalityMask = 3L << kUnicodeDirectionalityBits,
    kUnicodeVerticalFormMask = 1L << kUnicodeVerticalFormBit,
    kUnicodeLooseMappingsMask = 1L << kUnicodeLooseMappingsBit,
    kUnicodeStringUnterminatedMask = 1L << kUnicodeStringUnterminatedBit,
    kUnicodeTextRunMask = 1L << kUnicodeTextRunBit,
    kUnicodeKeepSameEncodingMask = 1L << kUnicodeKeepSameEncodingBit,
    kUnicodeForceASCIIRangeMask = 1L << kUnicodeForceASCIIRangeBit,
    kUnicodeNoHalfwidthCharsMask = 1L << kUnicodeNoHalfwidthCharsBit,
    kUnicodeTextRunHeuristicsMask = 1L << kUnicodeTextRunHeuristicsBit,
    kUnicodeMapLineFeedToReturnMask = 1L << kUnicodeMapLineFeedToReturnBit
};
```

Constants

`kUnicodeUseFallbacksMask`

A mask for setting the Unicode-use-fallbacks conversion flag. The Unicode Converter uses fallback mappings when it encounters a source text element for which there is no equivalent destination encoding. Fallback mappings are mappings that do not preserve the meaning or identity of the source character but represent a useful approximation of it. See the function `SetFallbackUnicodeToText` (page 48).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeKeepInfoMask`

A mask for setting the keep-information control flag which governs whether the Unicode Converter keeps the current state stored in the Unicode converter object before converting the text string.

If you clear this flag, the converter will initialize the Unicode converter object before converting the text string and assume that subsequent calls do not need any context, such as direction state for the current call.

If you set the flag, the converter uses the current state. This is useful if your application must convert a stream of text in pieces that are not block delimited. You should set this flag for each call in a series of calls on the same text stream.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeDirectionalityMask`

A mask for setting the directionality control flag

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeVerticalFormMask`

A mask for setting the vertical form control flag. The vertical form control flag tells the Unicode Converter how to map text elements for which there are both abstract and vertical presentation forms in the destination encoding.

If set, the converter maps these text elements to their vertical forms, if they are available.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeLooseMappingsMask`

A mask that determines whether the Unicode Converter should use the loose-mapping portion of a mapping table for character mapping if the strict mapping portion of the table does not include a destination encoding equivalent for the source text element.

If you clear this flag, the converter will use only the strict equivalence portion.

If set this flag and a conversion for the source text element does not exist in the strict equivalence portion of the mapping table, then the converter uses the loose mapping section.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeStringUnterminatedMask`

A mask for setting the string-unterminated control flag. Determines how the Unicode Converter handles text-element boundaries and direction resolution at the end of an input buffer.

If you clear this bit, the converter treats the end of the buffer as the end of text.

If you set this bit, the converter assumes that the next call you make using the current context will supply another buffer of text that should be treated as a continuation of the current text. For example, if the last character in the input buffer is `'A'`, `ConvertFromUnicodeToText` stops conversion at the `'A'` and returns `kTECIncompleteElementErr`, because the next buffer could begin with a combining diacritical mark that should be treated as part of the same text element. If the last character in the input buffer is a control character, `ConvertFromUnicodeToText` does not return `kTECIncompleteElementErr` because a control character could not be part of a multiple character text element.

In attempting to analyze the text direction, when the Unicode Converter reaches the end of the current input buffer and the direction of the current text element is still unresolved, if you clear this flag, the converter treats the end of the buffer as a block separator for direction resolution. If you set this flag, it sets the direction as undetermined

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeTextRunMask`

A mask for setting the text-run control flag which determines how the Unicode Converter converts Unicode text to a non-Unicode encoding when more than one possible destination encoding exists.

If you clear this flag, the function `ConvertFromUnicodeToTextRun` (page 25) or `ConvertFromUnicodeToScriptCodeRun` (page 20) attempts to convert the Unicode text to the single encoding from the list of encodings in the Unicode converter object that produces the best result, that is, that provides for the greatest amount of source text conversion.

If you set this flag, `ConvertFromUnicodeToTextRun` or `ConvertFromUnicodeToScriptCodeRun`, which are the only functions to which it applies, may generate a destination string that combines text in any of the encodings specified by the Unicode converter object.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeKeepSameEncodingMask`

A mask for setting the keep-same-encoding control flag. Determines how the Unicode Converter treats the conversion of Unicode text following a text element that could not be converted to the first destination encoding when multiple destination encodings exist. This control flag applies only if the `kUnicodeTextRunMask` control flag is set.

If you set this flag, the function `ConvertFromUnicodeToTextRun` (page 25) attempts to minimize encoding changes in the conversion of the source text string; that is, once it is forced to make an encoding change, it attempts to use that encoding as the conversion destination for as long as possible.

If you clear this flag, `ConvertFromUnicodeToTextRun` attempts to keep most of the converted string in one encoding, switching to other encodings only when necessary.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeForceASCIIRangeMask`

A mask for setting the force ASCII range control flag. If an encoding normally treats 1-byte code points `0x00`–`0x7F` as an ISO 646 national variant that is different from ASCII, setting this flag forces `0x00`–`0x7F` to be treated as ASCII. For example, Japanese encodings such as Shift-JIS generally treat `0x00`–`0x7F` as JIS Roman, with `0x5C` as YEN SIGN instead of REVERSE SOLIDUS, but when converting a DOS file path you may want to set this flag so that `0x5C` is mapped as REVERSE SOLIDUS.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeNoHalfwidthCharsMask`

Sets the no halfwidth characters control flag.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeTextRunHeuristicsMask`

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMapLineFeedToReturnMask`

Specifies mapping o the LF (LineFeed) character used in Unix to represent new lines to the CR (CarriageReturn) used in Mac encodings. This option has an effect only when used with the constant `kUnicodeLooseMappingsMask`. You can pass both constants as `iControlFlags` parameters to the functions `ConvertFromUnicodeToText`, `ConvertFromUnicodeToTextRun`, and `ConvertFromUnicodeToScriptCodeRun`.

Available in Mac OS X v10.2 and later.

Declared in `UnicodeConverter.h`.

Discussion

You use these constants to specify how the conversion of text from one encoding to another is performed. You use these masks as the `controlFlags` parameter in the `ConvertFromTextToUnicode` (page 17), `ConvertFromUnicodeToText` (page 23), `ConvertFromUnicodeToScriptCodeRun` (page 20), `ConvertFromUnicodeToTextRun` (page 25), and `TruncateForUnicodeToText` (page 75) functions. A different subset of control masks applies to each of these functions. Using the bitmask constants, you can perform a bitwise `OR` operation to set the pertinent flags for a particular function's parameters. For example, when you call a function, you might pass the following `controlFlags` parameter setting:

```
controlflags=kUnicodeUseFallbacksMask |kUnicodeLooseMappingsMask;
```

Directionality Flags

Specify a text direction.

```
enum {
    kUnicodeDefaultDirection = 0,
    kUnicodeLeftToRight = 1,
    kUnicodeRightToLeft = 2
};
```

Constants

`kUnicodeDefaultDirection`

Use the default direction.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeLeftToRight`

Indicates left to right direction.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeRightToLeft`

Indicates right to left direction.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

Directionality Masks

Set or text for directionality bits.

```
enum {
    kUnicodeDefaultDirectionMask = kUnicodeDefaultDirection <<
kUnicodeDirectionalityBits,
    kUnicodeLeftToRightMask = kUnicodeLeftToRight << kUnicodeDirectionalityBits,
    kUnicodeRightToLeftMask = kUnicodeRightToLeft << kUnicodeDirectionalityBits
};
```

Constants

`kUnicodeDefaultDirectionMask`

A mask for setting the global, or base, line direction for the text being converted. The value `kUnicodeDefaultDirectionMask` tells the converter to use the value of the first strong direction character in the string. This determines which direction the converter should use for resolution of neutral coded characters, such as spaces that occur between sets of coded characters having different directions—for example, between Latin and Arabic characters—rendering ambiguous the direction of the space character.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeLeftToRightMask`
A mask for setting the global, or base, line direction for the text being converted. The value `kUnicodeLeftToRightMask` tells the converter that the base paragraph direction is left to right. This determines which direction the converter should use for resolution of neutral coded characters, such as spaces that occur between sets of coded characters having different directions—for example, between Latin and Arabic characters—rendering ambiguous the direction of the space character.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeRightToLeftMask`
The value `kUnicodeRightToLeftMask` tells the converter that the base paragraph direction is right to left. This determines which direction the converter should use for resolution of neutral coded characters, such as spaces that occur between sets of coded characters having different directions—for example, between Latin and Arabic characters—rendering ambiguous the direction of the space character.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

Unicode Converter Flags

Specify features for bug fixes in the Unicode Converter.

```
enum {
    kTECKeepInfoFixBit = 0,
    kTECFallbackTextLengthFixBit = 1,
    kTECTextRunBitClearFixBit = 2,
    kTECTextToUnicodeScanFixBit = 3,
    kTECAddForceASCIIChangesBit = 4,
    kTECPreferredEncodingFixBit = 5,
    kTECAddTextRunHeuristicsBit = 6,
    kTECAddFallbackInterruptBit = 7
};
```

Constants

`kTECKeepInfoFixBit`
This is set if the Unicode Converter has a bug fix to stop ignoring certain control flags

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTECFallbackTextLengthFixBit`
This is set if the Unicode Converter has a bug fix to use the source length (`srcConvLen`) and destination length (`destConvLen`) returned by a caller-supplied fall-back handler for any status it returns except `kTECUnmappableElementErr`. Previously it honored only these values if `noErr` was returned.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTECTextRunBitClearFixBit`
This is set if `ConvertFromUnicodeToTextRun` and `ConvertFromUnicodeToScriptCodeRun` function correctly if the `kUnicodeTextRunBit` is clear.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTECTextToUnicodeScanFixBit`
: This is set if `ConvertFromTextToUnicode` is enhanced so mappings can depend on context and saved state. The consequences of this are (1) malformed input results in `kTextMalformedInputErr`; (2) `ConvertFromTextToUnicode` accepts the control flags `kUnicodeLooseMappingsMask`, `kUnicodeKeepInfoMask`, and `kUnicodeStringUnterminatedMask`; (3) elimination of redundant direction overrides when converting Mac OS Arabic and Hebrew to Unicode; and (4) improved mapping of 0x30-0x39 digits in Mac OS Arabic when loose mappings are used.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTECAddForceASCIIChangesBit`
: This is set if the new control flag bits `kUnicodeForceASCIIRangeBit` and `kUnicodeNoHalfwidthCharsBit` are supported for use with the functions `ConvertFromTextToUnicode`, `ConvertFromUnicodeToText`, and so forth.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTECPreferredEncodingFixBit`
: This is set to indicate that if a preferred encoding is specified for `CreateUnicodeToTextRunInfo` and related functions, they handle it correctly even if it does not match the system script.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

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

Declared in `TextCommon.h`.

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

Declared in `TextCommon.h`.

Discussion
These are bit flags to indicate new features for bug fixes in the Unicode Converter. They are used by the `TECInfo` (page 101) data type.

Unicode Converter Masks

Set or test for Unicode converter flags.

```
enum {
  kTECKeepInfoFixMask           = 1L << kTECKeepInfoFixBit,
  kTECFallbackTextLengthFixMask = 1L << kTECFallbackTextLengthFixBit,
  kTECTextRunBitClearFixMask    = 1L << kTECTextRunBitClearFixBit,
  kTECTextToUnicodeScanFixMask  = 1L << kTECTextToUnicodeScanFixBit,
  kTECAddForceASCIIChangesMask  = 1L << kTECAddForceASCIIChangesBit,
  kTECPreferredEncodingFixMask  = 1L << kTECPreferredEncodingFixBit,
  kTECAddTextRunHeuristicsMask  = 1L << kTECAddTextRunHeuristicsBit,
  kTECAddFallbackInterruptMask  = 1L << kTECAddFallbackInterruptBit
};
```

Unicode Fallback Sequencing Flag

Specifies options for setting fallback sequencing.

```
enum {
    kUnicodeFallbackSequencingBits = 0
};
```

Unicode Fallback Sequencing Masks

Set or text for Unicode sequencing flag.

```
enum {
    kUnicodeFallbackSequencingMask = 3L << kUnicodeFallbackSequencingBits,
    kUnicodeFallbackInterruptSafeMask = 1L << 2
};
```

Constants

`kUnicodeFallbackSequencingMask`

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeFallbackInterruptSafeMask`

Indicate that the caller's fallback routine doesn't move memory.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

Unicode Matching Flags

Specify matching criteria for Unicode mappings.

```
enum {
    kUnicodeMatchUnicodeBaseBit = 0,
    kUnicodeMatchUnicodeVariantBit = 1,
    kUnicodeMatchUnicodeFormatBit = 2,
    kUnicodeMatchOtherBaseBit = 3,
    kUnicodeMatchOtherVariantBit = 4,
    kUnicodeMatchOtherFormatBit = 5
};
```

Constants

`kUnicodeMatchUnicodeBaseBit`
Excludes mappings that do not match the text encoding base of the `unicodeEncoding` field of the structure `UnicodeMapping` (page 107).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchUnicodeVariantBit`
Excludes mappings that do not match the text encoding variant of the `unicodeEncoding` field of the specified Unicode mapping structure.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchUnicodeFormatBit`
Excludes mappings that do not match the text encoding format of the `unicodeEncoding` field of the specified Unicode mapping structure.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchOtherBaseBit`
Excludes mappings that do not match the text encoding base of the `otherEncoding` field of the structure `UnicodeMapping` (page 107).

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchOtherVariantBit`
Excludes mappings that do not match the text encoding variant of the `otherEncoding` field of the specified Unicode mapping structure.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchOtherFormatBit`
Excludes mappings that do not match the text encoding format of the `otherEncoding` field of the specified Unicode mapping structure.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

Unicode Matching Masks

Used to set or test for Unicode matching flags.

```
enum {
    kUnicodeMatchUnicodeBaseMask = 1L << kUnicodeMatchUnicodeBaseBit,
    kUnicodeMatchUnicodeVariantMask = 1L << kUnicodeMatchUnicodeVariantBit,
    kUnicodeMatchUnicodeFormatMask = 1L << kUnicodeMatchUnicodeFormatBit,
    kUnicodeMatchOtherBaseMask = 1L << kUnicodeMatchOtherBaseBit,
    kUnicodeMatchOtherVariantMask = 1L << kUnicodeMatchOtherVariantBit,
    kUnicodeMatchOtherFormatMask = 1L << kUnicodeMatchOtherFormatBit
};
```

Constants

`kUnicodeMatchUnicodeBaseMask`
If set, excludes mappings that do not match the text encoding base of the `unicodeEncoding` field of the structure `UnicodeMapping` (page 107). If not set, the function ignores the text encoding base of that field.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchUnicodeVariantMask`
If set, excludes mappings that do not match the text encoding variant of the `unicodeEncoding` field of the specified Unicode mapping structure. If not set, the function ignores the text encoding variant of that field.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchUnicodeFormatMask`
If set, excludes mappings that do not match the text encoding format of the `unicodeEncoding` field of the specified Unicode mapping structure. If not set, the function ignores the text encoding format of that field.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchOtherBaseMask`
If set, excludes mappings that do not match the text encoding base of the `otherEncoding` field of the structure `UnicodeMapping` (page 107). If not set, the function ignores the text encoding base of that field.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchOtherVariantMask`
If set, excludes mappings that do not match the text encoding variant of the `otherEncoding` field of the specified Unicode mapping structure. If not set, the function ignores the text encoding variant of that field.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeMatchOtherFormatMask`
If set, excludes mappings that do not match the text encoding format of the `otherEncoding` field of the specified Unicode mapping structure. If not set, the function ignores the text encoding format of that field.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

Fallback Handler Selectors

Specify a fallback handler for the Unicode Converter to use.

```
enum {
    kUnicodeFallbackDefaultOnly = 0,
    kUnicodeFallbackCustomOnly = 1,
    kUnicodeFallbackDefaultFirst = 2,
    kUnicodeFallbackCustomFirst = 3
};
```

Constants

`kUnicodeFallbackDefaultOnly`
Use the default fallback handler only.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeFallbackCustomOnly`
Use the custom fallback handler only.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeFallbackDefaultFirst`
Use the default fallback handler first, then the custom one.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

`kUnicodeFallbackCustomFirst`
Use the custom fallback handler first, then the default one.
Available in Mac OS X v10.0 and later.
Declared in `UnicodeConverter.h`.

Discussion

Used to specify which fallback handler the Unicode Converter should use. If you use both the custom and default handlers, you can set the order in which they are called. You use these constants to set the `controlFlags` parameter of the `SetFallbackUnicodeToText` (page 48) and `SetFallbackUnicodeToTextRun` (page 49) functions.

Encodings and Variants

Base Text Encodings

Specify base text encodings.

```
typedef UInt32 TextEncodingBase;
enum {
    kTextEncodingMacRoman = 0,
    kTextEncodingMacJapanese = 1,
    kTextEncodingMacChineseTrad = 2,
    kTextEncodingMacKorean = 3,
    kTextEncodingMacArabic = 4,
    kTextEncodingMacHebrew = 5,
    kTextEncodingMacGreek = 6,
    kTextEncodingMacCyrillic = 7,
    kTextEncodingMacDevanagari = 9,
    kTextEncodingMacGurmukhi = 10,
    kTextEncodingMacGujarati = 11,
    kTextEncodingMacOriya = 12,
    kTextEncodingMacBengali = 13,
    kTextEncodingMacTamil = 14,
    kTextEncodingMacTelugu = 15,
    kTextEncodingMacKannada = 16,
    kTextEncodingMacMalayalam = 17,
    kTextEncodingMacSinhalese = 18,
    kTextEncodingMacBurmese = 19,
    kTextEncodingMacKhmer = 20,
    kTextEncodingMacThai = 21,
    kTextEncodingMacLaotian = 22,
    kTextEncodingMacGeorgian = 23,
    kTextEncodingMacArmenian = 24,
    kTextEncodingMacChineseSimp = 25,
    kTextEncodingMacTibetan = 26,
    kTextEncodingMacMongolian = 27,
    kTextEncodingMacEthiopic = 28,
    kTextEncodingMacCentralEurRoman = 29,
    kTextEncodingMacVietnamese = 30,
    kTextEncodingMacExtArabic = 31,
    kTextEncodingMacSymbol = 33,
    kTextEncodingMacDingbats = 34,
    kTextEncodingMacTurkish = 35,
    kTextEncodingMacCroatian = 36,
    kTextEncodingMacIcelandic = 37,
    kTextEncodingMacRomanian = 38,
    kTextEncodingMacCeltic = 39,
    kTextEncodingMacGaelic = 40,
    kTextEncodingMacKeyboardGlyphs = 41
};
```

Constants

`kTextEncodingMacRoman`

The encoding for Mac OS Roman.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingMacJapanese`

The encoding for Mac OS Japanese.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingMacChineseTrad`
The encoding for Mac OS traditional Chinese.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacKorean`
The encoding for Mac OS Korean.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacArabic`
The encoding for Mac OS Arabic.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacHebrew`
The encoding for Mac OS Hebrew.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacGreek`
The encoding for Mac OS Greek.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacCyrillic`
The encoding for Mac OS Cyrillic.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacDevanagari`
The encoding for Mac OS Devanagari.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacGurmukhi`
The encoding for Mac OS Gurmukhi.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacGujarati`
The encoding for Mac OS Gujurati.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacOriya`
The encoding for Mac OS Oriya.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacBengali`
The encoding for Mac OS Bengali.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacTamil`
The encoding for Mac OS Tamil.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacTelugu`
The encoding for Mac OS Telugu.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacKannada`
The encoding for Mac OS Kannada.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacMalayalam`
The encoding for Mac OS Malayalam.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacSinhalese`
The encoding for Mac OS Sinhalese.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacBurmese`
The encoding for Mac OS Burmese.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacKhmer`
The encoding for Mac OS Khmer.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacThai`
The encoding for Mac OS Thai.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacLaotian`
The encoding for Mac OS Laotian.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacGeorgian`
The encoding for Mac OS Georgian.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacArmenian`
The encoding for Mac OS Armenian.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacChineseSimp`
The encoding for Mac OS simple Chinese.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacTibetan`
The encoding for Mac OS Tibetan.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacMongolian`
The encoding for Mac OS Mongolian.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacEthiopic`
The encoding for Mac OS Ethiopic.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacCentralEurRoman`
The encoding for Mac OS Central European Roman.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacVietnamese`
The encoding for Mac OS Vietnamese.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacExtArabic`
The encoding for Mac OS ExtArabic.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacSymbol`
This Mac OS encoding uses script code 0, `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacDingbats`
This Mac OS encoding uses script code 0, `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacTurkish`
This Mac OS encoding uses script code 0, `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacCroatian`
This Mac OS encoding uses script code 0, `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacIcelandic`
This Mac OS encoding uses script code 0, `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacRomanian`
This Mac OS encoding uses script code 0, `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacCeltic`
This Mac OS encoding uses script code 0, `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacGaelic`
This Mac OS encoding uses script code 0, `smRoman`.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingMacKeyboardGlyphs`
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

Discussion
You use a base text encoding data type to specify which text encoding or text encoding scheme you have used to express a given text. The text encoding base value is the primary specification of the source or target encoding. Values 0 through 32 correspond directly to Mac OS script codes. Values 33 through 254 are for other Mac OS encodings that do not have their own script codes, such as the Symbol encoding implemented by the Symbol font. You can also specify a meta-value as a base text encoding, such as `kTextEncodingMacHFS` and `kTextEncodingUnicodeDefault`. A meta-value is mapped to a real value.

The function `GetTextEncodingBase` (page 39) returns the text encoding base of a text encoding specification.

A base text encoding is defined by the `TextEncodingBase` data type.

Compatibility TextEncodings

Specify text encodings that are provided for backward compatibility.

```
enum {
    kTextEncodingMacTradChinese = kTextEncodingMacChineseTrad,
    kTextEncodingMacRSymbol = 8,
    kTextEncodingMacSimpChinese = kTextEncodingMacChineseSimp,
    kTextEncodingMacGeez = kTextEncodingMacEthiopic,
    kTextEncodingMacEastEurRoman = kTextEncodingMacCentralEurRoman,
    kTextEncodingMacUninterp = 32
};
```

EBCDIC and IBM Host Text Encodings

Specify text encodings used by IBM computers.

```
enum {
    kTextEncodingEBCDIC_US = 0x0C01,
    kTextEncodingEBCDIC_CP037 = 0x0C02
};
```

Constants

`kTextEncodingEBCDIC_US`
Basic EBCDIC-US encoding.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingEBCDIC_CP037`
Code page 037, extended EBCDIC-US Latin1.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

Discussion

EBCDIC (Extended Binary- Coded Decimal Interchange Code) is used by IBM computers to represent characters as numbers.

Encoding Variants for Big-5

Specify variants of Big-5 encoding.

```
enum {
    kBig5_BasicVariant = 0,
    kBig5_StandardVariant = 1,
    kBig5_ETenVariant = 2
};
```

Constants

`kBig5_BasicVariant`
The basic encoding variant.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kBig5_StandardVariant`
The standard variant; 0xC6A1-0xC7FC: kana, Cyrillic, enclosed numerics.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kBig5_ETenVariant`
Adds kana, Cyrillic, radicals, and so forth with high-bytes C6-C8, F9.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Discussion
Big-5 encoding was developed by five companies as a character set standard in Tawain.

Encoding Variants for Mac OS Encodings

Specify variant Mac OS encodings that use script codes other than 0

```
enum {
    kTextEncodingMacFarsi = 0x8C,
    kTextEncodingMacUkrainian = 0x98,
    kTextEncodingMacInuit = 0xEC,
    kTextEncodingMacVT100 = 0xFC
};
```

Constants
`kTextEncodingMacFarsi`
Uses script code 4, `smArabic`. It is similar to Mac Arabic but uses Farsi digits.]

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingMacUkrainian`
Uses script code 7, `smCyrillic`.]

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingMacInuit`
Uses script code 28, `smEthiopic`.]

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingMacVT100`
Uses script code 32, `smUninterp`; VT100/102 font from the common toolbox; Latin-1 characters plus box drawing.]

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacArabic

Specify variants of MacArabic.

```
enum {
    kMacArabicStandardVariant = 0,
    kMacArabicTrueTypeVariant = 1,
    kMacArabicThuluthVariant = 2,
    kMacArabicAlBayanVariant = 3
};
```

Constants

`kMacArabicStandardVariant`
A Mac OS Arabic variant is supported by the Cairo font (the system font for Arabic) and is the encoding supported by the text processing utilities.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacArabicTrueTypeVariant`
A Mac OS Arabic variant used for most of the Arabic TrueType fonts: Baghdad, Geeza, Kufi, Nadeem.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacArabicThuluthVariant`
A Mac OS Arabic variant used for the Arabic PostScript-only fonts: Thuluth and Thuluth bold.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacArabicAlBayanVariant`
A Mac OS Arabic variant used for the Arabic TrueType font Al Bayan.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacCroatian

Specify variants of MacCroation.

```
enum {
    kMacCroatianDefaultVariant = 0,
    kMacCroatianCurrencySignVariant = 1,
    kMacCroatianEuroSignVariant = 2
};
```

Constants

`kMacCroatianDefaultVariant`
This is a meta value that maps to one of the following constants, depending on version of the Mac OS.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacCroatianCurrencySignVariant`
In versions of Mac OS earlier than 8.5, 0xDB is the currency sign.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacCroatianEuroSignVariant`
In Mac OS version 8.5 and later, 0xDB is the Euro sign.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacCyrillic

Specify variants of MacCyrillic.

```
enum {
    kMacCyrillicDefaultVariant = 0,
    kMacCyrillicCurrSignStdVariant = 1,
    kMacCyrillicCurrSignUkrVariant = 2,
    kMacCyrillicEuroSignVariant = 3
};
```

Constants

`kMacCyrillicDefaultVariant`
This is a meta value that maps to one of the following constants, depending on version of the Mac OS.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacCyrillicCurrSignStdVariant`
In Mac OS versions prior to 9.0 (RU, BG), 0xFF = currency sign, 0xA2/0xB6 = CENT / PARTIAL DIFF.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacCyrillicCurrSignUkrVariant`
In Mac OS version 9.0 and later (UA, LangKit), 0xFF = currency sign, 0xA2/0xB6 = GHE with upturn.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacCyrillicEuroSignVariant`
In Mac OS 9.0 and later, 0xFF is Euro sign, 0xA2/0xB6 = GHE with upturn.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacFarsi

Specify variants of MacFarsi.

```
enum {
    kMacFarsiStandardVariant = 0,
    kMacFarsiTrueTypeVariant = 1
};
```

Constants

`kMacFarsiStandardVariant`

This Mac OS Farsi variant is supported by the Tehran font (the system font for Farsi) and is the encoding supported by the text processing utilities.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacFarsiTrueTypeVariant`

This Mac OS Farsi variant is used for most of the Farsi TrueType fonts: Ashfahan, Amir, Kamran, Mashad, NadeemFarsi.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacHebrew

Specify variants of MacHebrew.

```
enum {
    kMacHebrewStandardVariant = 0,
    kMacHebrewFigureSpaceVariant = 1
};
```

Constants

`kMacHebrewStandardVariant`

The standard Mac OS Hebrew variant.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacHebrewFigureSpaceVariant`

The Mac OS Hebrew variant in which 0xD4 represents figure space, not left single quotation mark as in the standard variant.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacIcelandic

Specify variants of MacIcelandic.

```
enum {
    kMacIcelandicStdDefaultVariant = 0,
    kMacIcelandicTTDefaultVariant = 1,
    kMacIcelandicStdCurrSignVariant = 2,
    kMacIcelandicTTCurrSignVariant = 3,
    kMacIcelandicStdEuroSignVariant = 4,
    kMacIcelandicTTEuroSignVariant = 5
};
```

Constants

`kMacIcelandicStdDefaultVariant`
This is a meta value that maps to `kMacIcelandicStdCurrSignVariant` or `kMacIcelandicStdEuroSignVariant`, depending on version of the Mac OS.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacIcelandicTTDefaultVariant`
This is a meta value that maps to `kMacIcelandicTTCurrSignVariant` or `kMacIcelandicTTEuroSignVariant`, depending on version of the Mac OS.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacIcelandicStdCurrSignVariant`
In Mac OS versions prior to 8.5, 0xDB is the currency sign; 0xBB/0xBC are fem./masc. ordinal indicators.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacIcelandicTTCurrSignVariant`
In Mac OS versions prior to 8.5, 0xDB is the currency sign; 0xBB/0xBC are fi/fl ligatures

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacIcelandicStdEuroSignVariant`
In Mac OS version 8.5 and later , 0xDB is the Euro sign ; 0xBB/0xBC are fem./masc. ordinal indicators.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacIcelandicTTEuroSignVariant`
In Mac OS versions earlier than 8.5, 0xDB is the Euro sign; 0xBB/0xBC are fi/fl ligatures.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacJapanese

Specify variants of MacJapanese.

```
enum {
    kMacJapaneseStandardVariant = 0,
    kMacJapaneseStdNoVerticalsVariant = 1,
    kMacJapaneseBasicVariant = 2,
    kMacJapanesePostScriptScrnVariant = 3,
    kMacJapanesePostScriptPrintVariant = 4,
    kMacJapaneseVertAtKuPlusTenVariant = 5
};
```

Constants

`kMacJapaneseStandardVariant`
The standard Mac OS Japanese variant. Shift-JIS with JIS Roman modifications, extra 1-byte characters, 2-byte Apple extensions, and some vertical presentation forms in the range 0xEB40—0xEDFE ("ku plus 84").

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacJapaneseStdNoVerticalsVariant`
An artificial Mac OS Japanese variant for callers who don't want to use separately encoded vertical forms (for example, developers using QuickDraw GX).

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacJapaneseBasicVariant`
An artificial Mac OS Japanese variant without Apple double-byte extensions.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacJapanesePostScriptScrnVariant`
The Mac OS Japanese variant for the screen bitmap version of the Sai Mincho and Chu Gothic fonts.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacJapanesePostScriptPrintVariant`
The Mac OS Japanese variant for PostScript printing versions of the Sai Mincho and Chu Gothic PostScript fonts. This version includes double-byte half-width characters in addition to single-byte half-width characters.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacJapaneseVertAtKuPlusTenVariant`
The Mac OS Japanese variant for the Hon Mincho and Maru Gothic fonts used in the Japanese localized version of System 7.1. It does not include the standard Apple extensions, and encodes vertical forms at a different location.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacRoman

Specify variants of MacRoman.

```
enum {
    kMacRomanStandardVariant = 0,
    kMacIcelandicStandardVariant = 0,
    kMacIcelandicTrueTypeVariant = 1,
    kJapaneseStandardVariant = 0,
    kJapaneseStdNoVerticalsVariant = 1,
    kJapaneseBasicVariant = 2,
    kJapanesePostScriptScrnVariant = 3,
    kJapanesePostScriptPrintVariant = 4,
    kJapaneseVertAtKuPlusTenVariant = 5,
    kHebrewStandardVariant = 0,
    kHebrewFigureSpaceVariant = 1,
    kUnicodeMaxDecomposedVariant = 2,
    kUnicodeNoComposedVariant = 3,
    kJapaneseNoOneByteKanaOption = 0x20,
    kJapaneseUseAsciiBackslashOption = 0x40
};
```

Constants

`kMacRomanStandardVariant`
The standard variant of Mac OS Roman for Mac OS 8.5 and later; 0xDB is the Euro sign.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacIcelandicStandardVariant`
The standard Mac OS Icelandic encoding supported by the bitmap versions of Chicago, Geneva, Monaco, and New York in the Icelandic system. This is also the variant supported by the text processing utilities.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacIcelandicTrueTypeVariant`
The Mac OS Icelandic variant used for the bitmap versions of Courier, Helvetica, Palatino, and Times in the Icelandic system, and for the TrueType versions of Chicago, Geneva, Monaco, New York, Courier, Helvetica, Palatino, and Times.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kJapaneseStandardVariant`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kJapaneseStdNoVerticalsVariant`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kJapaneseBasicVariant`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kJapanesePostScriptScrnVariant`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kJapanesePostScriptPrintVariant`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kJapaneseVertAtKuPlusTenVariant`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kHebrewStandardVariant`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kHebrewFigureSpaceVariant`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeMaxDecomposedVariant`
Replaced by `kUnicodeCanonicalDecompVariant`.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeNoComposedVariant`
Replaced by `kUnicodeCanonicalCompVariant`.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kJapaneseNoOneByteKanaOption`
Replaced by Unicode Converter option `kUnicodeNoHalfwidthCharsBit`.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `TextCommon.h`.

`kJapaneseUseAsciiBackslashOption`
Replaced by Unicode Converter option `kUnicodeForceASCIIRangeBit`.

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `TextCommon.h`.

Encoding Variants for MacRoman Related to Currency

Specify variants of MacRoman that are related to currency.

```
enum {
    kMacRomanDefaultVariant = 0,
    kMacRomanCurrencySignVariant = 1,
    kMacRomanEuroSignVariant = 2
};
```

Constants

`kMacRomanDefaultVariant`
This is a meta value that maps to one of the following constants, depending on version of the Mac OS.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacRomanCurrencySignVariant`
: In Mac OS versions earlier than 8.5 0xDB is the currency sign; still used for some older fonts even in Mac OS 8.5.

  Available in Mac OS X v10.0 and later.

  Declared in `TextCommon.h`.

`kMacRomanEuroSignVariant`
: In Mac OS version 8.5 and later, 0xDB is the Euro sign.

  Available in Mac OS X v10.0 and later.

  Declared in `TextCommon.h`.

Encoding Variants for MacRomanian

Specify variants of MacRomanian.

```
enum {
    kMacRomanianDefaultVariant = 0,
    kMacRomanianCurrencySignVariant = 1,
    kMacRomanianEuroSignVariant = 2
};
```

Constants

`kMacRomanianDefaultVariant`
: This is a meta value that maps to one of the following constants, depending on version of the Mac OS.

  Available in Mac OS X v10.0 and later.

  Declared in `TextCommon.h`.

`kMacRomanianCurrencySignVariant`
: In Mac OS versions earlier than 8.5, 0xDB is the currency sign.

  Available in Mac OS X v10.0 and later.

  Declared in `TextCommon.h`.

`kMacRomanianEuroSignVariant`
: In Mac OS version 8.5 and later, 0xDB is the Euro sign.

  Available in Mac OS X v10.0 and later.

  Declared in `TextCommon.h`.

Encoding Variants for MacRomanLatin1

Specify variants of MacRomanLatin1.

```
enum {
    kMacRomanLatin1DefaultVariant = 0,
    kMacRomanLatin1StandardVariant = 2,
    kMacRomanLatin1TurkishVariant = 6,
    kMacRomanLatin1CroatianVariant = 8,
    kMacRomanLatin1IcelandicVariant = 11,
    kMacRomanLatin1RomanianVariant = 14
};
```

Constants

`kMacRomanLatin1DefaultVariant`
This is a meta value that maps to one of the following constants, depending on version of the Mac OS.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacRomanLatin1StandardVariant`
Permuted MacRoman, Euro sign variant.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacRomanLatin1TurkishVariant`
Permuted MacTurkish.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacRomanLatin1CroatianVariant`
Permuted MacCroatian, Euro sign variant.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacRomanLatin1IcelandicVariant`
Permuted MacIcelandic, standard Euro sign variant.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacRomanLatin1RomanianVariant`
Permuted MacRomanian, Euro sign variant.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for MacVT100

Specify variants of MacVT100.

```
enum {
    kMacVT100DefaultVariant = 0,
    kMacVT100CurrencySignVariant = 1,
    kMacVT100EuroSignVariant = 2
};
```

Constants

`kMacVT100DefaultVariant`

This is a meta value that maps to one of the following constants, depending on version of the Mac OS.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacVT100CurrencySignVariant`

In Mac OS versions earlier than 8.5, 0xDB is the currency sign.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kMacVT100EuroSignVariant`

In Mac OS version 8.5 and later, 0xDB is the Euro sign.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Encoding Variants for Unicode

Specify variants of Unicode.

```
enum {
    kUnicodeNoSubset = 0,
    kUnicodeCanonicalDecompVariant = 2,
    kUnicodeCanonicalCompVariant = 3,
    kUnicodeHFSPlusDecompVariant = 8,
    kUnicodeHFSPlusCompVariant = 9
};
```

Constants

`kUnicodeNoSubset`

The standard Unicode encoded character set in which the full set of Unicode characters are supported.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeCanonicalDecompVariant`

A variant of Unicode using maximal decomposition with characters in canonical order. This variant does not include most characters which have a canonical decomposition, such as single characters for accented Latin letters or single characters for Korean Hangul syllables (however, this restriction is relaxed for symbol characters in the range U+2000 to U+2FFF). In TEC Manager 1.3, the Unicode Converter supports this variant for converting to and from Mac OS encodings.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeCanonicalCompVariant`
This is the normal canonical composition according to Unicode 3.2 rules.

Available in Mac OS X v10.2 and later.

Declared in `TextCommon.h`.

`kUnicodeHFSPlusDecompVariant`
Specifies canonical decomposition according to Unicode 3.2 rules, with HFS+ exclusions ("HFS+ decomposition 3.2"). That is, it doesn't decompose in 2000-2FFF, F900-FAFF, 2F800-2FAFF. You can use ths option when converting HFS file names.

Available in Mac OS X v10.2 and later.

Declared in `TextCommon.h`.

`kUnicodeHFSPlusCompVariant`
Specifies canonical composition according to Unicode 3.2 rules, but using the HFS+ decomposition exclusions. You can use ths option when converting HFS file names. You should use this form when you want to obtain a composed form that can be converted to and from the decomposed form specified by `kUnicodeHFSPlusDecompVariant`. This is the recommended way to request decompositions with HFS+ exclusions, instead of using `mappingVersion = kUnicodeUseHFSPlusMapping`.

Available in Mac OS X v10.2 and later.

Declared in `TextCommon.h`.

EUC Text Encodings

Specify Extendec Unix Code text encodings.

```
enum {
    kTextEncodingEUC_JP = 0x0920,
    kTextEncodingEUC_CN = 0x0930,
    kTextEncodingEUC_TW = 0x0931,
    kTextEncodingEUC_KR = 0x0940
};
```

Constants

`kTextEncodingEUC_JP`
ISO 646,1-byte katakana,JIS 208 ,JIS 212.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingEUC_CN`
ISO 646, GB 2312-80.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingEUC_TW`
ISO 646, CNS 11643-1992 Planes 1-16.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingEUC_KR`
ISO 646, KS C 5601-1987.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

HFS Text Encoding

Specifies a Mac OS HFS text encoding.

```
enum {
    kTextEncodingMacHFS = 0xFF
};
```

Constants

`kTextEncodingMacHFS`
This is a metavalue for a special Mac OS encoding.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

ISO 2022 Text Encodings

Specify text encodings for ISO 2002.

```
enum {
    kTextEncodingISO_2022_JP = 0x0820,
    kTextEncodingISO_2022_JP_2 = 0x0821,
    kTextEncodingISO_2022_JP_1 = 0x0822,
    kTextEncodingISO_2022_JP_3 = 0x0823,
    kTextEncodingISO_2022_CN = 0x0830,
    kTextEncodingISO_2022_CN_EXT = 0x0831,
    kTextEncodingISO_2022_KR = 0x0840
};
```

Constants

`kTextEncodingISO_2022_JP`
See RFC 1468.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISO_2022_JP_2`
See RFC 1554.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISO_2022_JP_1`
See RFC 2237.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISO_2022_JP_3`
JIS X0213

Available in Mac OS X v10.1 and later.

Declared in `TextCommon.h`.

`kTextEncodingISO_2022_CN`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISO_2022_CN_EXT`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISO_2022_KR`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

ISO 8-bit and 7-bit Text Encodings

Specify text encodings for ISO 8-bit and 7-bit.

```
enum {
    kTextEncodingISOLatin1 = 0x0201,
    kTextEncodingISOLatin2 = 0x0202,
    kTextEncodingISOLatin3 = 0x0203,
    kTextEncodingISOLatin4 = 0x0204,
    kTextEncodingISOLatinCyrillic = 0x0205,
    kTextEncodingISOLatinArabic = 0x0206,
    kTextEncodingISOLatinGreek = 0x0207,
    kTextEncodingISOLatinHebrew = 0x0208,
    kTextEncodingISOLatin5 = 0x0209,
    kTextEncodingISOLatin6 = 0x020A,
    kTextEncodingISOLatin7 = 0x020D,
    kTextEncodingISOLatin8 = 0x020E,
    kTextEncodingISOLatin9 = 0x020F
};
```

Constants

`kTextEncodingISOLatin1`
ISO 8859-1.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatin2`
ISO 8859-2.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatin3`
ISO 8859-3.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatin4`
ISO 8859-4.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatinCyrillic`
ISO 8859-5.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatinArabic`
ISO 8859-6; equivalent to ASMO 708 and DOS CP 708.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatinGreek`
ISO 8859-7.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatinHebrew`
ISO 8859-8.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatin5`
ISO 8859-9.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatin6`
ISO 8859-10.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatin7`
ISO 8859-13; Baltic Rim

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatin8`
ISO 8859-14; Celtic

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISOLatin9`
ISO 8859-15, 8859-1; changed for Euro & CP1252 letters

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Mac Unicode Text Encoding

Specifies a script code that should be handled as a special Mac OS script code.

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

Constants

`kTextEncodingMacUnicode`

Beginning with Mac OS 8.5, the set of Mac OS script codes has been extended for some Mac OS components to include Unicode. Some of these components have only 7 bits available for script code, so `kTextEncodingUnicodeDefault` cannot be used to indicate Unicode. Instead, `kTextEncodingMacUnicode` is used as a meta-value to indicate that Unicode handles the script code a special Mac OS script code. The Text Encoding Converter handles this value similar to the way it handles the constant `kTextEncodingUnicodeDefault`.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Miscellaneous Text Encoding Standards

Specify miscellaneous text encodings.

```
enum {
    kTextEncodingShiftJIS = 0x0A01,
    kTextEncodingKOI8_R = 0x0A02,
    kTextEncodingBig5 = 0x0A03,
    kTextEncodingMacRomanLatin1 = 0x0A04,
    kTextEncodingHZ_GB_2312 = 0x0A05,
    kTextEncodingBig5_HKSCS_1999 = 0x0A06
};
```

Constants

`kTextEncodingShiftJIS`

Plain Shift-JIS.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingKOI8_R`

Russian Internet standard.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingBig5`

Big-5 encoding.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingMacRomanLatin1`

Mac OS Roman permuted to align with 8859-1.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingHZ_GB_2312`
See RFC 1842; for Chinese mail and news.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingBig5_HKSCS_1999`
Available in Mac OS X v10.1 and later.

Declared in `TextCommon.h`.

MS-DOS and Windows Text Encodings

Specify text encodings for MS-DOS and Windows.

```
enum {
    kTextEncodingDOSLatinUS = 0x0400,
    kTextEncodingDOSGreek = 0x0405,
    kTextEncodingDOSBalticRim = 0x0406,
    kTextEncodingDOSLatin1 = 0x0410,
    kTextEncodingDOSGreek1 = 0x0411,
    kTextEncodingDOSLatin2 = 0x0412,
    kTextEncodingDOSCyrillic = 0x0413,
    kTextEncodingDOSTurkish = 0x0414,
    kTextEncodingDOSPortuguese = 0x0415,
    kTextEncodingDOSIcelandic = 0x0416,
    kTextEncodingDOSHebrew = 0x0417,
    kTextEncodingDOSCanadianFrench = 0x0418,
    kTextEncodingDOSArabic = 0x0419,
    kTextEncodingDOSNordic = 0x041A,
    kTextEncodingDOSRussian = 0x041B,
    kTextEncodingDOSGreek2 = 0x041C,
    kTextEncodingDOSThai = 0x041D,
    kTextEncodingDOSJapanese = 0x0420,
    kTextEncodingDOSChineseSimplif = 0x0421,
    kTextEncodingDOSKorean = 0x0422,
    kTextEncodingDOSChineseTrad = 0x0423,
    kTextEncodingWindowsLatin1 = 0x0500,
    kTextEncodingWindowsANSI = 0x0500,
    kTextEncodingWindowsLatin2 = 0x0501,
    kTextEncodingWindowsCyrillic = 0x0502,
    kTextEncodingWindowsGreek = 0x0503,
    kTextEncodingWindowsLatin5 = 0x0504,
    kTextEncodingWindowsHebrew = 0x0505,
    kTextEncodingWindowsArabic = 0x0506,
    kTextEncodingWindowsBalticRim = 0x0507,
    kTextEncodingWindowsVietnamese = 0x0508,
    kTextEncodingWindowsKoreanJohab = 0x0510
};
```

Constants

`kTextEncodingDOSLatinUS`
Code page 437.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingDOSGreek`
Code page 737, formerly 437G.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSBalticRim`
Code page 775.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSLatin1`
Code page 860. “multilingual.”
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSGreek1`
Code page 851.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSLatin2`
Code page 852, Slavic.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSCyrillic`
Code page 855, IBM Cyrillic.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSTurkish`
Code page 857, IBM Turkish.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSPortuguese`
Code page 860.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSIcelandic`
Code page 861.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSHebrew`
Code page 862.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSCanadianFrench`
Code page 863.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSArabic`
Code page 864.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSNordic`
Cde page 865.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSRussian`
Code page 866.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSGreek2`
Code page 869, IBM Modern Green.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSThai`
Code page 874, also for Windows.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSJapanese`
Code page 932, also for Windows
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSChineseSimplif`
Code page 936, also for Windows.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSKorean`
Code page 949, also for Windows; unified Hangul.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingDOSChineseTrad`
Code page 950, also for Windows.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsLatin1`
Code page 1252.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsANSI`
Code page 1252 (alternate name).
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsLatin2`
Code page 1250, Central Europe.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsCyrillic`
Code page 1251, Slavic Cyrillic.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsGreek`
Code page 1253.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsLatin5`
Code page 1254, Turkish.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsHebrew`
Code page 1255.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsArabic`
Code page 1256.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsBalticRim`
Code page 1257.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsVietnamese`
Code page 1258.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kTextEncodingWindowsKoreanJohab`
Code page 1361, for Window NT.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

National Standard Text Encodings

Specify text encodings for various national standards.

```
enum {
    kTextEncodingUS_ASCII = 0x0600,
    kTextEncodingJIS_X0201_76 = 0x0620,
    kTextEncodingJIS_X0208_83 = 0x0621,
    kTextEncodingJIS_X0208_90 = 0x0622,
    kTextEncodingJIS_X0212_90 = 0x0623,
    kTextEncodingJIS_C6226_78 = 0x0624,
    kTextEncodingShiftJIS_X0213_00 = 0x0628,
    kTextEncodingGB_2312_80 = 0x0630,
    kTextEncodingGBK_95 = 0x0631,
    kTextEncodingGB_18030_2000 = 0x0632,
    kTextEncodingKSC_5601_87 = 0x0640,
    kTextEncodingKSC_5601_92_Johab = 0x0641,
    kTextEncodingCNS_11643_92_P1 = 0x0651,
    kTextEncodingCNS_11643_92_P2 = 0x0652,
    kTextEncodingCNS_11643_92_P3 = 0x0653
};
```

Constants

`kTextEncodingUS_ASCII`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingJIS_X0201_76`
JIS Roman and 1-byte katakana (halfwidth).

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingJIS_X0208_83`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingJIS_X0208_90`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingJIS_X0212_90`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingJIS_C6226_78`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingShiftJIS_X0213_00`
Shift-JIS format encoding of JIS X0213 planes 1 and 2

Available in Mac OS X v10.1 and later.

Declared in `TextCommon.h`.

`kTextEncodingGB_2312_80`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingGBK_95`
Annex to GB13000-93, for Windows 95; EUC-CN extended.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingGB_18030_2000`
Available in Mac OS X v10.1 and later.

Declared in `TextCommon.h`.

`kTextEncodingKSC_5601_87`
This is the same as KSC 5601-92 without Johab annex.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingKSC_5601_92_Johab`
KSC 5601-92 Johab annex.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingCNS_11643_92_P1`
CNS 11643-1992 plane 1.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingCNS_11643_92_P2`
CNS 11643-1992 plane 2.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingCNS_11643_92_P3`
CNS 11643-1992 plane 3 (11643-1986 plane 14).

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

NextStep Platform Encodings

Specify text encodings for the NextStep platform.

```
enum {
    kTextEncodingNextStepLatin = 0x0B01,
    kTextEncodingNextStepJapanese = 0x0B02
};
```

Special Text Encoding Values

Specify special cases of text encodings.

```
enum {
    kTextEncodingMultiRun = 0x0FFF,
    kTextEncodingUnknown = 0xFFFF
};
```

Constants

`kTextEncodingMultiRun`

This is a special value for multiple encoded text, external run information.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingUnknown`

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Text Encoding Formats

Specify a text encoding format.

```
typedef UInt32 TextEncodingFormat;
enum {
    kTextEncodingDefaultFormat = 0,
    kUnicode16BitFormat = 0,
    kUnicodeUTF7Format = 1,
    kUnicodeUTF8Format = 2,
    kUnicode32BitFormat = 3
};
```

Constants

`kTextEncodingDefaultFormat`

The standard default format for any base encoding.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicode16BitFormat`

The 16-bit character encoding format specified by the Unicode standard, equivalent to the UCS-2 format for ISO 10646. This includes support for the UTF-16 method of including non-BMP characters in a stream of 16-bit values.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeUTF7Format`
: The Unicode transformation format in which characters encodings are represented by a sequence of 7-bit values. This format cannot be handled by the Unicode Converter, only by the Text Encoding Converter.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeUTF8Format`
: The Unicode transformation format in which characters are represented by a sequence of 8-bit values.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicode32BitFormat`
: The UCS-4 32-bit format defined for ISO 10646. This format is not currently supported.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Discussion
A text encoding format specifies a way of formatting or algorithmically transforming a particular base encoding. For example, the UTF-7 format is the Unicode standard formatted for transmission through channels that can handle only 7-bit values. Other text encoding formats for Unicode include UTF-8 and 16-bit or 32-bit formats. These transformations are not viewed as different base encodings. Rather, they are different formats for representing the same base encoding.

Similar to text encoding variant values, text encoding format values are specific to a particular text encoding base value or to a small set of text encoding base values. A text encoding format is defined by the `TextEncodingFormat` data type.

The function `GetTextEncodingFormat` (page 39) returns the text encoding format of a text encoding specification.

Text Encoding Name Selectors

Specify the part of an encoding name you want to obtain.

```
typedef UInt32 TextEncodingNameSelector;
enum {
    kTextEncodingFullName = 0,
    kTextEncodingBaseName = 1,
    kTextEncodingVariantName = 2,
    kTextEncodingFormatName = 3
};
```

Constants
`kTextEncodingFullName`
: Requests the full name of the text encoding.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingBaseName`
: Requests the name of the base encoding.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingVariantName`
Requests the name of the encoding variant, if available.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingFormatName`
Requests the name of the encoding format, if available.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Discussion
You use a selector for the `GetTextEncodingName` function to indicate which part of an encoding name you want to determine. The text encoding name selector is defined by the `TextEncodingNameSelector` data type.

Text Encoding Variants

Specify minor variants of a base encoding or group of base encodings.

```
enum {
    kTextEncodingDefaultVariant = 0
};
```

Constants
`kTextEncodingDefaultVariant`
The standard default variant for any base encoding.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Discussion
This enumeration defines constants for the default variant of any base text encoding and for variants of the Mac OS Japanese, Mac OS Arabic, Mac OS Farsi, Mac OS Hebrew, and Unicode base encodings.

A text encoding variant specifies one among possibly several minor variants of a particular base encoding or group of base encodings. Text encoding variants are often used to support special cases such as the following:

- Differences among fonts that are all intended to support the same encoding. For example, different fonts associated with the MacJapanese and MacArabic encodings support slightly different encoding variants. These fonts would typically coexist on the same system without the user being aware of any differences.
- Artificial variants created by excluding some of the characters in an encoding. For example, the MacJapanese encoding includes separately-encoded vertical forms for some characters. In some contexts (such as with QuickDraw GX), it may be desirable to exclude these.
- Different mappings of a particular character or group of characters for different usages.

For a given text encoding base or small set of related text encoding base values, there may be an enumeration of `TextEncodingVariant` values, which always begins with 0, the default variant. In addition, for a possibly larger set of related text encoding base values, there may be bit masks that can be used independently to designate additional artificial variants. For example, there is an enumeration of six variants for the Mac OS

Japanese encoding. In addition, there are bit masks that can also be used as part of the variant for any Japanese encoding to exclude 1-byte kana or to control the mapping of the reverse solidus (backslash) character.

Languages that are dissimilar but use similar character sets are generally not designated as variants of the same base encoding (for example, MacIcelandic and MacTurkish both use a slight modification of the MacRoman character set, but they are considered separate base encodings).

When you create a new text encoding, you can specify an explicit variant of a base encoding or you can specify the default variant of that base. A text encoding variant is defined by the `TextEncodingVariant` data type. The function `GetTextEncodingVariant` (page 41) returns the text encoding variant of a text encoding specification.

Unicode and ISO UCS Text Encodings

Specify Unicode and IOS UCS text encodings.

```
enum {
    kTextEncodingUnicodeDefault = 0x0100,
    kTextEncodingUnicodeV1_1 = 0x0101,
    kTextEncodingISO10646_1993 = 0x0101,
    kTextEncodingUnicodeV2_0 = 0x0103,
    kTextEncodingUnicodeV2_1 = 0x0103,
    kTextEncodingUnicodeV3_0 = 0x0104,
    kTextEncodingUnicodeV3_1 = 0x0105,
    kTextEncodingUnicodeV3_2 = 0x0106
};
```

Constants

`kTextEncodingUnicodeDefault`

This is a meta value that takes on one of the following values, depending on the system.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingUnicodeV1_1`

This is a Unicode encoding.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingISO10646_1993`

This ISO UCS encoding has code points identical to Unicode 1.1.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingUnicodeV2_0`

This is the new location for Korean Hangul.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingUnicodeV2_1`

For the Text Encoding Converter, Unicode 2.0 is equivalent to Unicode 2.1.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingUnicodeV3_0`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextEncodingUnicodeV3_1`
Adds characters requiring surrogate pairs in UTF-16

Available in Mac OS X v10.1 and later.

Declared in `TextCommon.h`.

`kTextEncodingUnicodeV3_2`
Available in Mac OS X v10.1 and later.

Declared in `TextCommon.h`.

Unsupported Unicode Variants

Represent Unicode variants that are not yet supported or fully defined.

```
enum {
    kUnicodeNoCompatibilityVariant = 1,
    kUnicodeNoCorporateVariant = 4
};
```

Assorted Constants

Bidirectional Character Values

Specify bidirectional character properties.

```
enum {
    kUCBidiCatNotApplicable = 0,
    kUCBidiCatLeftRight = 1,
    kUCBidiCatRightLeft = 2,
    kUCBidiCatEuroNumber = 3,
    kUCBidiCatEuroNumberSeparator = 4,
    kUCBidiCatEuroNumberTerminator = 5,
    kUCBidiCatArabicNumber = 6,
    kUCBidiCatCommonNumberSeparator = 7,
    kUCBidiCatBlockSeparator = 8,
    kUCBidiCatSegmentSeparator = 9,
    kUCBidiCatWhitespace = 10,
    kUCBidiCatOtherNeutral = 11,
    kUCBidiCatRightLeftArabic = 12,
    kUCBidiCatLeftRightEmbedding = 13,
    kUCBidiCatRightLeftEmbedding = 14,
    kUCBidiCatLeftRightOverride = 15,
    kUCBidiCatRightLeftOverride = 16,
    kUCBidiCatPopDirectionalFormat = 17,
    kUCBidiCatNonSpacingMark = 18,
    kUCBidiCatBoundaryNeutral = 19
};
```

Constants

`kUCBidiCatNotApplicable`
Unassigned.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatLeftRight`
Strong types: L left-to-right.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatRightLeft`
Strong types: R right-to-left.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatEuroNumber`
Weak types: EN European number.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatEuroNumberSeparator`
Weak types: ES European number separator.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatEuroNumberTerminator`
Weak types: ET European number terminator.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatArabicNumber`
Weak types: AN Arabic number.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatCommonNumberSeparator`
Weak types: CS common number separator.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatBlockSeparator`
Separators: B paragraph separator (was block separator).

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatSegmentSeparator`
Separators: S segment separator.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatWhitespace`
Neutrals: WS whitespace.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatOtherNeutral`
Neutrals: ON other neutrals (unassigned codes could use this).

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatRightLeftArabic`
Unicode 3.0; AL right-to-left Arabic (was Arabic letter).

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatLeftRightEmbedding`
Unicode 3.0; LRE eft-to-right embedding.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatRightLeftEmbedding`
Unicode 3.0; RLE right-to-left embedding.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatLeftRightOverride`
Unicode 3.0; LRO left-to-right override.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatRightLeftOverride`
Unicode 3.0; RLO right-to-left override.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatPopDirectionalFormat`
Unicode 3.0; PDF pop directional Format.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatNonSpacingMark`
Unicode 3.0; NSM non-spacing mark.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCBidiCatBoundaryNeutral`
Unicode 3.0; BN boundary neutral.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Discussion
These values are requested by `kUCCharPropTypeBidiCategory`.

Common and Special Unicode Values

Specify sommon and special Unicode code values.

```
enum {
    kUnicodeByteOrderMark = 0xFEFF,
    kUnicodeObjectReplacement = 0xFFFC,
    kUnicodeReplacementChar = 0xFFFD,
    kUnicodeSwappedByteOrderMark = 0xFFFE,
    kUnicodeNotAChar = 0xFFFF
};
```

Constants
`kUnicodeByteOrderMark`
Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeObjectReplacement`
A placeholder for a non-text object.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeReplacementChar`
Unicode replacement for an input character that cannot be converted.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeSwappedByteOrderMark`
Not a Unicode character; byte-swapped version of FEFF.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUnicodeNotAChar`
Not a Unicode character; may be used as a terminator.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

TEC Plugin Dispatch Table Versions

Specify a version for a TEC plug-in dispatch table.

```
enum {
    kTECPluginDispatchTableVersion1 = 0x00010000,
    kTECPluginDispatchTableVersion1_1 = 0x00010001,
    kTECPluginDispatchTableVersion1_2 = 0x00010002,
    kTECPluginDispatchTableCurrentVersion = kTECPluginDispatchTableVersion1_2
};
```

Constants

`kTECPluginDispatchTableVersion1`
Specifies versions 1.0 through 1.0.3.

Available in Mac OS X v10.0 and later.

Declared in `TextEncodingPlugin.h`.

`kTECPluginDispatchTableVersion1_1`
Specifies version 1.1.

Available in Mac OS X v10.0 and later.

Declared in `TextEncodingPlugin.h`.

`kTECPluginDispatchTableVersion1_2`
Specifies version 1.2.

Available in Mac OS X v10.0 and later.

Declared in `TextEncodingPlugin.h`.

`kTECPluginDispatchTableCurrentVersion`
A meta value that specifies the current version.

Available in Mac OS X v10.0 and later.

Declared in `TextEncodingPlugin.h`.

TEC Plug-in Signatures

Specify a TEC plug-in signature.

```
enum {
    kTECSignature = 'encv',
    kTECUnicodePluginSignature = 'puni',
    kTECJapanesePluginSignature = 'pjpn',
    kTECChinesePluginSignature = 'pzho',
    kTECKoreanPluginSignature = 'pkor'
};
```

Unicode Character Property Types

Specify property types for a Unicode charater.

```
typedef SInt32 UCCharPropertyType;
enum {
    kUCCharPropTypeGenlCategory = 1,
    kUCCharPropTypeCombiningClass = 2,
    kUCCharPropTypeBidiCategory = 3
};
```

Constants

`kUCCharPropTypeGenlCategory`

Requests enumeration value.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCCharPropTypeCombiningClass`

Requests numeric value 0 to 255.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCCharPropTypeBidiCategory`

Requests enumeration value.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Unicode Character Property Values

Specify a propery value for a Unicode character.

```
typedef UInt32 UCCharPropertyValue;
enum {
    kUCGenlCatOtherNotAssigned = 0,
    kUCGenlCatOtherControl = 1,
    kUCGenlCatOtherFormat = 2,
    kUCGenlCatOtherSurrogate = 3,
    kUCGenlCatOtherPrivateUse = 4,
    kUCGenlCatMarkNonSpacing = 5,
    kUCGenlCatMarkSpacingCombining = 6,
    kUCGenlCatMarkEnclosing = 7,
    kUCGenlCatNumberDecimalDigit = 8,
    kUCGenlCatNumberLetter = 9,
    kUCGenlCatNumberOther = 10,
    kUCGenlCatSeparatorSpace = 11,
    kUCGenlCatSeparatorLine = 12,
    kUCGenlCatSeparatorParagraph = 13,
    kUCGenlCatLetterUppercase = 14,
    kUCGenlCatLetterLowercase = 15,
    kUCGenlCatLetterTitlecase = 16,
    kUCGenlCatLetterModifier = 17,
    kUCGenlCatLetterOther = 18,
    kUCGenlCatPunctConnector = 20,
    kUCGenlCatPunctDash = 21,
    kUCGenlCatPunctOpen = 22,
    kUCGenlCatPunctClose = 23,
    kUCGenlCatPunctInitialQuote = 24,
    kUCGenlCatPunctFinalQuote = 25,
    kUCGenlCatPunctOther = 26,
    kUCGenlCatSymbolMath = 28,
    kUCGenlCatSymbolCurrency = 29,
    kUCGenlCatSymbolModifier = 30,
    kUCGenlCatSymbolOther = 31
};
```

Constants

`kUCGenlCatOtherNotAssigned`
Cn other; not assigned.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatOtherControl`
Cc other; control.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatOtherFormat`
Cf other; format.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatOtherSurrogate`
Cs other; surrogate.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatOtherPrivateUse`
Co other; private use.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kUCGenlCatMarkNonSpacing`
Mn mark; non-spacing.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kUCGenlCatMarkSpacingCombining`
Mc mark; spacing combining.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kUCGenlCatMarkEnclosing`
Me mark; enclosing.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kUCGenlCatNumberDecimalDigit`
Nd number; decimal digit.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kUCGenlCatNumberLetter`
Nl number; letter.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kUCGenlCatNumberOther`
No number; other.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kUCGenlCatSeparatorSpace`
Zs separator; space.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kUCGenlCatSeparatorLine`
Zl separator; Line.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kUCGenlCatSeparatorParagraph`
Zp separator; paragraph.
Available in Mac OS X v10.0 and later.
Declared in `TextCommon.h`.

`kUCGenlCatLetterUppercase`
Lu Letter; uppercase.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatLetterLowercase`
Ll Letter; lowercase.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatLetterTitlecase`
Lt Letter; titlecase.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatLetterModifier`
Lm Letter; modifier.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatLetterOther`
Lo Letter; other.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatPunctConnector`
Pc punctuation; connector.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatPunctDash`
Pd punctuation; dash.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatPunctOpen`
Ps punctuation; open.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatPunctClose`
Pe punctuation; close.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatPunctInitialQuote`
Pi punctuation; initial quote.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatPunctFinalQuote`
Pf punctuation; final quote.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatPunctOther`
Po punctuation; other.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatSymbolMath`
Sm symbol; math.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatSymbolCurrency`
Sc symbol; currency.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatSymbolModifier`
Sk symbol; modifier.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kUCGenlCatSymbolOther`
So symbol; other.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Unicode Mapping Versions

Specify a Unicode mapping version.

```
typedef SInt32 UnicodeMapVersion;
enum {
    kUnicodeUseLatestMapping = -1,
    kUnicodeUseHFSPlusMapping = 4
};
```

Constants

`kUnicodeUseLatestMapping`
Instead of explicitly specifying the mapping version of the Unicode mapping table to be used for conversion of a text string, you can use this constant to specify that the latest version be used.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

`kUnicodeUseHFSPlusMapping`
Indicates the mapping version used by HFS Plus to convert filenames between Mac OS encodings and Unicode. Only one constant is defined so far for a specific mapping version.

Available in Mac OS X v10.0 and later.

Declared in `UnicodeConverter.h`.

Discussion
When performing conversions, you specify the version of the Unicode mapping table to be used for the conversion. You provide the version number in the mapping version field of the structure `UnicodeMapping` (page 107) that is passed to a function. A Unicode mapping version is defined by the `UnicodeMapVersion` data type.

Unwanted Data Constants

Specify data you don't care about receiving.

```
enum {
    kTextScriptDontCare = -128,
    kTextLanguageDontCare = -128,
    kTextRegionDontCare = -128
};
```

Constants
`kTextScriptDontCare`
Indicates that the code is not provided for the derivation.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextLanguageDontCare`
Indicates that language code is not provided for the derivation.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

`kTextRegionDontCare`
The region code is not provided for the derivation.

Available in Mac OS X v10.0 and later.

Declared in `TextCommon.h`.

Discussion
For backward compatibility with earlier releases of the Mac OS, the Text Encoding Conversion Manager provides the functions `UpgradeScriptInfoToTextEncoding` (page 77) and `RevertTextEncodingToScriptInfo` (page 47) that you can use to derive Script Manager values from a text encoding or vice versa.

When using these functions, you can specify a Script Manager language code, script code, and/or font values to derive a text encoding. These three constants are defined to allow you to identify any part of the derivation you don't care about. When reverting from a text encoding to Script Manager values, the Unicode Converter returns these constants for a corresponding value it does not derive: `kTextLanguageDontCare`, `kTextScriptDontCare`, and `kTextRegionDontCare`.

Result Codes

The most common result codes returned by Text Encoding Conversion Manager are listed below.

Result Code	Value	Description
`kTextUnsupportedEncodingErr`	-8738	The encoding or mapping is not supported for this function by the current set of tables or plug-ins. Available in Mac OS X v10.0 and later.
`kTextMalformedInputErr`	-8739	The text input contains a sequence that is not legal in the specified encoding, such as a DBCS high byte followed by an invalid low byte (0x8120 in Shift-JIS). Available in Mac OS X v10.0 and later.
`kTextUndefinedElementErr`	-8740	The text input contains a code point that is undefined in the specified encoding. The function did not completely convert the input string. You can resume conversion from a point beyond the offending character, or take some other action. Available in Mac OS X v10.0 and later.
`kTECMissingTableErr`	-8745	The specified encoding is partially supported, but a specific table required for this function is missing. Available in Mac OS X v10.0 and later.
`kTECTableChecksumErr`	-8746	A specific table required for this function has a checksum error, indicating that it has become corrupted. Available in Mac OS X v10.0 and later.
`kTECTableFormatErr`	-8747	The table format is either invalid or it cannot be handled by the current version of the code. The function did not convert the string Available in Mac OS X v10.0 and later.
`kTECCorruptConverterErr`	-8748	The converter object is invalid. Returned by the Text Encoding Converter functions only. Available in Mac OS X v10.0 and later.
`kTECNoConversionPathErr`	-8749	The converter supports both the source and target encodings, but cannot convert between them either directly or indirectly. Returned by the Text Encoding Converter functions only. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kTECBufferBelowMinimumSizeErr	-8750	The output text buffer is too small to accommodate the result of processing of the first input text element. No part of the input string was processed. Available in Mac OS X v10.0 and later.
kTECArrayFullErr	-8751	The supplied TextEncodingRun, ScriptCodeRun, or UnicodeMapping array is too small, and the input was not completely converted. Call the function again with another output buffer—or with the same output buffer after copying its contents—to convert the remainder of the string Available in Mac OS X v10.0 and later.
kTECPartialCharErr	-8753	The input text ends in the middle of a multibyte character and conversion stopped. Append the unconverted input from this call to the beginning of the subsequent input text and call the function again. Available in Mac OS X v10.0 and later.
kTECUnmappableElementErr	-8754	An input text element cannot be mapped to the specified output encoding(s) using the specified options. For the Unicode Converter, this error can occur only if kUnicodeUseFallbacksBit control flag is not set. Available in Mac OS X v10.0 and later.
kTECIncompleteElementErr	-8755	The input text ends with a text element that might be incomplete, or contains a text element that is too long for the internal buffers. Available in Mac OS X v10.0 and later.
kTECDirectionErr	-8756	An error, such as a direction stack overflow, occurred in directionality processing. Available in Mac OS X v10.0 and later.
kTECGlobalsUnavailableErr	-8770	Global variables have already been deallocated, premature termination. The function did not convert the string. Available in Mac OS X v10.0 and later.
kTECItemUnavailableErr	-8771	An item (for example, a name) is not available for the specified region (and encoding, if relevant). Available in Mac OS X v10.0 and later.
kTECUsedFallbacksStatus	-8783	The function has completely converted the input string to the specified target using one or more fallbacks. For the Unicode Converter, this status code can only occur if the kUnicodeUseFallbacksBit control flag is set. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kTECNeedFlushStatus	-8784	The application disposed of a converter object by calling TECDisposeConverter, but there is still text contained in internal buffers. Returned by the Text Encoding Converter functions only. Available in Mac OS X v10.0 and later.
kTECOutputBufferFullStatus	-8785	The converter successfully converted part of the input text, but the output buffer was not large enough to accommodate the entire input text after conversion. Convert the remaining text beginning from the position where the conversion stopped. Available in Mac OS X v10.0 and later.

Updated Unicode Decompositions

The Text Encoding Converter now uses the decompositions defined in Unicode 3.2. The changes are limited to characters in Greek, Thai, Gurmukhi, and Arabic/Farsi. This change affects conversion of characters between Unicode and the Mac encodings for these scripts.

1. MacThai

   xD3 = u0E33 for composed Unicode; now maps to u0E33 for decomposed Unicode too, instead of to uF860+u0E4D+u0E32 (old mapping is loosely mapped to xD3)

2. MacGurmukhi

   x91=u0A5C for composed Unicode; now maps to u0A5C for decomposed Unicode too, instead of to uF860+u0A21+u0A3C (old mapping is loosely mapped to x91)

   xD5 is now always (composed & decomposed) mapped to xF860+u0A38+u0A3C instead of u0A36, since the latter is in CompostionExclusions.txt (the old mapping is loosely mapped back)

3. MacGreek

   For mapping to decomposed Unicode - all of the decompositions that formerly used u030D now use u0301; the affected characters (and their mappings for composed Unicode) are:

   x87=u0385, xC0=u03AC, xCD=u0386, xCE=u0388, xD7=u0389, xD8=u038A, xD9=u038C, xDA=u038E, xDB=u03AD, xDC=u03AE, xDD=u03AF, xDE=u03CC, xDF= u038F, xE0=u03CD, xF1=u03CE, xFD=u0390, xFE=u03B0 (the old mappings are loosely mapped back)

4. MacArabic (all variants), MacFarsi (both variants)

   Table A-1shows the mapping from composed to decomposed Unicode. The items in the table were not previously decomposed.

   Table A-1 MacArabic/MacFarsi mapping from composed to decomposed

Char	Composed	Decomposed
xC2	u0622	u0627+u0653
xC3	u0623	u0627+u0654
xC4	u0624	u0648+u0654
xC5	u0625	u0627+u0655
xC6	u0626	u064A+u0654

These encodings are now supported by the Text Encoding Converter:

- GB 18030

  Full support for the new Chinese standard has been added to the TEC, along with new fonts in the system to support the new characters.

- DOS encodings for Simba

  ```
  kTextEncodingDOSGreek
  kTextEncodingDOSBalticRim
  kTextEncodingDOSLatin2
  kTextEncodingDOSTurkish
  kTextEncodingDOSIcelandic
  kTextEncodingDOSRussian
  ```

Document Revision History

This table describes the changes to *Text Encoding Conversion Manager Reference*.

Date	Notes
2005-07-07	Corrected typographical errors.
2003-05-01	Modified the description of the `UniCharArrayOffset` data type.
2003-01-20	Fixed formatting.
	Added information to the callbacks needed to create a TEC plug-in.
	Added information to the Discussion of the `UnicodeMapping` structure that describes how to set up normalization options.
	Added the section "Updated Unicode Decompositions" (page 169).

Document Revision History

Index

H

I

K

M

N

Q

R

S

T

U