Supporting Unicode Input

Carbon > Text & Fonts

2005-07-07

Contents

Tables and Listings

Introduction to Supporting Unicode Input

This document describes how applications and input methods can use the Text Services Manager and Unicode Utilities programming interfaces to support Unicode input in Mac OS X.

The Text Services Manager is the part of the Mac OS that provides an environment for applications to use text services such as input methods. The Text Services Manager handles communication between client applications that request text services and the software modules, known as text service components, that provide them. The Text Services Manager presents two separate programming interfaces to the features it provides: one for applications and another for text service components.

Unicode Utilities allow applications and text service components (such as input methods) to perform various operations on Unicode text. You can use Unicode Utilities to control Unicode-related text behavior, such as the specification of Unicode keyboard layout.

In addition to Unicode input, most complete Unicode applications need various other services which are not be covered in this document, such as imaging Unicode and processing text. You can use Apple Type Services for Unicode Imaging (ATSUI) and Multilingual Text Engine (MLTE) for these services. In addition, Unicode input using the methods described in this document is compatible with any other service for imaging and processing Unicode text.

See Also

For information on ATSUI see *ATSUI Programming Guide*.

For information on MLTE see *Handling Unicode Text Editing With MLTE*.

International Text in Mac OS X

This section contains an overview of international text handling on the Mac OS and a more specific introduction to some of the Unicode facilities available with Mac OS X. If you would like more information on converting between text encodings, see *Programming With the Text Encoding Conversion Manager*.

Languages, Writing Systems, Scripts, and Orthographies

Written representation of a spoken language relies on a writing system. A **writing system** is an artificial construct used to record language in written form. It can be viewed as having three main components—language, scripts, and orthography—with well-defined relations to one another.

A **script** comprises a set of symbols that represent the components of a language. A writing system uses one or more scripts for the symbols required to represent linguistic elements, which include sound, meaning, syntax and so forth. A script can be coupled with one language, or it can represent and be used by many languages. Moreover, a language can have more than one script associated with it. For example, the Japanese language uses the Japanese script, while the French, Italian, and Spanish languages all use parts of the Latin script.

A script exists apart from both the languages it represents and the writing systems for which it is used. (A small number of scripts, less than 100, are used by writing systems despite the large number of existing modern and archaic languages.) A special category of scripts, called pseudoscripts, exists for use with other scripts. These pseudoscripts include symbols, numbers, and punctuation.

Writing systems can use different scripts at the same time. A writing system uses at least one script and typically one or more pseudoscripts. In this sense, it is best to refer to the characters a writing system includes as a repertoire of characters, rather than a character set, because these characters can belong to different scripts.

The writing system for a language entails an **orthography** which defines the relationship between the written language and one or more scripts. Among the rules an orthography specifies are rules of directionality, level of discreteness, and units of representation. For example, for mixed-directional text, the direction of a paragraph is important. For writing systems based in European languages, a paragraph is considered a unit of representation, as is a word. Word division and paragraph identification are easily determined for these languages, but this is not necessarily the case for other writing systems, such as those based in Japanese or Indic languages.

Script Systems and Script Codes

Traditionally, in the Mac OS, a **script system** has been understood to be a collection of software facilities that provides for the representation of a specific writing system. This usage of the term “script” in the phrase “script system” should not be confused with the more current, linguistics-derived notion of scripts that is used in the Mac OS and described in “Languages, Writing Systems, Scripts, and Orthographies” (page 9).

Types of Mac OS script systems include the following:

- single-byte simple: small character set, non-contextual, not bidirectional (example: English)
- single-byte complex: small character set, but with contextual or bidirectional text (example: Devanagari)
- double-byte: large character set (examples: Japanese, Korean, Chinese, and Simplified Chinese)

At minimum, a script system consists of the following items:

- keyboard resources, which provide for text input in any language from any keyboard; these allow for convenient switching from one input language to another on a single keyboard
- international resources, which contain information specific to a particular language, such as its date and time formats, sorting order, and word-break rules
- fonts, that is, sets of glyphs that are associated with specified characters

A **script code** is a numeric value indicating a particular Mac OS script system. Constants are defined for each of the script codes recognized by the Mac OS.

Characters, Character Encodings, and Unicode

A writing system's alphabet, numbers, punctuation, and other writing marks consist of characters. A **character** is a symbolic representation of an element of a writing system; it is the concept of, for example, "lowercase a" or "number 3".

In memory, text is stored as character codes, where each code is a numeric value that defines a particular character. A **character encoding** is the organization of the set of numeric codes that represent all the meaningful characters of a script system in memory. There are two fundamental classes of Mac OS character encodings: single-byte and double-byte.

Unicode is an international standard that combines the characters for all commonly used writing systems into a single, coded character set, based upon a 16-bit character encoding standard. With a universal character encoding such as Unicode, the character sets of separate writing systems do not overlap. Furthermore, Unicode resolves the issue of conflicting character encodings within a single writing system; for example, in Unicode, there is no overlap between Roman character codes and the Symbol font's character codes.

Keyboards and Input Methods

By means of keyboard input, the user can create text that your application stores as character codes. The system reports the user's key-down, key-up, and auto-key events to your application through events. Key-down and key-up events report that the user pressed or released a key, respectively. Auto-key events report that the user has held a key down for a certain amount of time. For keyboard-related events, the application receives both the virtual key code and the character code for the key that is pressed, as well as the state of any modifier keys (Shift, Caps Lock, Command, Option, and Control) at the time of the event.

To obtain this information for your application, the Mac OS uses keyboard resources to convert key presses into the correct character codes for the current writing system, taking into account the type of keyboard being used.

Key translation is the process by which character codes are generated. Each keyboard has a particular physical arrangement of keys, and each keypress generates a value called a raw key code, which indicates which key was pressed. The keyboard driver that handles the keypress maps these raw key codes to keyboard-independent virtual key codes.

Any given script system has one or more keyboard-layout resources. The keyboard-layout resources provide script-specific maps for converting a virtual key code into the character code that is passed to your application. As part of the key-translation process, the keyboard-layout resources must take into account the current dead-key state. A **dead key** is a keypress or modifier-plus-keypress combination that produces no immediate character output, but instead affects the character(s) that are ultimately produced by the following key press(es).

A keyboard layout is what the Key Caps application shows. For the purposes of this document, a keyboard-layout resource is the critical item in determining keyboard layout; changing the keyboard layout means changing the keyboard-layout resource. Because keyboard layouts are independent of the physical keyboard attached to the computer, your application has the flexibility of changing text input from one writing system to another by simply using a different keyboard-layout resource.

For languages with large character sets, it is impractical to manufacture keyboards with keys for every possible character. In such a case, it is usually the job of an input method, working in conjunction with a keyboard, to handle text input. An **input method** is a software module, often independent of the application it serves, that performs complex processing of text input, prior to the application's processing of the text. A typical example of an input method is a translation service that converts character codes that can be entered from the keyboard into character codes that cannot; text input in Japanese, Chinese, and Korean usually requires an input method.

Unicode Script Codes

The set of Mac OS script codes that identify particular script systems includes Unicode, which is handled as a special Mac OS script code. The Text Encoding Converter and other Mac OS facilities use the constant `kTextEncodingUnicodeDefault` (0x0100) to designate Unicode. However, because some components have only 7 bits available for a script code, rather than the typical 16 bits, the value `smUnicodeScript` (0x7E) can also be used to indicate Unicode. For example, the Text Encoding Converter handles the `smUnicodeScript` value similar to `kTextEncodingUnicodeDefault`.

Unicode Keyboard-Layout Resource and the UCKeyTranslate Function

Similar to the (pre-Unicode) keyboard-layout resource (`'KCHR'`), the Unicode keyboard-layout resource (`'uchr'`) contains the data necessary to map virtual key codes to character codes for various keyboard layouts. However, the `'uchr'` resource specifies Unicode keyboard layouts—that is, keyboard layouts which produce Unicode character codes, rather than characters in a Mac OS encoding.

Because some Unicode character codes can be mapped to Mac OS encoded character codes (while some cannot), for the purposes of key translation there are considered to be two categories of Unicode keyboard-layout resources. The first category of `'uchr'` resources is one that produces Unicode character codes that are all within the range of a single Mac OS encoding. That is, these partial Unicode `'uchr'` resources contain only Unicode characters that can be mapped to characters belonging to the Mac OS encoding associated with its ID range.

The second category of `'uchr'` resources may produce any Unicode characters. That is, these full Unicode `'uchr'` resources contain Unicode characters that are either not all within the range of a single Mac OS encoding or are not within the range of any Mac OS encoding. Table 2-1 shows the relationships of keyboard-layout resources to differing types of text input.

Table 1-1 Text input types and keyboard layouts

Input Type	Keyboard Layout (resource type, ID)
Produces Mac OS encoded characters	KCHR, >= 0
Produces partial Unicode characters	uchr, >= 0
Produces full Unicode characters	uchr < 0

The function `UCKeyTranslate` uses the `'uchr'` resource to produce Unicode character codes. However, unlike its non-Unicode counterpart (the `KeyTranslate` function), `UCKeyTranslate` also does the following:

1. Outputs multiple character codes. A single keycode (or a dead-key sequence) can produce a string of up to 255 Unicode characters. This facility is useful both for some international script systems and for the production of macros. As an example of the former, the Devanagari keyboard in the Indian Language Kit must be able to produce up to three characters from a single keypress to support the keyboard standards of India.

2. Allows multiple dead keys. The keyboard standards for some countries require double dead keys. For example, Greek keyboards use two dead keys for adding diacritical marks.

3. Handles virtual key codes with a range greater than 0-127. While this requirement is currently uncommon in the Mac OS, some types of keyboards—for example, older Kanji keyboards and keyboards for some other operating systems—may use a larger key code range.

4. Allows virtual key code mapping to depend on keyboard type. While the use of virtual key codes should theoretically remove all dependencies on particular physical keyboards, in some cases key translation does depend on the keyboard type (due to certain scripts, languages, and regions needing subtle differences in layout for specific keyboards). The `UCKeyTranslate` function accommodates this need by requesting keyboard type information and using the `'uchr'` resource to access the proper keyboard's mapping tables in cases where there is a keyboard-specific dependency, thus eliminating the need to use the `'itlk'` resource.

Unicode in the Keyboard Menu

The Keyboard menu in Mac OS X appears on the menu bar when more than one script system is enabled. It permits the user to choose among keyboard layouts, input methods, and script systems, for text input.

If there are input methods for any of the Mac OS double-byte script systems that are enabled, the Keyboard menu shows only the input methods; otherwise, in the absence of input methods, it shows the keyboard layouts. For all other enabled script systems, including Unicode, the keyboard menu shows keyboard layouts and input methods.

Note: The Keyboard menu shows each keyboard layout as a single entry, regardless of whether it is specified by a `'KCHR'`, a `'uchr'`, or both.

To display a full Unicode script system in the Keyboard menu, the System must include an international bundle resource (`'itlb'`) with a resource ID of `smUnicodeScript` (0x7E) and one or more full Unicode keyboard layouts or input methods.

Full Unicode keyboard layouts and input methods (that is, for input sources that produce Unicode characters that are not within the range of a single Mac encoding), if enabled, are shown in their own section of the menu, after all of those for Mac OS script systems.

Supporting Unicode Input in Applications and Input Methods

This chapter describes how applications and input methods can support Unicode input by using Text Services Manager and Unicode Utilities.

- Application developers should read "Supporting Unicode Input in Applications" (page 15) to learn about the steps required for an application to support Unicode input.
- Input method developers should read "Providing Unicode Support in Input Methods" (page 20) to learn about the steps required for an input method to support Unicode input.
- Typically the Text Services Manager calls the `UCKeyTranslate` function when needed. However, there are occasions when your application or input method may need to call this function directly. See "Using the UCKeyTranslate Function" (page 23) for more details.

Supporting Unicode Input in Applications

To support Unicode input, an application must both support the Text Services Manager and request Unicode input. Applications that do not support Unicode input fall in two categories: those that do not support the Text Services Manager, and those that do, but which do not request Unicode input. In both cases, these applications do receive some of the benefit of text input from Unicode input sources which can take the form of either Unicode keyboard layouts (specified by `'uchr'` resources) or Unicode input methods and text services.

However, the kinds of Unicode input available to applications that do not support Unicode input are restricted. These applications receive only input from partial Unicode input sources, that is sources that generate only Unicode characters that are all within the repertoire of a single Mac encoding, usually the Mac encoding determined by the current keyboard script. This is because text from partial Unicode input sources is automatically converted by the Text Services Manager to a Mac OS encoding for delivery to these applications. Full Unicode input sources—that is, those which either generate characters within the repertoire of several Mac encodings or outside the repertoire of any Mac encoding—are not available to these applications and appear disabled in the Keyboard menu.

You application can support the Text Services Manager in one of the two following ways:

- Implement Carbon events for text handlers. This is the preferred way because the performance advantages of using Carbon events over Apple events are significant. In additoin, you can target Carbon events at either the application or control and window level, whereas Apple events are targeted only at the application level. For information on implementing Carbon events for text handlers, see *Understanding Text Input and the Text Services Manager in Carbon*.
- Provide Apple event handlers for he full suite of Text Services Manager Apple events in order to support inline input of text. While input can be handled using the bottomline method, this mode of input does not support full Unicode input sources, but only those input sources whose output can be converted to a given Mac encoding (that is, partial Unicode input sources).

Implementing a set of Apple event handlers for the Text Services Manager suite, for the purpose of supporting inline input in general and Unicode input in particular, greatly enhances the text input experience for users of your applications in a variety of existing input sources as well as new Unicode input sources. Even if the majority of existing input methods are associated with a particular Mac script system (and therefore a particular Mac encoding), your application will automatically support these input sources because the Text Services Manager converts all text from Mac OS encoding input sources to Unicode for delivery to applications that have requested Unicode input.

See "Using Apple Events to Handle Unicode Text" (page 16) for more information.

Note: TSMTE does not currently support Unicode input. If an application does rely on TSMTE for input, its input sources will be limited to those which generate input within the repertoire of individual Mac OS encodings.

Identifying an Application as Supporting Unicode

Text Services Manager client applications must create an internal record called a TSM document (defined by the `TSMDocument` data type) before they can use any services provided through the Text Services Manager. A TSM document is a private data structure that your application associates with each of its documents that use a text service.

You use the TSM document type `kUnicodeDocument` (`'udoc'`) to request Unicode input. When a Unicode-input TSM document is active, the associated application receives input in Unicode. The application can receive input from all input types: full Unicode, partial Unicode, and Mac OS encodings.

Non-Unicode (Mac OS encoded) input is converted to Unicode before being delivered to a Unicode-input TSM document.

When non-Unicode TSM documents are active or when the current application is not a Text Services Manager client, the application receives Mac OS encoded input. In these cases, full Unicode input sources are disabled in the Keyboard menu and cannot be used, and input from partial Unicode sources is automatically converted to the current keyboard script (a Mac OS encoding) by the Text Services Manager.

Your application creates a Unicode TSM document by specifying the `kUnicodeDocument` (`'udoc'`) type in the `supportedInterfaceTypes` parameter of the function `NewTSMDocument`.

Using Apple Events to Handle Unicode Text

Text Services Manager uses a Unicode Apple event that allows applications with Unicode TSM documents to streamline their event handling.

In this Apple event model of text event handling, your application calls `WaitNextEvent` and passes low-level keyboard events to the Text Services Manager through the function `TSMEvent`. The function `TSMEvent` always returns true, to indicate that the key event was processed, either by an input method (and delivered through the standard Text Services Manager Apple events) or by means of direct delivery to the application (through the `kUnicodeNotFromInputMethod` Apple event). Because the `kUnicodeNotFromInputMethod` Apple event contains both the Unicode character code(s) and a copy of the original low-level key event record, your application can now consolidate all of its keyboard input processing in a single logical unit in its Apple event handlers, rather than in an event loop.

This section provides details on how to modify existing Text Services Manager Apple event handlers and discusses the Text Services Manager Apple event required to support Unicode input. If your application already supports the Text Services Manager, these changes are minimal. If your application does not currently support the Text Services Manager, you should first implement support for the Text Services Manager.

Modifying Existing Apple Event Handlers for Unicode

When the active TSM document is of type `kUnicodeDocument`, the Text Services Manager delivers all text content in Text Services Manager Apple events as Unicode text, in a descriptor whose keyword continues to be `keyAETheData`, but whose descriptor type is type`UnicodeText`.

When known data structures accompanying the Unicode text contain offsets to text, these offsets are also converted, if needed, to Unicode (byte) offsets to match the encoding of the text delivered to the application's Apple event handler. This delivery of text (and accompanying byte offsets) in Unicode occurs regardless of the type of input source. If the input source is a Unicode input method, text and offsets are passed through by the Text Services Manager to the application's handler unchanged, but if the input source generates text in a Mac encoding, the generated text is converted to Unicode automatically by the Text Services Manager.

Text is converted between Unicode and Mac OS encodings as necessary. Text from Unicode input sources is automatically converted to Mac encodings for delivery to applications that don't use Unicode TSM documents. Text from Mac OS encoding input sources is converted to Unicode for delivery to applications using Unicode TSM documents. Similarly, application text requested by an input method (with the Apple event ID `kGetSelectedText`) is converted as necessary.

The Update Active Input Area Event

Your application's Apple event handler for the `kUpdateActiveInputArea` Apple event must obtain the `keyAETheData` parameter using the descriptor type `typeUnicodeText` to obtain the Unicode content of the active input area. The `keyAEFixLength`, `keyAEHiliteRange`, `keyAEUpdateRange`, and `keyAEClauseOffsets` parameters all contain byte offsets into the Unicode text.

The Position To Offset Event

Your application's Apple event handler for the `kPos2Offset` Apple event must reply with the `keyAEOffset` parameter containing a Unicode text (byte) offset. If the text service requesting the offset is associated with a Mac OS encoding, the Text Service Manager converts the text offset from Unicode to that of the Mac OS encoding.

The Offset To Position Event

Your application's Apple event handler for the `kOffset2Pos` Apple event must treat the `keyAEOffset` parameter as a Unicode text (byte) offset. If the text service specifying the text offset is associated with a Mac OS encoding, the Text Services Manager converts the text offset from the Mac OS encoding to Unicode before forwarding the Apple event to the application.

The Get Selected Text Event

Your application's Apple event handler for the `kGetSelectedText` Apple event must return the current text selection as Unicode text. If the text service specifying the text offset is associated with a Mac OS encoding, the Text Services Manager will convert the Unicode text to the Mac OS encoding before forwarding the Apple event to the text service. Supporting this event is optional, but recommended.

Supporting the Unicode (Not From Input Method) Apple Event

To support Unicode input through the Text Services Manager, your application must provide a handler for the Text Services Manager Unicode Apple event whose event ID is `kUnicodeNotFromInputMethod`. When the user generates Unicode input that does not originate from an input method (that is, the Unicode text may be generated by a keyboard layout or is simply not handled by an input method) the Text Services Manager forwards the generated input to your application as Unicode text in the `kUnicodeNotFromInputMethod` Apple event.

Note: Unicode text resulting from input method interactions is delivered using the `UpdateActiveInputArea` Apple event, as is the case for non-Unicode text.

The `kUnicodeNotFromInputMethod` Apple event contains the Unicode text, a copy of the original low-level key event, and a `ScriptLanguageRecord` structure that identifies the current keyboard script. Your application's event handler for the `kUnicodeNotFromInputMethod` Apple event must obtain the `keyAETheData` parameter using the descriptor type t`ypeUnicodeText` to obtain the input as Unicode text.

Your application's Apple event handler can also obtain the original low-level key event from a parameter whose keyword is `keyAETSMEventRecord` and whose descriptor type is `typeLowLevelEventRecord`. If the current keyboard layout is determined by a `'KCHR'` resource, you can pass the virtual key code and modifiers to the function `KeyTranslate` to produce a Mac OS encoding character code. Otherwise, if a Unicode keyboard layout is being used (that is, if the keyboard layout is determined by a `'uchr'` resource), you can use the `UCKeyTranslate` function. Typically, you do not need to perform either action.

The application's Apple event handler for the `kUnicodeNotFromInputMethod` event should always fully process the input and return `noErr`. Returning any error or not providing a handler causse the `TSMEvent` function to indicate that the low-level key event was not handled, in which case your application may not be able to generate the correct text, depending on whether the input source is a Unicode keyboard layout and whether a dead-key sequence is in progress.

Note: In most cases, the Text Services Manager Apple event contains two required parameters, one of which is the `keyAEServerInstance` parameter, which identifies the component that is sending the Apple event. However, in the case of the `kUnicodeNotFromInputMethod` Apple event, this parameter is not included because the event only pertains to cases where a component (such as, an input method) is not handling the data.

Class	`kTextServiceClass`
ID	`kUnicodeNotFromInputMethod`
Requested action	Accept Unicode text.

The required parameters are as follows:

Keyword	`keyAETheData`
Descriptor type	`typeUnicodeText`
Data	Unicode text. Note that this text data has not been processed in any way by a text servcie component.

Keyword	`keyAETSMEventRecord`

Descriptor type	`typeLowLevelEventRecord`
Data	A copy of the original low-level key event record.

Keyword	`keyAETSMDocumentRefcon`
Descriptor type	`typeLongInteger`
Data	A TSMdocument specfier (reference constant0 supplied by the appcliation in a prior call to the function `NewTSMDocument`. This value is associated with the TSM document that receives the Unicode text input.

Keyword	`keyAETSMScriptTag`
Descriptor type	`typeIntlWritingCode`
Data	A `ScriptLangagueRecord` structure that identifies the script code and language code associated with the text returned in the `keyAETheData` parameter. If the current input source is partial Unicode, this contains a Mac OS script code. If the current input source is full Unicode, it is 0x7E (`smUnicodeScript`).

There are no optional parameters.

The return parameter is the following:

Keyword	`keyErrorNumber`
Descriptor type	`typeShortInteger`
Data	Any errors that the application needs to return to the Text Services Manager to terminate processing of the key event that the application passed to `TSMEvent`. The function `TSMEvent` returns `false` to indicate to the applicatoin that the key even was not handled. The application can then attempt to process the event. Note that the character code data in the returned key event is not valid in general, but the virtual key code and modifier-key data can still be processed.

Handling Low-Level Keyboard Events for Applications

While low-level keyboard events appear essentially unchanged with Unicode text input, there are certain differences which can affect how text is converted.

Whether or not a Unicode script system is present, the keyboard driver always uses a `'KCHR'` resource to generate the character codes that are posted in the low-level event. Even if the current keyboard layout is specified solely by a `'uchr'` resource, the Script Manager supplies the keyboard driver with the best approximation of an appropriate `'KCHR'` resource to use. However, the resulting character in the low-level event may have no relation to the actual Unicode character, as specified by the `'uchr'` resource. Also, in this case, when the current keyboard layout is specified by a `'uchr'` resource alone, the Text Services Manager disables driver dead-key processing for `'KCHR'` resources and performs all dead-key processing itself.

If the current keyboard layout is specified only by a partial Unicode `'uchr'` resource, and the current application is not using a Unicode TSM document, the Text Services Manager intercepts the key event posted by the driver before it is delivered to the application. The Text Services Manager uses the `'uchr'` resource with the function `UCKeyTranslate` to map the virtual key code and modifiers in the event to a string of Unicode character codes. It then converts these to character codes in the appropriate Mac OS encoding and post these for delivery to the application in a series of keyboard events. While these appear to your application as normal keyboard events, you cannot automatically reproduce the characters in the events by using the (pre-Unicode) `KeyTranslate` function to convert the key code and modifiers in the event. Instead, you must check to see if a `'uchr'` resource is present to know whether to use `KeyTranslate` or `UCKeyTranslate`.

If the current application is using a Unicode TSM document, the keyboard event posted by the driver is not modified before delivery to the application. Instead, the application is expected to pass the event to the Text Services Manager through the function `TSMEvent`, which handles all necessary `UCKeyTranslate` calls or conversion to Unicode.

For keyboard layouts that have `'uchr'` resources, `TSMEvent` uses `UCKeyTranslate` to convert the keycode and modifiers in the key event to a sequence of Unicode characters. For keyboard layouts that only have `'KCHR'` resources, `TSMEvent` converts the Mac OS encoding character in the event to Unicode.

Providing Unicode Support in Input Methods

While existing applications process inline input text in Mac OS encodings, as applications adopt Unicode they will also support input from Unicode input methods, greatly increasing the characters available to the user in individual scripts and offering a convenient and comprehensive environment for multi-script or multilingual text entry. Also, because text contained in Apple events from Unicode input methods does not need to be converted by the Text Services Manager to Unicode for application delivery, the efficiency of inline input processing is greatly improved.

This section identifies the requirements for development of Unicode input methods. While the main requirement imposed by the Text Services Manager is that these input methods communicate externally using Unicode text, the Text Services Manager does not require that an input method perform its internal processing in Unicode nor that the input method images Unicode text in its user interface (input method palettes), although these features are assumed to be desirable or necessary for other reasons.

Text Services Manager defines two types of Unicode input methods: full Unicode input methods and partial Unicode input methods. A **full Unicode input method** is defined to be an input method which may generate Unicode characters outside of the repertoire of any given Mac OS encoding, in multiple Mac OS encoding repertoires, or both. A **partial Unicode input method** always adheres (externally) to the repertoire of the Mac OS encoding defined by the Mac OS script system to which it belongs.

Partial Unicode input methods appear in the Keyboard menu section for the script to which they belong. Full Unicode input methods and keyboard layouts appear in a new section near the bottom of the Keyboard menu, after the section for Mac OS encodings.

Identifying an Input Method as Supporting Unicode

Both partial and full Unicode input methods continue to be Component Manager components, described by the `ComponentDescription` flags in the component `'thng'` resource. A partial Unicode input method specifies the Mac OS script code with which it is associated, while a full Unicode input method specifies the constant 0x7E (`smUnicodeScript`). Note that while a partial Unicode input method, like a non-Unicode (Mac OS encoding) input method, advertises itself as being associated with a Mac OS script code, it is distinguished by the contents of the `ScriptLanguageRecord` structure that it returns when it responds to a `GetScriptLanguageSupport` call.

The `GetScriptLanguageSupport` function is the mechanism used by the Text Services Manager to distinguish a Mac OS encoding input method from a partial Unicode input method. Since both of these input methods specify a Mac OS script code in the component description flags of the `'thng'` resource, a partial Unicode input method implements its `GetScriptLanguageSupport` function to return an array that includes a `ScriptLanguageRecord` structure with the proper Mac OS language code and a script code of `kTextEncodingUnicodeDefault` (`0x0100`).

Full Unicode input methods, similar to non-Unicode input methods, do not need to implement this function, although a full Unicode input method may wish to return an array of `ScriptLanguageRecord` structures, each specifying the `kTextEncodingUnicodeDefault` constant for the script code and the appropriate language code to identify those languages for which it is most suited.

Table 3-1 shows the relationships of keyboard-layout resources and input methods to differing types of text input, including whether the input method must identify the script systems it supports in a `ScriptLanguageRecord` structure to respond to the Text Services Manager function `GetScriptLanguageSupport`.

Table 2-1 Text input types, keyboard layouts, and input method script systems

Input type	Keyboard layout (resourcetype, ID)	Input method script systems (ComponentDescription flags)	Input methd script systems (ScriptLanguageRecord structure)
Produces Mac OS encoded characters	KCHR, >= 0	Supply any Mac OS script code (0x00-0x20)	Not necessary, but can supply any Mac OS script code (0x00-0x20)
Produces partial Unicode characters	uchr, >=0	Supply any Mac OS script code (0x00-0x20)	Necessary; must supply the 16-bit Unicode script code (0x100 = `kTextEncodingUnicodeDefault`)
Produces full Unicode characters	uchr, < 0	Supply the 7-bit Unicode script code (0x7E-s`mUnicodeScript`)	Not necessary, but can supply the 16-bit Unicode script code (0x100 = `kTextEncodingUnicodeDefault`)

Responding to the UCTextServiceEvent Function

For any Unicode input method, the Text Services Manager always uses the `UCTextServiceEvent` function. This function specifies the low-level event record, but it also contains the Unicode text stream resulting from the keypress. This is important because the keyboard layout being used may be a Unicode keyboard-layout (`'uchr'`) resource, which may generate more than one character as the result of a single keypress or no characters in the case of a dead-key sequence.

Note that the Text Services Manager forwards the key event to the input method in all cases, even when no output is produced by the `'uchr'` resource. Therefore, the input method should be prepared to be called by the `UCTextServiceEvent` function with just the key event and no Unicode text (`unicodeString=NULL`, `unicodeStrLength=0`). This allows input methods to process Option-Shift equivalents without the need to override the keyboard layout data used by the keyboard driver, as sometimes has been necessary in the past.

Supporting Unicode in Text Services Manager Apple Events

A Unicode input method must transmit all text that is sent through Text Services Manager Apple events as Unicode text, in a descriptor whose keyword is `keyAETheData` and whose descriptor type is `typeUnicodeText`. All text offsets specified in these Apple events must specify byte offsets into the corresponding Unicode text. This applies to all currently defined Text Services Manager Apple events: Update Active Input Area, Offset To Position, Position To Offset, and Get Selected Text.

Handling Low-Level Keyboard Events for Input Methods

While low-level keyboard events appear essentially unchanged with Unicode text input, there are certain differences which can affect how text is converted. Whether or not a Unicode script system is present, the keyboard driver always uses a `'KCHR'` resource to generate the character codes that are posted in the low-level event. Even if the current keyboard layout is specified solely by a `'uchr'` resource, the Script Manager will supply the keyboard driver with the best approximation of an appropriate `'KCHR'` resource to use. However, in the latter case, the resulting character in the low-level event may have no relation to the actual Unicode character as specified by the `'uchr'` resource.

Because keyboard drivers are not equipped to handle a Unicode keyboard-layout (`'uchr'`) resource, which may generate more than one character as the result of a single keypress or no characters in the case of a dead-key sequence, there are three cases where the Text Services Manager disables keyboard driver dead-key processing and performs all dead-key processing itself:

- if an input method of any type is in use
- if the current keyboard layout is specified solely by a `'uchr'` resource (that is, if no `'KCHR'` resource is available)
- if the current document identifies itself as a Unicode TSM document and a 'uchr' resource is available

In any of these cases, when the Text Services Manager disables dead-key processing in the keyboard driver, it passes each key event to the `UCKeyTranslate` function, whose output is then forwarded to the input method. When a `'uchr'` is not available for input into a Unicode input method, the Text Services Manager relies on the Text Encoding Converter to generate the Unicode characters.

Handling Compatibility Issues

There are two main compatibility issues for Unicode input methods:

- running on systems with Text Services Manager 1.0
- providing support for applications that do not themselves support Unicode

Unicode input methods of any kind cannot be selected, and are not loaded, on a system with Text Services Manager 1.0. While this is true of both full Unicode input methods and partial Unicode input methods, a partial Unicode input method could be implemented such that it behaves as a Mac OS encoding input method with Text Services Manager 1.0, and a partial Unicode input method with Text Services Manager 1.5. In the presence of Text Services Manager 1.0, the input method could continue to perform its internal processing in Unicode and convert text to Mac encoding using the Text Encoding Converter either for display in its own palettes (if ATSUI is not available) or for Apple event content. The input method's component description flags specify the Mac script in either world, and, in the presence of Text Services Manager 1.5, the input method may respond to a `GetScriptLanguageSupport` call by returning an array that includes a `ScriptLanguageRecord` structure with the proper Mac OS script code and a language code of `kTextEncodingUnicodeDefault`.

Full Unicode input methods cannot be selected by the user unless the current application's active TSM Document is created with the `kUnicodeDocument` interface type. Until Unicode is adopted to a greater extent, input methods may benefit from restricting Unicode output to the repertoire of a single Mac OS script system, and possibly generate Unicode outside of a Mac encoding's repertoire only when it is certain that the current document is a Unicode TSM document.

Using the UCKeyTranslate Function

In most cases, application and input methods do not need to use the `UCKeyTranslate` function because the Text Services Manager automatically calls it when handling input from a Unicode keyboard layout. However, there may be some circumstances when you want to call the function `UCKeyTranslate` directly.

For example, your application may need to determine what character code(s) would have been generated for the virtual key code in the current key-down event if a different modifier-key combination had been used. Listing 3-1 shows how your application can use the function `UCKeyTranslate` to perform its own virtual key code to Unicode character code conversion. Note that this code is a fragment; the ellipses indicates code that you would need to add for your application. The code is intended for use in an application that has an event loop.

Listing 2-1 A code fragment that uses the function UCKeyTranslate in an event loop

```
enum {
    kMaxUnicodeInputStringLength = 16
};

 // Code fragment
    EventRecord *eventPtr;
    UCKeyboardLayout myKeyLayout;
    UInt32 deadKeyState;
    SInt16 currentKeyScript;
    SInt16 lastKeyLayoutID;
    UniChar unicodeInputString[kMaxUnicodeInputStringLength];
    OSStatus status;

    // initialization
    currentKeyScript = GetScriptManagerVariable(smKeyScript);
    lastKeyLayoutID = GetScriptVariable(currentKeyScript, smScriptKeys);
    deadKeyState = 0;
    myKeyLayout = GetResource('uchr', lastKeyLayoutID);
    // …
    // event loop
    while(true)
    {
        // get next event from WaitNextEvent, then
        switch (eventPtr->what)
        {
            //add other relevant cases here
            case keyDown:
            case keyUp:
```

```
        case autoKey:
        {
            SInt16 currentKeyLayoutID;
            currentKeyScript = GetScriptManagerVariable(smKeyScript);
            currentKeyLayoutID = GetScriptVariable(currentKeyScript,
                      smScriptKeys);
            if (currentKeyLayoutID != lastKeyLayoutID){
                // reset the dead key state
                // if the keyboard layout has changed
                deadKeyState = 0;
                // attempt to get the handle for
                // the new keyboard layout's 'uchr'
                myKeyLayout = GetResource('uchr', currentKeyLayoutID);
                lastKeyLayoutID = currentKeyLayoutID;
            }
            // if there is a 'uchr' for the current keyboard layout,
            // use it
            if (myKeyLayout != NULL){
                UInt32 keyboardType;
                UInt32 modifierKeyState;
                UInt16 virtualKeyCode;
                UInt16 keyAction;
                UniCharCount actualStringLength;

                virtualKeyCode = ((eventPtr->message) >> 8) & 0xFF;
                keyAction = eventPtr->what - keyDown;
                modifierKeyState = ((eventPtr->modifiers) >> 8) & 0xFF;
                keyboardType = LMGetKbdType();
                status = UCKeyTranslate(*myKeyLayout,
                            virtualKeyCode, keyAction,
                            modifierKeyState, keyboardType, 0,
                            &deadKeyState,
                            kMaxUnicodeInputStringLength,
                            &actualStringLength, unicodeInputString);
                // now do something with status and unicodeInputString
                // add your code here
            }
            else{
                // no 'uchr' resource, do something with 'KCHR'?
                // add your code here
            }
        }
        break;
    } // end switch on eventPtr->what
} // end of while statement for event loop
```

Document Revision History

This table describes the changes to *Supporting Unicode Input*.

Date	Notes
2005-07-07	Fixed indenting for sample code.
2003-02-17	Removed the Unicode Utilities reference documentation. You can find it in the most recent version of *Unicode Utilities Reference*.
	Removed the `'uchr'` specification. It is now in *Unicode Utilities Reference*.
	Removed or edited out-of-date information, such as references to nonCarbon features.
	Fixed typographical errors; updated formatting.
1998-10-01	First version of this document.