Text Services Manager Reference

Carbon > Text & Fonts

2005-11-09

Contents

Tables

Text Services Manager Reference

Framework:	Carbon/Carbon.h
Declared in	CarbonEvents.h TextServices.h

Overview

The Text Services Manager ("TSM") provides an environment for applications to use non-application-specific text services. 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 exists so that these two types of programs can work together without needing to know anything about the internal structures or identities of each other.

A **client application** is any text-processing program that uses the Text Services Manager to request a service from a text service component. To accomplish this, a client application needs to make specific Text Services Manager calls during execution.

A **text service component** is a utility program that uses the Text Services Manager to provide a text service to an application. Text service components are registered components with the Component Manager. Text services can include many different types of specific text-handling tasks, including spell-checking, hyphenation, and handling the input of complex text.

The most prevalent category of text services are those that handle the entry of complex text, that is, input methods. A typical example of an input method is a service that converts keyboard input into text that cannot be directly entered via a keyboard. Text input in Japanese, Chinese, Korean, or Unicode usually requires an input method.

TSM introduces input modes in Mac OS X version 10.3. An **input mode** allows an input method to temporarily accept text input in a script other than the one it normally supports. An input method uses a CFDictionary to define the input modes it supports and the tag `kTextServiceInputModePropertyTag` to specify that the input method supports input modes. An application finds out what input modes are supported by an input method by calling the function `CopyTextServiceInputModeList`.

Also new in Mac OS X version 10.3 is a suite of Carbon events that allow a text service relatively direct access to a document's text content and text attributes, such as font and glyph information. To take advantage of this new functionality in TSM, all text and offsets in your application must map to and from a flattened Unicode space. Your application must also implement callback functions to handle the appropriate Carbon events.

Mac OS X version 10.4 introduces input mode palette configuration routines.

The Text Services Manager defines three separate programming interfaces:

- The first are functions implemented by the Text Services Manager and called by the application clients of text service components.
- The second are functions implemented by the Text Services Manager and called by text service components.
- The third are low-level functions implemented by text service components and called by either applications or the Text Services Manager.

Functions by Task

Applications - Facilitating User Interactions With Components

`CopyTextServiceInputModeList` (page 14)
: Obtains a copy of the set of input modes supported by a keyboard-class input method.

`CloseTextService` (page 69) Deprecated in Mac OS X v10.5
: Closes a text service component, other than an input method, and disassociates it from the active TSM document.

`GetDefaultInputMethod` (page 70) Deprecated in Mac OS X v10.5
: Obtains the default input method text service component for a given script and language.

`GetDefaultInputMethodOfClass` (page 71) Deprecated in Mac OS X v10.5
: Obtains the default input method text service component for a given text service class.

`GetServiceList` (page 74) Deprecated in Mac OS X v10.5
: Obtains a list of the text service components of a specified type that are currently available.

`GetTextServiceLanguage` (page 75) Deprecated in Mac OS X v10.5
: Obtains the current input script and language.

`OpenTextService` (page 76) Deprecated in Mac OS X v10.5
: Opens a text service component, other than an input method, and associates it with a TSM document.

`SetDefaultInputMethod` (page 80) Deprecated in Mac OS X v10.5
: Sets a default input method to a given script and language.

`SetDefaultInputMethodOfClass` (page 80) Deprecated in Mac OS X v10.5
: Sets the default input method text service component for a given text service class.

`SetTextServiceLanguage` (page 81) Deprecated in Mac OS X v10.5
: Changes the current input script and language.

`TSMCopyInputMethodEnabledInputModes` (page 82) Deprecated in Mac OS X v10.5
: Obtain the array of the enabled (and visible) input modes for a component.

`TSMSelectInputMode` (page 83) Deprecated in Mac OS X v10.5
: Sets the specified input method input mode as the current input source.

Applications - Managing TSM Documents

`NewTSMDocument` (page 23)
: Creates a TSM document and returns a handle to the document's ID.

`DeleteTSMDocument` (page 17)
Closes all opened text service components for the TSM document.

`ActivateTSMDocument` (page 14)
Informs the Text Services Manager that a TSM document is active.

`DeactivateTSMDocument` (page 17)
Informs the Text Services Manager that a TSM document is inactive.

`FixTSMDocument` (page 19)
Informs the Text Services Manager that user input for a TSM document has been interrupted.

`UseInputWindow` (page 31)
Associates a floating input window with one or more TSM documents.

`TSMGetActiveDocument` (page 28)
Obtains the active TSM document in the current application context.

`TSMSetInlineInputRegion` (page 84) Deprecated in Mac OS X v10.5
Defines a region within a TSM document in which inline input can occur.

Components - Sending Events

`SendTextInputEvent` (page 25)
Sends Carbon text input events from a text service component to a client application.

`SendAEFromTSMComponent` (page 78) Deprecated in Mac OS X v10.5
Sends Apple events from a text service component to a client application.

Low Level - Accessing Text Service Properties

`SetTextServiceProperty` (page 26)
Notifies a text service component that one of its properties has been selected.

`GetTextServiceProperty` (page 20)
Notifies a text service component that it must identify the current value of one of its properties.

Low Level - Confirming Text Service Input

`FixTextService` (page 18)
Notifies a text service component that it must complete the processing of any input that is in progress.

Low Level - Managing Text Service States

`InitiateTextService` (page 22)
Notifies a text service component that it must perform any necessary set-up tasks and begin operating.

`ActivateTextService` (page 13)
Notifies a text service component that its associated document window is becoming active.

`DeactivateTextService` (page 16)
Notifies a text service component that its associated document window is becoming inactive.

`TerminateTextService` (page 27)
Notifies a text service component that it must terminate its operations in preparation for closing.

`HidePaletteWindows` (page 21)
Notifies a text service component that it must hide its floating windows.

`DeselectTextService` (page 70) Deprecated in Mac OS X v10.5
Notifies TSM that an input method has been closed.

`IsTextServiceSelected` (page 76) Deprecated in Mac OS X v10.5
Determines if a text service component is selected.

`SelectTextService` (page 77) Deprecated in Mac OS X v10.5
Selects a text service.

Low Level - Querying Text Services

`GetTextServiceMenu` (page 20)
Notifies a text service component that it must produce a handle to its menu.

`GetScriptLanguageSupport` (page 72) Deprecated in Mac OS X v10.5
Notifies a text service component that it must produce a list of its supported languages and scripts.

Low Level - Sending Events to Text Services

`TextServiceEventRef` (page 27)
Provides an opportunity for a text service component to handle a Carbon event.

Working With Document Properties

`TSMSetDocumentProperty` (page 31)
Sets a property for a TSM document.

`TSMGetDocumentProperty` (page 29)
Obtains a TSM document property.

`TSMRemoveDocumentProperty` (page 30)
Removes a property from a TSM document.

Input Mode Palette Configuration

`GetInputModePaletteMenu` (page 72) Deprecated in Mac OS X v10.5
Obtains from an input method the menu to display for a pull-down menu on the input mode palette.

`InputModePaletteItemHit` (page 75) Deprecated in Mac OS X v10.5
Informs an input method that a function button on the input mode palette was pressed.

`TSMInputModePaletteLoadButtons` (page 82) Deprecated in Mac OS X v10.5
Notifies the input mode palette of changes to the controls for an input method and replaces the current controls with the new control array.

`TSMInputModePaletteUpdateButtons` (page 83) Deprecated in Mac OS X v10.5
: Notifies the input mode palette of changes to the controls for an input method and updates the controls.

Functions

ActivateTextService

Notifies a text service component that its associated document window is becoming active.

```
ComponentResult ActivateTextService (
   ComponentInstance ts
);
```

Parameters

ts

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 76).

Return Value

See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

Text service components must implement a function for this call.

The appropriate response to `ActivateTextService` is for the text service component to restore its active state, including displaying all floating windows if they have been hidden. (Note that typically an input-method component should not hide its windows in response to being deactivated. If the subsequent document being activated is using the same component's service, it would be irritating to the user to hide and then immediately redisplay the same windows. An input-method component should hide its windows only in response to a `HidePaletteWindows` call.) If the component is an input method, it should specify the redisplay of any unconfirmed text currently in the active input area.

The Text Services Manager makes this call either on its own or in response to application-interface calls it receives from client applications. Client applications may directly make this call, but the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

ActivateTSMDocument

Informs the Text Services Manager that a TSM document is active.

```
OSErr ActivateTSMDocument (
   TSMDocumentID idocID
);
```

Parameters

idocID

A TSM document identification number created by a prior call to the `NewTSMDocument` (page 23) function.

Return Value

A result code. See "Text Services Manager Result Codes" (page 66).

Discussion

When a window that has an associated TSM document becomes active, your client application must call the `ActivateTSMDocument` function to inform the Text Services Manager that the document is activated and is ready to use text service components.

`ActivateTSMDocument` calls the equivalent text service component function `ActivateTextService` (page 13) for all open text service components associated with the TSM document.

If a text service component has a menu, the Text Services Manager inserts the menu into the menu bar.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

CopyTextServiceInputModeList

Obtains a copy of the set of input modes supported by a keyboard-class input method.

```
ComponentResult CopyTextServiceInputModeList (
   ComponentInstance ts,
   CFDictionaryRef *outInputModes
);
```

Parameters

ts

The component whose set of input modes you want to obtain.

outInputModes

On output, the CFDictionary contains the list of supported input modes. See the Discussion for more information on the structure and requirements of the dictionary.

Return Value

See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

This function is supported by input methods that adopt the input mode protocol. If this component call is not supported by an input method, calls to functions that access text service properties (using the tag `kTextServiceInputModePropertyTag`) return the result `tsmComponentPropertyUnsupportedErr`.

The CFDictionary of input modes (available in TSM 2.2 and later) must have the following form:

```
<dict>
<key> kTSInputModeListKey </key>
<dict>
<key> modeSignature : (internal ascii name)</key>
<!-- This can be any of the generic input modes defined in this  file,-->
<!-- such as kTextServiceInputModeRoman,or can be a private input-->
<!-- mode such as CFSTR("com.apple.MyInputmethod.Japanese.CoolMode")  -->
 <dict>
<key>menuIconFile</key>
<string> (path for menu icon image file)</string>
<key>alternateMenuIconFile</key>
<string> (path for alternate menu icon image file, when item  is hilited)</string>
<key>paletteIconFile</key>
<string> (path for palette icon image file) </string>
<key>defaultState</key>
<boolean> (default on/off state) </boolean>
<key>script</key>
<string> (scriptCode string for this mode, for example, "smRoman")  </string>
<key>primaryInScript</key>
<boolean> (true if this is primary mode in this script) </boolean>
<key>isVisible</key>
<boolean> (true if this input mode should appear in System  UI) </boolean>
<key>keyEquivalentModifiers</key>
<integer> (modifiers)</integer>
<key>keyEquivalent</key>
<string> (key equivalent character) </string>
<key>JISKeyboardShortcut</key>
<integer> (optional: 0=none,1=hiragana, 2=katakana, 3=eisu)  </integer>
</dict>
</dict>
</dict>
```

This dictionary must also be present in the Info.plist for the component bundle, in addition to being available through this component call. Availability in the Info.plist allows retrieval of input modes by the system without opening the component. The component call is used whenever the system is notified of a change in the contents of the input mode list, such as when the name or key-equivalents of individual input modes have changed.

If, when the input method is first activated in a login session, the settings of the individual input modes (names or key-equivalents) differ from the default settings as found in the component bundle Info.plist, the system needs to be notified of the change. The input method does this by sending out the Carbon event `kEventTextInputInputMenuChanged`, just as when the change originally took place.

For more information on the dictionary keys used to define input modes and the input mode dictionary, see "Input Mode Dictionary Key" (page 52) and "Individual Input Mode Keys" (page 54).

Availability

Not available in CarbonLib.

Available in Mac OS X v 10.3 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

DeactivateTextService

Notifies a text service component that its associated document window is becoming inactive.

```
ComponentResult DeactivateTextService (
   ComponentInstance ts
);
```

Parameters

ts

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 76).

Return Value

See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

Text service components must implement a function for this call.

When it receives a `DeactivateTextService` call, the text service component is responsible for saving whatever state information it needs to save, so that it can restore the proper information when it becomes active again. Note that an input method should not confirm any unconfirmed text in the active input area, but should save it until reactivated.

A component other than an input method should hide all its floating windows and menus. However, an input-method component should not hide its windows in response to this call. If the subsequent document being activated is using the same component's service, it would be irritating to the user to hide and then immediately redisplay the same windows. An input-method component should hide its windows only in response to a `HidePaletteWindows` call.

The Text Services Manager makes this call either on its own or in response to application-interface calls it receives from client applications. Client applications may directly make this call, but the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

DeactivateTSMDocument

Informs the Text Services Manager that a TSM document is inactive.

```
OSErr DeactivateTSMDocument (
   TSMDocumentID idocID
);
```

Parameters

idocID

A TSM document identification number created by a prior call to the `NewTSMDocument` (page 23) function.

Return Value

A result code. See "Text Services Manager Result Codes" (page 66).

Discussion

The `DeactivateTSMDocument` function lets you inform the Text Services Manager that a TSM document in your client application is no longer active and must temporarily stop using text service components.

The Text Services Manager calls the equivalent text service component function `DeactivateTextService` (page 16) for any text service component associated with the TSM document being deactivated.

An application that supports inline input should always strive to have a TSM document active at all times. If a situation arises in which all TSM documents are inactive and keyboard input occurs, the Text Services Manager automatically interacts with the user via its floating input window. (This is the same floating window that the Text Services Manager displays if an application calls the function `UseInputWindow` (page 31) with a value of `TRUE` for the `useWindow` parameter.)

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

DeleteTSMDocument

Closes all opened text service components for the TSM document.

```
OSErr DeleteTSMDocument (
   TSMDocumentID idocID
);
```

Parameters

idocID

A TSM document identification number created by a prior call to the `NewTSMDocument` (page 23) function.

Return Value

A result code. See "Text Services Manager Result Codes" (page 66).

Discussion

When your application disposes of a TSM document, it must call the `DeleteTSMDocument` function to inform the Text Services Manager that the document is no longer using text service components. `DeleteTSMDocument` invokes the Component Manager `CloseComponent` function for each open text service component associated with this document. It also disposes of the internal data structure for the TSM document.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

FixTextService

Notifies a text service component that it must complete the processing of any input that is in progress.

```
ComponentResult FixTextService (
   ComponentInstance ts
);
```

Parameters

ts

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 76).

Return Value

See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

Text service components must implement a function for this call.

For input method components, this function is equivalent to the user explicitly confirming text, but in this case the request comes instead from the application or from the Text Services Manager. Typically, users confirm text explicitly (such as by pressing the Return key), and input methods continually process these user events and send the confirmed text to client applications. Circumstances may arise, however, in which an application needs the input method to confirm and send input without an explicit confirmation from the user.

If, for example, the user clicks the mouse in text outside the active input area, that constitutes an implicit user acceptance of the text in the active input area. In this case, applications should explicitly terminate any active input by calling the `FixTSMDocument` (page 19) function, which notifies the Text Services Manager. The Text Services Manager then calls the `FixTextService` function, which notifies the text service component that it must stop accepting further input and pass the current contents (both converted and unconverted) of the active input area as confirmed text to the client application.

Client applications may directly make this call, but the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X v 10.0 and later.
Not available to 64-bit applications.

Declared In
`TextServices.h`

FixTSMDocument

Informs the Text Services Manager that user input for a TSM document has been interrupted.

```
OSErr FixTSMDocument (
   TSMDocumentID idocID
);
```

Parameters

`idocID`
The identification number of a TSM document created by a prior call to the `NewTSMDocument` (page 23) function.

Return Value
A result code. See "Text Services Manager Result Codes" (page 66).

Discussion
Typically, an inline input text service component removes confirmed input from the active input area each time the user presses the Return key, and passes the confirmed text to your application through a Carbon event or an Apple event.

In certain situations, however, your client application may need to inform the text service component that input in the active input area of a specified TSM document has been interrupted, and that the text service component must confirm the text and terminate user input.In this case you call the `FixTSMDocument` function to give the input method text service component the opportunity to confirm any input in progress.

For instance, if the user clicks in the close box of the window in which active input is taking place, call `FixTSMDocument` before you close the window. The text service component will pass you the current contents (both converted and unconverted) of the active input area as confirmed text.

For simple activating and deactivating of your application's window, it is not necessary to confirm the text in the active inline area. The input method saves the text and restores it when your window is reactivated.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X v 10.0 and later.
Not available to 64-bit applications.

Declared In
`TextServices.h`

GetTextServiceMenu

Notifies a text service component that it must produce a handle to its menu.

```
ComponentResult GetTextServiceMenu (
   ComponentInstance ts,
   MenuRef *serviceMenu
);
```

Parameters

ts

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 76).

serviceMenu

A pointer to a menu handle (defined by the Menu Manager `MenuHandle` data type) for the text service component that is to be updated. The menu handle may be preallocated or it may be `NULL`. If the menu handle is `NULL`, the text service component should allocate a new menu and return it. On Mac OS 8 and 9, note that all instances of a component must share a single menu handle, allocated in the system heap. On Mac OS X, all instances of a component must share a single menu handle within an application's context.

Return Value

If the text service component does not have a menu, it should return a `ComponentResult` value of `TSMHasNoMenuErr`. See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

Text service components must implement a function for this call.

The Text Services Manager calls `GetTextServiceMenu` when a component is opened or activated, so that it can put the component's menu on the menu bar.

Client applications may directly make this call, but the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

GetTextServiceProperty

Notifies a text service component that it must identify the current value of one of its properties.

```
ComponentResult GetTextServiceProperty (
   ComponentInstance ts,
   TextServicePropertyTag inPropertyTag,
   TextServicePropertyValue *outPropertyValue
);
```

Parameters

`ts`

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 76).

`propertySelector`

A constant identifying a general property of a text service. For descriptions of the system-defined property selectors, see "Text Service Properties" (page 59).

`result`

On return, a constant specifying the value for a text service property. For descriptions of the system-defined property values, see "Text Service Properties" (page 59).

Return Value

See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

Text service components have the option of implementing a function for this call.

Both the Text Services Manager and client applications can call this function to manage text service properties. If client applications directly make this call, the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

HidePaletteWindows

Notifies a text service component that it must hide its floating windows.

```
ComponentResult HidePaletteWindows (
   ComponentInstance ts
);
```

Parameters

ts

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 76).

Return Value

If the text service component has no palettes, it should return a `ComponentResult` value of `noErr`. See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

Text service components must implement a function for this call.

If a window associated with a TSM document associated with your text service is being deactivated, your text service component receives the `DeactivateTextService` (page 16) call. You should perform any necessary cleanup or other tasks associated with deactivating your current component instance. If your text service component is not an input method, you should also hide all floating windows associated with the document being deactivated. If your text service component is an input method and if the newly activated document does not use your text services, you receive the `HidePaletteWindows` call. When it receives a `HidePaletteWindows` call, your input method should hide all its floating and nonfloating windows associated with the component instance being deactivated. Its menus, if any, will be removed from the menu bar by the Text Services Manager.

The Text Services Manager makes this call either on its own or in response to application-interface calls it receives from client applications. Client applications may directly make this call, but the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

InitiateTextService

Notifies a text service component that it must perform any necessary set-up tasks and begin operating.

```
ComponentResult InitiateTextService (
   ComponentInstance ts
);
```

Parameters

`ts`

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 76).

Return Value

This function should return a `ComponentResult` value of zero if there is no error, and an error code if there is one. See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

Text service components must implement a function for this call.

The Text Services Manager can call `InitiateTextService` to any component that it has already opened with the Component Manager `OpenComponent` or `OpenDefaultComponent` functions. Text service components should be prepared to handle `InitiateTextService` calls at any time.

Any text service component can receive multiple `InitiateTextService` calls. The Text Services Manager calls `InitiateTextService` each time the user adds a text service to a TSM document, even if the text service component has already been opened. This provides an opportunity for the component to restart or to display user interface elements that the user may have closed.

The Text Services Manager makes this call either on its own or in response to application-interface calls it receives from client applications. Client applications may directly make this call, but the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

NewTSMDocument

Creates a TSM document and returns a handle to the document's ID.

```
OSErr NewTSMDocument (
   SInt16 numOfInterface,
   InterfaceTypeList supportedInterfaceTypes,
   TSMDocumentID *idocID,
   SRefCon refcon
);
```

Parameters

numOfInterface

The number of text service interface types that your application supports.

supportedInterfaceTypes

A value of type `InterfaceTypeList` (page 33) specifying the kinds of text services that your program supports. This list helps the Text Services Manager locate the text services that have the correct interface type.

idocID

Upon successful completion of the call, a pointer to the document identification number of the TSM document created. If `NewTSMDocument` fails to create a new TSM document, it returns an error and sets `idocID` to `NULL`.

refcon

A reference constant to be associated with the TSM document. This may have any value you wish.

Return Value

A result code. See "Text Services Manager Result Codes" (page 66).

Discussion

Each time your client application calls the `NewTSMDocument` function, the Text Services Manager creates an internal structure called a TSM document and returns its ID.

If the call is successful, `NewTSMDocument` opens the default input method text service component of the current keyboard script and assigns it to this document. If `NewTSMDocument` returns `tsmScriptHasNoIMErr`, it has still created a valid TSM document, but has not associated an input method with it.

Starting in Mac OS X v10.3, the `NewTSMDocument` function turns on the `kTSMDocumentUnicodeInputWindowPropertyTag` implicitly for TSMDocuments of interface type `kUnicodeDocumentInterfaceType`. The effect is that Unicode input sources (keyboard layouts) can remain available, not only in an editing mode where Unicode is supported by an application, but also outside of editing mode where no TSMDocument in particular is active.

This change also provides compatibility with many applications that relied on Unicode input being available even without activating any of their own TSMDocuments, as well as other applications that do create their own Unicode TSMDocument but call the function `UseInputWindow` passing `true` (which was really an undefined operation in the original Unicode/TSM specification).

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

SendTextInputEvent

Sends Carbon text input events from a text service component to a client application.

```
OSStatus SendTextInputEvent (
   EventRef inEvent
);
```

Parameters

inEvent

A reference to the Carbon event to be sent.

Return Value

A result code. See "Text Services Manager Result Codes" (page 66). The `SendTextInputEvent` function returns `noErr` if the event is successfully handled. If the event is not handled, the function may return a Carbon Event Manager error, as well as Apple event or Text Encoding Conversion Manager errors.

Discussion

The `SendTextInputEvent` function allows a Carbon text service component on Mac OS X to send a Carbon text input event to the Text Services Manager for dispatching to a client application. This function can be used for events of Carbon event class `kEventClassTextInput` as well as for events of class `kEventClassTSMDocumentAccess`.

If the client application does not handle a particular Carbon text input event, the Text Services Manager converts the event to the corresponding Apple event and sends it again. An exception to this is when the application is not Unicode-aware (that is, the active TSM document was not created with the `kUnicodeDocument` interface type). In this case, a `kEventUnicodeForKeyEvent` Carbon event would not be converted to the corresponding Apple event (`kUnicodeNotFromInputMethod`). In every case, if the application handles neither the Unicode Carbon text input event nor the corresponding Apple event, the Text Services Manager converts the component's text input event into a stream of "classic" key events for delivery to `WaitNextEvent` clients.

If the application has no active TSM documents or has called the function `UseInputWindow` (page 31) to request input via the Text Services Manager's floating input window—that is, if the application does not handle the event at all—the Text Services Manager routes the component's text input event to the floating input window to allow bottom-line input.

Availability

Not available in CarbonLib.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Carbon Porting Notes

Note that this function replaces the function `SendAEFromTSMComponent` (page 78) on Mac OS X only. With Mac OS X, text service components must be Carbon clients. This is in contrast to Mac OS 8 and 9, where text service components must not be Carbon clients. (This restriction is due to the fact that it is potentially destabilizing for a Carbon-based component to load Carbon in the context of a non-Carbon application.) Therefore, text service components use Carbon text input events and the `SendTextInputEvent` function only on Mac OS X. The function `SendAEFromTSMComponent` must be used by components running on Mac OS 8 and 9.

On any system, the Text Services Manager automatically converts component-originated text input events to the proper form for client applications. On Mac OS X, the Text Services Manager automatically converts component-originated Carbon events to Apple events, if a client application does not provide handlers for

Carbon events. Conversely, on Mac OS 8 and 9, the Text Services Manager automatically converts component-originated Apple events to Carbon events and provides these Carbon events to applications, so they have the option of handling them.

Declared In
`TextServices.h`

SetTextServiceProperty

Notifies a text service component that one of its properties has been selected.

```
ComponentResult SetTextServiceProperty (
   ComponentInstance ts,
   TextServicePropertyTag inPropertyTag,
   TextServicePropertyValue inPropertyValue
);
```

Parameters

`ts`

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 76).

`propertySelector`

A constant identifying a general property of a text service. For descriptions of the system-defined property selectors, see "Text Service Properties" (page 59).

`value`

A constant specifying a particular value for the text service property. For descriptions of the system-defined property values, see "Text Service Properties" (page 59).

Return Value
See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion
Text service components have the option of implementing a function for this call.

An application can call `SetTextServiceProperty` to request that a text service component use a specific feature or functionality of the component's program. For example, if an application knows that a Japanese input method which supports various typing methods is the currently active input method, the application can solicit the user's preference of typing methods. Then the application can call `SetTextServiceProperty` to request that the input method use the preferred typing method, for example, Roman or Kana. Currently, the only properties that are defined by the system are typing methods for Japanese input methods.

Both the Text Services Manager and client applications can call this function to manage text service properties. If client applications directly make this call, the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability
Available in CarbonLib 1.0 and later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In
`TextServices.h`

TerminateTextService

Notifies a text service component that it must terminate its operations in preparation for closing.

```
ComponentResult TerminateTextService (
   ComponentInstance ts
);
```

Parameters

`ts`

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 76).

Return Value
If the text service component needs to remain open, it should return an `OSErr` value in the component result return value. This could happen, for example, if the user chooses Cancel in response to a text service component dialog box. See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion
Text service components must implement a function for this call.

The Text Services Manager calls `TerminateTextService` before closing the component instance. A text service component must use this opportunity to confirm any inline input in progress. If this call is made to the last open instance of a text service component, the component should hide any open palette windows. If it is an input method, the component should not dispose of its menu handle if it has a menu.

The Text Services Manager makes this call either on its own or in response to application-interface calls it receives from client applications. Client applications may directly make this call, but the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In
`TextServices.h`

TextServiceEventRef

Provides an opportunity for a text service component to handle a Carbon event.

```
ComponentResult TextServiceEventRef (
   ComponentInstance ts,
   EventRef event
);
```

Parameters

ts

A Component Manager value of type ComponentInstance that identifies the component being called. When the Text Services Manager makes this call, it passes the ComponentInstance value returned by its call to the OpenComponent function. If an application makes this call, it may use the ComponentInstance value obtained from the kEventParamTextInputSendComponentInstance parameter of the Carbon event or the keyAEServerInstance parameter of an Apple event sent by the component being called. Alternately, an application may obtain a ComponentInstance value from a prior call to the function OpenTextService (page 76).

event

A reference to the Carbon event being passed to the component.

Return Value

If the text service component handles the event, it should return a nonzero value for componentResult. If it does not handle the event, it should return 0. Note that the Text Services Manager clones an event before passing it to a component, so any changes made to the contents of an event by the text service have no effect on the original event. See the Component Manager documentation for a description of the ComponentResult data type.

Discussion

Carbon text service components (that is, Mac OS X text services) must implement a function for this call.

The Text Services Manager automatically passes raw keyboard Carbon events (events of class kEventClassKeyboard) and some Carbon mouse events to text service components associated with an active TSM document. The Text Services Manager passes mouse-click events (kEventMouseDown, kEventMouseUp, kEventMouseDragged) to active text services directly. However, the Text Services Manager does not send kEventMouseMoved events to text service components. Instead, when a mouse-moved event occurs inside an inline input region (as registered via an application call to the TSMSetInlineInputRegion function), the Text Services Manager promotes the kEventMouseMoved event to the window-specific kEventWindowCursorChange event, which it then sends to the text service. For more details, see the function TSMSetInlineInputRegion (page 84).

Both the Text Services Manager and client applications can call this function to send Carbon events to components. If client applications directly make this call, the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

TextServices.h

TSMGetActiveDocument

Obtains the active TSM document in the current application context.

```
TSMDocumentID TSMGetActiveDocument (
   void
);
```

Parameters

Return Value
If the Text Services Manager has enabled bottom line input because no TSM document is active, `NULL` is returned. See the description of the `TSMDocumentID` data type.

Discussion
This function can be useful to identify whether the currently active TSM document belongs to the application, or whether it may belong to a control or a plug-in which has user focus within the application's window.

Availability
Available in CarbonLib 1.3 and later.

Available in Mac OS X v 10.0 and later.

Declared In
`TextServices.h`

TSMGetDocumentProperty

Obtains a TSM document property.

```
OSStatus TSMGetDocumentProperty (
   TSMDocumentID docID,
   TSMDocumentPropertyTag propertyTag,
   UInt32 bufferSize,
   UInt32 *actualSize,
   void *propertyBuffer
);
```

Parameters

docID
The `TSMDocumentID` that identifies the document whose property you want to obtain.

propertyTag
A tag that specifies the property you want to obtain.

bufferSize
The size of the data pointed to by the `propertyBuffer` parameter. See the Discussion for what to supply.

actualSize
On return, the actual size of the data.

propertyBuffer
On return, a pointer to the property data.

Return Value
A result code. See "Text Services Manager Result Codes" (page 66).

Discussion
You can call the function `TSMGetDocumentProperty` to retrieve arbitrary data with a specific TSM document. You associated arbitrary data with a TSM document by calling the function `TSMSetDocumentProperty`.

Input methods can call the function `TSMGetDocumentProperty` to determine whether the application that owns the document supports the glyph ID specification (`kTSMDocumentPropertySupportGlyphInfo`).

You can call the function `TSMGetDocumentProperty` to check for the following predefined properties:

- `kUseFloatingWindowTag`. The presence of this property indicates the document is using bottom-line floating window for input. See "Collection Tags" (page 63) for more information.
- `kUnicodeDocument`. The presence of this property indicates the document is Unicode-savvy. See "Unicode Identifiers" (page 62) for more information.
- `kTSMTEInterfaceType`. The presence of this property indicates a TSM Text Edit interface. See "TSM Document Interfaces" (page 61) for more information.
- `kTextService`. The presence of this property indicates the document is not Unicode-savvy. See "TSM Document Interfaces" (page 61) for more information.

These properties do not have any data associated with them. If the function `TSMGetDocumentProperty` returns `noErr` when you call the function with one of these properties, it indicated the property is present.

Typically you need to call this function twice, as follows:

1. Pass the document ID for the document, the tag that specifies the property you want to obtain, 0 for the `bufferSize` parameter, `NULL` for the `actualSize` parameter, and `NULL` for the `propertyBuffer` parameter.

2. Allocate enough space for a buffer of the returned size, then call the function again, passing a pointer in the `propertyBuffer` parameter. On return, the pointer references the property data.

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.2 and later.

Declared In
`TextServices.h`

TSMRemoveDocumentProperty

Removes a property from a TSM document.

```
OSStatus TSMRemoveDocumentProperty (
   TSMDocumentID docID,
   TSMDocumentPropertyTag propertyTag
);
```

Parameters

`docID`
The `TSMDocumentID` that identifies the document whose property you want to obtain.

`propertyTag`
A tag that specifies the property you want to remove.

Return Value
A result code. See "Text Services Manager Result Codes" (page 66).

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.2 and later.

Declared In
`TextServices.h`

TSMSetDocumentProperty

Sets a property for a TSM document.

```
OSStatus TSMSetDocumentProperty (
   TSMDocumentID docID,
   TSMDocumentPropertyTag propertyTag,
   UInt32 propertySize,
   void *propertyData
);
```

Parameters

docID
The `TSMDocumentID` that identifies the document whose property you want to set.

propertyTag
A tag that specifies the property you want to set.

propertySize
The size of the property data.

propertyData
A pointer to the property data.

Return Value
A result code. See "Text Services Manager Result Codes" (page 66).

Discussion
You can call the function `TSMSetDocumentProperty` to associate arbitrary data with a specific TSM document. You can call the function `TSMGetDocumentProperty` to retrieve arbitrary data.

If your application supports input of unencoded glyphs you must notify the Text Service Manager and input methods by setting the glyph ID specification (`kTSMDocumentPropertySupportGlyphInfo`) as a property of each TSM document.

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.2 and later.

Declared In
`TextServices.h`

UseInputWindow

Associates a floating input window with one or more TSM documents.

```
OSErr UseInputWindow (
   TSMDocumentID idocID,
   Boolean useWindow
);
```

Parameters

idocID

The TSM document ID of the particular TSM document to be associated with the floating input window. If `NULL`, this call affects all your application's TSM documents.

useWindow

Indicates whether to use the floating input window. Pass `TRUE` if you want to use a floating window; pass `FALSE` if you do not want to use a floating window.

Return Value

A result code. See "Text Services Manager Result Codes" (page 66).

Discussion

The Text Services Manager provides a floating input window for your application's use if you call `UseInputWindow` with a value of `TRUE` in the `useWindow` parameter. To specify inline input instead, call `UseInputWindow` with a value of `FALSE` in the `useWindow` parameter.

The default value for `useWindow` is `FALSE`; if you do not call `UseInputWindow`, the Text Services Manager assumes that your application wants to use inline input. If your application wants to save the user's choice, it can put the last-used value for `useWindow` in a preferences file before quitting.

If you pass a valid TSM document ID for the `idocID` parameter, the `useWindow` parameter affects only that TSM document. If you pass `NULL` for the `idocID` parameter, the `useWindow` parameter affects all your application's TSM documents, including documents you create after making this call.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

Data Types

TSM Document Interface Type

Defines an interface type for a TSM document.

```
typedef OSType TSMDocumentInterfaceType;
```

Discussion

As of Mac OS X version 10.3, TSM interface types are also stored as TSM document properties. Interface types are a subset of TSM document properties; not all properties are interface types. Once a TSM document is created, you can easily find out its interface types at document creation. See "TSM Document Interfaces" (page 61) for a list of the possible interface types.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`TextServices.h`

InterfaceTypeList

An array of four-character codes identifying Text Services Manager interface types.

```
typedef OSType InterfaceTypeList[1];
```

Discussion
The `InterfaceTypeList` type is used in the function `NewTSMDocument` (page 23) to identify the type of interfaces that an application supports and in the function `GetServiceList` (page 74) to identify the types of interfaces that are currently available. See "TSM Document Interfaces" (page 61) for a list of the possible interfaces.

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

Declared In
`TextServices.h`

ScriptLanguageRecord

Identifies a specific script-language combination.

```
struct ScriptLanguageRecord {
    ScriptCode fScript;
    LangCode fLanguage;
};
typedef struct ScriptLanguageRecord ScriptLanguageRecord;
```

Fields

`fScript`
A `ScriptCode` value identifying a particular set of written characters (for example, Roman versus Cyrillic) and their encoding.

`fLanguage`
A `LangCode` value identifying a particular language (for example, English), as represented using a particular `ScriptCode` value.

Discussion
Structures of type ScriptLanguageRecord are used in the functions `SetDefaultInputMethod` (page 80), `GetDefaultInputMethod` (page 70), `SetTextServiceLanguage` (page 81), and `GetTextServiceLanguage` (page 75).

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

Declared In
`TextServices.h`

ScriptLanguageSupport

Lists a component's supported scripts and languages.

```
struct ScriptLanguageSupport {
    short fScriptLanguageCount;
    ScriptLanguageRecord fScriptLanguageArray[1];
};
typedef struct ScriptLanguageSupport ScriptLanguageSupport;
typedef ScriptLanguageSupport * ScriptLanguageSupportPtr;
typedef ScriptLanguageSupportPtr * ScriptLanguageSupportHandle;
```

Fields

`fScriptLanguageCount`

An integer specifying the number of `ScriptLanguageRecord` structures provided in the `fScriptLanguageArray` field.

`fScriptLanguageArray`

A variable-length array of structures of type `ScriptLanguageRecord` (page 33). Each of these structures identifies a specific script-language combination.

Discussion

A structure of type `ScriptLanguageSupport` is used in the function `GetScriptLanguageSupport` (page 72) to list all of a component's supported scripts and languages. If you are a component developer filling out a `ScriptLanguageSupport` structure, you should start with the component's primary script and language as specified in the `componentFlags` field of its `ComponentDescription` structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextServices.h`

TextServiceInfo

Identifies a single text service component by name and `Component` value.

```
struct TextServiceInfo {
    Component fComponent;
    Str255 fItemName;
};
typedef struct TextServiceInfo TextServiceInfo;
typedef TextServiceInfo *TextServiceInfoPtr;
```

Fields

`fComponent`

A Component Manager value of type `Component`. A `Component` value is a pointer to an opaque structure called a `ComponentRecord` that describes a component. You must supply a `Component` value in the function `OpenTextService` (page 76).

`fItemName`

A Pascal string with the name of a text service component. (The script system to use for displaying the string is specified in the `componentFlags` field of a `ComponentDescription` structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`TextServices.h`

TextServiceList

Lists one or more text service components by name and `Component` value.

```
struct TextServiceList {
    short fTextServiceCount;
    TextServiceInfo fServices[1];
};
typedef struct TextServiceList TextServiceList;
typedef TextServiceList * TextServiceListPtr;
typedef TextServiceListPtr * TextServiceListHandle;
```

Fields

`fTextServiceCount`
: An integer specifying the number of `TextServiceInfo` structures in the text service component list provided in the `fServices` field.

`fServices`
: A variable-length array of structures of type `TextServiceInfo` (page 34). Each `TextServiceInfo` structure identifies a specific component by name and `Component` value.

Discussion
A structure of type `TextServiceInfo` is used in the function `GetServiceList` (page 74) to list of all the text service components of a specified type that are currently available on a system.

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

Declared In
`TextServices.h`

TextServicePropertyValue

Defines a data type for text service property values.

```
typedef SInt32 TextServicePropertyValue;
```

Discussion
The property values associated with this data type are "Text Services Property Values" (page 60). Note that these values are declared as `CFStringRef` data types, so they require a cast to the `SInt32` data type before you can supply them as a `TextServicePropertyValue`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`TextServices.h`

TSMContext

A reference to an opaque object that specifies a TSM context.

```
typedef struct OpaqueTSMContext * TSMContext;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.2.

Declared In
`TextServices.h`

TSMDocumentID

A reference to an opaque object that identifies a specific TSM document.

```
typedef struct OpaqueTSMDocumentID * TSMDocumentID;
```

Discussion
Each time a client application calls the function `NewTSMDocument` (page 23), the Text Services Manager creates an opaque internal structure called a TSM Document and returns a pointer to the document's identification number.

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

Declared In
`TextServices.h`

TSMGlyphInfo

Describes one glyph embedded in a run of text.

```
struct TSMGlyphInfo {
    CFRange range;
    ATSFontRef fontRef;
    UInt16 collection;
    UInt16 glyphID;
};
typedef struct TSMGlyphInfo TSMGlyphInfo;
```

Fields

`range`
: A `CFRange` data structure that specifies, in UTF-16 offsets, a range within the text to which this `TSMGlyphInfo` data structure applies. I

`fontRef`
: An ATS font reference that specifies the font with which the glyph should be displayed. Note that the character collection ROS (Adobe Registry, Ordering, Supplement) is a property of the font.

`collection`

A glyph collection type that specifies how the `glyphID` parameter should be interpreted. When the value is `kGlyphCollectionID`, `glyphID` specifies the glyph's ID. When `collection` is a non-zero value, it specifies a character collection and `glyphID` specifies a CID. Note that `collection` must match the character collection of the font specified by the `fontRef` parameter.

When collections do not match, the `TSMGlyphInfo` data structure is invalid and should be ignored. You need to supply an ATSUI constant of type `GlyphCollection` to specify the character set you want to use. See *Inside Mac OS X: ATSUI Reference* for a list of the glyph collection constants you can specify.

`glyphID`

A glyph ID that specifies the glyph to use you to use in place of the current glyph. If you pass 0 instead of specifying a glyph, the `TSMGlyphInfo` data structure is used to attach a font to a range of text. In this case, the `fontRef` parameter specifies a font that should be used to display the range of text specified by the `range` parameter. This is useful when using characters in the Unicode private use area. Windings and other Windows based pi fonts are examples of such characters. When `glyphID` is zero, `collection` should also be zero and applications should ignore its value.

Discussion

The `TSMGlyphInfo` data structure is used as an item in the `TSMGlyphInfoArray` (page 37) data structure. You use these structures to provide TSM with glyph and font information when you want to override the current glyph or font.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`TextServices.h`

TSMGlyphInfoArray

Contains an array of glyph information structures.

```
struct TSMGlyphInfoArray {
    ItemCount numGlyphInfo;
    TSMGlyphInfo glyphInfo[1];
};
typedef struct TSMGlyphInfoArray TSMGlyphInfoArray;
```

Fields

`numGlyphInfo`

The number of items in the `glyphInfo` array.

`glyphInfo`

An array of glyph information structures.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`TextServices.h`

Constants

Attribute Bits for TSM Document Access Carbon Events

Represents TSM document attributes.

```
enum {
    kTSMDocAccessFontSizeAttributeBit = 0,
    kTSMDocAccessEffectiveRangeAttributeBit = 1
};
```

Constants

`kTSMDocAccessFontSizeAttributeBit`

When this bit is set, indicates to obtain font size information; used in the Carbon event `kEventTSMDocumentAccessGetFont`.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kTSMDocAccessEffectiveRangeAttributeBit`

When this bit is set, indicates to obtain effective range information used in the Carbon events `kEventTSMDocumentAccessGetFont` and `kEventTSMDocumentAccessGetGlyphInfo`.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

Discussion

These bit assignments are used for the TSM document access attribute parameters. You can use these bits to specify desired (optional) attributes in the `kEventParamTSMDocAccessRequestedCharacterAttributes` parameter available for the events `kEventTSMDocumentAccessGetFont` and `kEventTSMDocumentAccessGetGlyphInfo`.

Availability

Not available in CarbonLib.

Available in Mac OS X v 10.3 and later.

Attribute Masks for TSM Document Access Carbon Events

Used to set or test for document-access attributes.

```
typedef UInt32 TSMDocAccessAttributes;
enum {
    kTSMDocAccessFontSizeAttribute = 1L << kTSMDocAccessFontSizeAttributeBit,
    kTSMDocAccessEffectiveRangeAttribute = 1L <<
kTSMDocAccessEffectiveRangeAttributeBit
};
```

Constants

`kTSMDocAccessFontSizeAttribute`

Use to set or test for the `kTSMDocAccessFontSizeAttributeBit` bit.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kTSMDocAccessEffectiveRangeAttribute`
Use to set or test for the `kTSMDocAccessEffectiveRangeAttributeBit` bit.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.3 and later.

Carbon Event Class for TSM Document Access

Defines a constant for the Carbon event class used to allow TSM access to application documents content.

```
enum {
    kEventClassTSMDocumentAccess = 'tdac'
};
```

Constants
`kEventClassTSMDocumentAccess`
Used to request and deliver document content information. The events associated with this class provide text access, text attribute access, and transaction information. See "Carbon Events for TSM Document Access" (page 39) for a list of the events defined for this class.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

Discussion
The Text Services Manager (TSM) dispatches TSM document access events as Carbon events. You must install a Carbon event handler to access these events because they are not available through AppleEvent handlers.

Text Services dispatches these Carbon events through the function `SendTextInputEvent` (page 25).

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.3 and later.

Carbon Events for TSM Document Access

Define constants for the Carbon events associated with the TSM document access event class.

```
enum {
    kEventTSMDocumentAccessGetLength = 1,
    kEventTSMDocumentAccessGetSelectedRange = 2,
    kEventTSMDocumentAccessGetCharactersPtr = 3,
    kEventTSMDocumentAccessGetCharactersPtrForLargestBuffer = 4,
    kEventTSMDocumentAccessGetCharacters = 5,
    kEventTSMDocumentAccessGetFont = 6,
    kEventTSMDocumentAccessGetGlyphInfo = 7,
    kEventTSMDocumentAccessLockDocument = 8,
    kEventTSMDocumentAccessUnlockDocument = 9
};
```

Constants

`kEventTSMDocumentAccessGetLength`

Returns the number of 16-bit Unicode characters in the document.

You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the text service originating the event.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent` (page 25), called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessCharacterCount`. The size of the document in `UniChar` characters.

You can obtain the same information from this event as you can by calling the function `CFStringGetLength`, passing the document content formatted as a CFString.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventTSMDocumentAccessGetSelectedRange`

Returns the selection range in the document.

You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the text service originating the event. This can be `NULL` for input methods of the palette class, such as the typography panel.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent` (page 25), called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessReplyCharacterRange`. The selection range as a CFRange in `UniChar` characters. If the selection is empty, the range identifies the insertion point and the range specifies a length of 0.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventTSMDocumentAccessGetCharactersPtr`

Returns a Unicode pointer to the entire document content. Handle this event when your application has access to the entire document. If your application has access to a cache, use the event `kEventTSMDocumentAccessGetCharactersPtrForLargestBuffer`.

Some text engines may not support this event for reasons that are implementation-dependent. For example, a text engine backing store may consist of legacy encoding runs. It may also consist of unflattened Unicode, stored as a B-tree of text blocks. For such reasons, a text engine may reject a request for a pointer to a flattened Unicode buffer. Note that text access through this pointer is to be strictly read-only, so any changes to the document should be made through TSM text input events, such as `kEventTextInputUpdateActiveInputArea` or `kEventTextInputUnicodeText`. This pointer is valid only during a transaction surrounded by document lock/unlock events, or until an event causes the document to change, such as dispatching `kEventTextInputUpdateActiveInputArea` or `kEventTextInputUnicodeText` events, whichever occurs first.

You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the input method originating the event.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent` (page 25), called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessReplyCharactersPtr`. The `UniChar` pointer to the document.

You can obtain the same information from this event as you can by calling the function `CFStringGetCharactersPtr`, passing the document content formatted as a CFString.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventTSMDocumentAccessGetCharactersPtrForLargestBuffer`

Returns a Unicode pointer to a portion of the document. Handle this event when your application has access to a cache. If your application has access to the entire document, use the event `kEventTSMDocumentAccessGetCharactersPtr`.

Some text engines keep text in unflattened Unicode—for example, stored as a B-tree of text blocks. Sometimes, especially for chunks of text near the insertion point, the text engine caches a chunk of text to which it can readily provide a pointer. But because the text is not flattened, the text engine might reject a request for such a pointer. See the Discussion for more information.

Note that text access through this pointer is strictly read-only, so any changes to the document should be made through TSM text input events, such as `kEventTextInputUpdateActiveInputArea` or `kEventTextInputUnicodeText`. This pointer is valid only during a transaction surrounded by document lock/unlock, or until an event causes the document to change, such as dispatching `kEventTextInputUpdateActiveInputArea` or `kEventTextInputUnicodeText` events.

You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the input method originating the event.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent`, called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessSendCharacterIndex`. The location in the document for which the caller wants a pointer to a buffer of text that includes that location. This buffer could be available from a cache due to recent interaction near that location, such as the insertion point.
- `kEventParamTSMDocAccessReplyCharactersPtr`. The `UniChar` pointer to a portion of the document text.
- `kEventParamTSMDocAccessReplyCharacterRange`. A `CFRange` value for the text returned by the text pointer. The initial offset in the range is document-relative.

This event is similar to calling the function `CFStringGetCharactersPtr` on a portion of the document content formatted as a CFString, except that the substring is determined by the text engine.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventTSMDocumentAccessGetCharacters`

This fills a caller provided buffer with Unicode characters in the specified range. This event is equivalent to calling the function `CFStringGetCharacters` on the document content treated as a CFString.

You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the input method originating the event.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent`, called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessSendCharacterRange`. The range of text that should be copied into the buffer provided by the caller.
- `kEventParamTSMDocAccessSendCharactersPtr`. A buffer provided by the caller to contain the specified range of `UniChar` characters. This buffer is identical in usage to the one used in the function `CFStringGetCharacters`.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventTSMDocumentAccessGetFont`

Returns font, font size, and the range over which these attributes are constant. Where the font/font size attributes span multiple characters, an effective range (over which requested attributes are constant) is returned by the text engine.

You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the input method originating the event.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent`, called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessSendCharacterIndex`. The location in the document for which the caller would like font information.
- `kEventParamTSMDocAccessRequestedCharacterAttributes`. A `TSMDocAccessAttributes` bit field filled out with the desired attributes. Applicable values for this event are: `kTSMDocAccessFontSizeAttribute` which requests font size information through the `kEventParamTSMDocAccessReplyFontSize` parameter, and `kTSMDocAccessEffectiveRangeAttribute` which requests the text range over which font or font/size is constant.
- `kEventParamTSMDocAccessReplyATSFont`. The `ATSFontRef` for the location specified by the caller.
- `kEventParamTSMDocAccessReplyFontSize`. The font size for the requested location. This is an optional reply parameter. Return this information if `kTSMDocAccessFontSizeAttribute` is specified in the bit field passed as the `kEventParamTSMDocAccessRequestedCharacterAttributes` parameter.
- `kEventParamTSMDocAccessSendCharacterRange`. The maximum range of text the caller cares about. This is used to restrict the area of interest to the caller so the text engine doesn't process more characters than necessary in order to return an effective range.
- `kEventParamTSMDocAccessEffectiveRange`. The range of text over which both font and size are constant, within the bounds of the `kEventParamTSMDocAccessSendCharacterRange` parameter. This is an optional reply parameter. Return this information if `kTSMDocAccessEffectiveRangeAttribute` is specified in the bit field passed as the `kEventParamTSMDocAccessRequestedCharacterAttributes` parameter.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventTSMDocumentAccessGetGlyphInfo`

Returns glyph info and the range covered by that glyph. Where a glyph spans multiple characters, the effective range, represented by the glyph, is returned by the application.

You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the input method originating the event.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent`, called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessSendCharacterIndex`. The location in the document for which the caller would like glyph information.
- `kEventParamTSMDocAccessRequestedCharacterAttributes`. A `TSMDocAccessAttributes` bit field filled out with the information desired. The applicable value for this event is `kTSMDocAccessEffectiveRangeAttribute`, which requests the text range represented by a glyph.
- `kEventParamTSMDocAccessReplyATSUGlyphSelector`. The glyph used to display the range of text returned in the `kEventParamTSMDocAccessEffectiveRange` parameter. If the glyph used is the one that ATSUI would normally derive, this parameter can be omitted.
- `kEventParamTSMDocAccessEffectiveRange`. The range of text displayed as a glyph ID or CID. This is an optional reply parameter. Return this information if `kTSMDocAccessEffectiveRangeAttribute` is specified in the bit field passed as the `kEventParamTSMDocAccessRequestedCharacterAttributes` parameter.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventTSMDocumentAccessLockDocument`

Notifies the application that it should not change its document's text content (on its own) while a text service is involved in a transaction. The application should not allow changes, for example, by its secondary threads. This type of event defines how a text service can obtain access to a document in a way that ensures data integrity during its transaction. The event can be used to prevent the application from letting its secondary threads modify the document while a text service is busy servicing an event, such as a key event, or some user interaction with text-service-provided user interface such as a menu selection. Also, while the document is locked, a text service is free to request pointer access to the document's text content (if this is supported by the application's text engine.) These lock-related events should be implemented using a retention counting scheme. Most applications will not support this kind of threading, so implementation of these events in the text engine are optional. In most text engines, the implementation of these events should be trivial, that is, just maintain a simple semaphore. TSM itself will implicitly lock/unlock around normal entry points into a text service, such as when it delivers key events to an input method, but there may be times when document changes can be driven by an input method without TSM involvement, such as the Carbon events involved when the user interacts with some user interface. In this case, the input method must manage locking, if the application supports it. However, the logic in an input method should not depend on whether TSM is in the call chain or not, and TSM should not depend on whether an input method performs correctly. This is why the lock mechanism needs to be some kind of retention counting scheme instead of a simple on and off mechanism. Document lock support is optional on the part of the text engine (if it is not threaded). TSM implicitly locks/unlocks the document around delivery of events to input methods, if the application supports it.You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the input method originating the event.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent` (page 25), called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessLockCount`. The resulting retention count of locks on the document.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventTSMDocumentAccessUnlockDocument`

Unlock the document so the application text engine is free to initiate changes again. (See `kEventTSMDocumentAccessLockDocument`).

You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the input method originating the event.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent`, called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessLockCount`. The resulting retention count of locks on the document.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

Discussion

Text-access events are very similar in design to the CFString API. You can think of an entire document as a flattened Unicode string, and the events in this interface can access any portion of it. Just as the text is Unicode, the text offsets are also Unicode.

The event `kEventTSMDocumentAccessGetSelectedRange` allows a text service to obtain text near the insertion point (or selection), but access is by no means restricted to this vicinity. Use the event `kEventTSMDocumentAccessGetLength` to obtain the size of the document.

Supporting these events effectively provide hooks into the text engine, but it is understood that access to a document in this way is strictly read-only. Where direct access to document content cannot be provided through a pointer, the requested text can be copied instead. Situations where a pointer may not be available from the text engine include the following:

- The pointer requires conversion of text in Mac encodings to Unicode.
- The pointer requires sparse Unicode text blocks to be flattened into a single buffer.

The idea is to minimize copying and converting text encodings where possible. The text service typically begins by asking for a document pointer through the event `kEventTSMDocumentAccessGetCharactersPtr`. If this fails, it typically falls back to the event `kEventTSMDocumentAccessGetCharactersPtrForLargestBuffer`, specifying a location of interest. If this fails, it falls back to `kEventTSMDocumentAccessGetCharacters`, specifying a range of interest. Of course, when requesting small amounts of data with such a few characters on either side of the insertion point, there is no obligation to optimize in this way. It's valid to simply use `kEventTSMDocumentAccessGetCharacters`.

The text engine is entirely free to deny a request for a text pointer for these or any other implementation-specific reason.

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.3 and later.

Carbon Event Parameters for General TSM Events

Define general parameters for TSM events.

```
enum {
    kEventParamTSMSendRefCon      = 'tsrc',
    kEventParamTSMSendComponentInstance = 'tsci'
};
```

Constants

`kEventParamTSMSendRefCon`
: This parameter is equivalent to the text input parameter `kEventParamTextInputSendRefCon`; the parameter data type is `typeLongInteger`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`kEventParamTSMSendComponentInstance`
: This parameter is equivalent to the text input parameter `kEventParamTextInputSendComponentInstance`; the parameter data type is `typeComponentInstance`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

Availability
Not available in CarbonLib.

Available in Mac OS X v 10.3 and later.

Carbon Event Parameters for TSM Document Access

Define document access parameters for TSM events.

```
enum {
    kEventParamTSMDocAccessSendRefCon = kEventParamTSMSendRefCon,
    kEventParamTSMDocAccessSendComponentInstance =
kEventParamTSMSendComponentInstance,
    kEventParamTSMDocAccessCharacterCount = 'tdct',
    kEventParamTSMDocAccessReplyCharacterRange = 'tdrr',
    kEventParamTSMDocAccessReplyCharactersPtr = 'tdrp',
    kEventParamTSMDocAccessSendCharacterIndex = 'tdsi',
    kEventParamTSMDocAccessSendCharacterRange = 'tdsr',
    kEventParamTSMDocAccessSendCharactersPtr = 'tdsp',
    kEventParamTSMDocAccessRequestedCharacterAttributes = 'tdca',
    kEventParamTSMDocAccessReplyATSFont = 'tdaf',
    kEventParamTSMDocAccessReplyFontSize = 'tdrs',
    kEventParamTSMDocAccessEffectiveRange = 'tder',
    kEventParamTSMDocAccessReplyATSUGlyphSelector = 'tdrg',
    kEventParamTSMDocAccessLockCount = 'tdlc',
    typeATSFontRef                = 'atsf',
    typeGlyphSelector             = 'glfs'
};
```

Constants

`kEventParamTSMDocAccessSendRefCon`
: The TSM function `SendTextInputEvent` (page 25), called by an input method, inserts this parameter before dispatching the event to the user focus. The parameter data type is `typeLongInteger`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessSendComponentInstance`
: The parameter data type is `typeComponentInstance`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessCharacterCount`
: The parameter data type is `typeCFIndex`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessReplyCharacterRange`
: The parameter data type is `typeCFRange`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessReplyCharactersPtr`
: The parameter data type is `typePtr`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessSendCharacterIndex`
: The parameter data type is `typeCFIndex`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessSendCharacterRange`
: The parameter data type is `typeCFRange`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessSendCharactersPtr`
: The parameter data type is `typePtr`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessRequestedCharacterAttributes`
: The parameter data type is `typeUInt32`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessReplyATSFont`
: The parameter data type is t`ypeATSFontRef`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessReplyFontSize`
: The parameter data type is `typeFloat`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessEffectiveRange`
: The parameter data type is `typeRange`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessReplyATSUGlyphSelector`
: The parameter data type is `typeGlyphSelector`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessLockCount`
: The parameter data type is `typeCFIndex`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`typeATSFontRef`
: The parameter data type is `ATSFontRef`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`typeGlyphSelector`
: The parameter data type is `ATSUGlyphSelector`.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

Discussion

See "Carbon Events for TSM Document Access" (page 39) for more information on these parameters and the information they contain for a specific event.

Availability

Not available in CarbonLib.

Available in Mac OS X v 10.3 and later.

Component Flags

Specify flags used for input method components.

```
enum {
    bTakeActiveEvent = 15,
    bHandleAERecording = 16,
    bScriptMask = 0x00007F00,
    bLanguageMask = 0x000000FF,
    bScriptLanguageMask = bScriptMask + bLanguageMask
};
```

Constants

`bTakeActiveEvent`
: This bit is set if the component takes an active event,

  Available in Mac OS X v10.0 and later.

  Declared in `TextServices.h`.

`bHandleAERecording`
: This bit is set if the component takes care of recording Apple Events.

  Available beginning with version 2.0.

  Declared in `TextServices.h`.

`bScriptMask`
: Specifies bits 8 - 14.

  Available in Mac OS X v10.0 and later.

  Declared in `TextServices.h`.

`bLanguageMask`
: Specifies bits 0 - 7.

  Available in Mac OS X v10.0 and later.

  Declared in `TextServices.h`.

`bScriptLanguageMask`
: Specifies bits 0 - 14.

  Available in Mac OS X v10.0 and later.

  Declared in `TextServices.h`.

Document Property Tags

Specify property tags for a TSM document.

```
typedef OSType      TSMDocumentPropertyTag;
enum {

kTSMDocumentSupportGlyphInfoPropertyTag = 'dpgi',
kTSMDocumentUseFloatingWindowPropertyTag = 'uswm',
kTSMDocumentUnicodeInputWindowPropertyTag = 'dpub',
kTSMDocumentSupportDocumentAccessPropertyTag = 'dapy',
kTSMDocumentRefconPropertyTag = 'refc',
kTSMDocumentInputModePropertyTag = 'imim',
kTSMDocumentPropertySupportGlyphInfo =
        kTSMDocumentSupportGlyphInfoPropertyTag,
kTSMDocumentPropertyUnicodeInputWindow =
        kTSMDocumentUnicodeInputWindowPropertyTag,
kTSMDocumentTextServicePropertyTag = kTextServiceDocumentInterfaceType,
kTSMDocumentUnicodePropertyTag = kUnicodeDocumentInterfaceType,
kTSMDocumentTSMTEPropertyTag  = kTSMTEDocumentInterfaceType
};
```

Constants

`kTSMDocumentSupportGlyphInfoPropertyTag`
: The existence of this property in a TSM document indicates that the event handlers associated with he TSM document are aware of the TSM `GlyhInfo` data structure. This structure allows the input source producing text to apply Glyph IDs, CIDs, or fonts to subranges of text produced. This is useful or characters in Unicode private use area, such as Windings. For more information, see Technical Note TN2079 Glyph Access Protocol. By convention, this value can be a `UInt32` with a value of 0, but this is arbitrary. Available in TSM 1.5, in Mac OS X 10.2 and later.

  Available in Mac OS X v10.3 and later.

  Declared in `TextServices.h`.

`kTSMDocumentUseFloatingWindowPropertyTag`
: The presence of this property tag indicates that the TSM document should use the TSM floating input window to handle input from input methods. This form of input does not support Unicode input by default, unless the property `kTSMDocumentUnicodeInputWindowPropertyTag` is set.

  Available in Mac OS X v10.3 and later.

  Declared in `TextServices.h`.

`kTSMDocumentUnicodeInputWindowPropertyTag`
: The presence of this property tag indicates that although the TSM document has been told to use the TSM floating input window to handle input from input methods, the floating window is to support Unicode input. This is useful when non input-related activity is to produce Unicode, such as keyboard navigation.

  Available in Mac OS X v10.3 and later.

  Declared in `TextServices.h`.

`kTSMDocumentSupportDocumentAccessPropertyTag`

The presence of this property tag indicates that the event handlers associated with this TSM document support the TSM document access event suite (see "Carbon Events for TSM Document Access" (page 39).) This property also indicates that the handler for the TSMevent `kEventTextInputUpdateActiveInputArea` supports the `replaceRange` parameter and that the handler is a Carbon event handler, not an AppleEvent handler.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSMDocumentRefconPropertyTag`

The property value initially contains the `refcon` value passed to the function `NewTSMDocument`. This property is useful for changing the `refcon` value after the TSM document has been created. The `refcon` value is a `long`, the same as that passed to `NewTSMDocument`. Property is value-dependent; see the Discussion for more information.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSMDocumentInputModePropertyTag`

The property value indicates which input mode should be used by the current keyboard-class input method. It is useful for temporarily restricting text input to a subset of characters normally produced by an input method in a given script, such as Katakana for Japanese input. See "Text Service Properties" (page 59) for more details. Also note that this property tag and value are passed unchanged to the function "SetTextServiceProperty" (page 26), so it also serves as a text service property tag. See `kTextServiceInputModePropertyTag` for discussion on the values associated with this property.

The property value is a `CFStringRef` data type. With the function `TSMGetTextServiceProperty`, the behavior is that of a Copy function. The implementation of `SetTextServiceProperty` (in the component) retains or copies the CFString object. In either case the caller is responsible for releasing the reference. Property is value-dependent; see the Discussion for more information.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSMDocumentPropertySupportGlyphInfo`

You should no longer use this property.

Available in Mac OS X v10.2 and later.

Declared in `TextServices.h`.

`kTSMDocumentPropertyUnicodeInputWindow`

You should no longer use this property.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSMDocumentTextServicePropertyTag`

Specifies a non-Unicode savvy document. This property is equivalent to a pre-existing document interface type.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSMDocumentUnicodePropertyTag`

This property is equivalent to the Unicode document interface type.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSMDocumentTSMTEPropertyTag`

This property is equivalent to the TSMTE document interface type.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

Discussion

You can use the functions `TSMSetDocumentProperty` (page 31) and `TSMGetDocumentProperty` (page 29) to set and get arbitrary property data needed by your application.

Unless otherwise noted, all properties are read-only, value-independent, and available in TSM version 2.2, which is the version available starting in Mac OS X version 10.3.

Value-independent properties are used where the existence of the property, and not its value, is sufficient. These properties can read by other clients, and are most often used by input methods. For example, input methods can query the current TSM document to see if supports unrestricted Unicode input, or if it supports the `GlyphInfo` protocol.

Value-dependent properties are used when the value associated with a property is meaningful.

Input Method Identifier

Specifies a keyboard input method text service.

```
enum {
    kInputMethodService = kKeyboardInputMethodClass
};
```

Constants

`kInputMethodService`

A four-character code identifying an input method text service. Specifies that the older constant name `kInputMethodService` is equivalent to the newer constant name `kKeyboardInputMethodClass`.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

Input Mode Dictionary Key

Defines a string for the input mode dictionary key that you can use in the Component bundle info.plist.

```
#define kComponentBundleInputModeDictKey  CFSTR("ComponentInputModeDict")
```

Discussion

If you are developing an input method, you use this key in the component bundle info.plist to identify a dictionary of input mode information. The dictionary should contain keys to identify input modes—see "Individual Input Mode Keys" (page 54)—and should have the form described in the Discussion section of the function `CopyTextServiceInputModeList` (page 14). The function `CopyTextServiceInputModeList` returns an input mode dictionary.

Input Mode Palette Menu Definition Keys

Defines keys used to describe the items in a pull-down menu.

```
#define kTSInputModePaletteItemTitleKey   CFSTR("tsInputModePaletteItemTitleKey")
#define kTSInputModePaletteItemKeyEquivalentKey
CFSTR("tsInputModePaletteItemKeyEquivalentKey")
#define kTSInputModePaletteItemKeyEquivalentModifiersKey
CFSTR("tsInputModePaletteItemKeyEquivalentModifiersKey")
```

Constants

`kTSInputModePaletteItemTitleKey`
A CFString that specifies a menu item title. Use - for a separator.

`kTSInputModePaletteItemKeyEquivalentKey`
A CFString that specifies a menu item keyboard shortcut .

`kTSInputModePaletteItemKeyEquivalentModifiersKey`
A CFNumber that specifies a menu item keyboard shortcut modifier (from Events.h).

Discussion

These keys are returned by the function `GetInputModePaletteMenu` (page 72), in the `outMenuItemsArray` parameter. For information on the structure of the CFDictionary, see the TextServices.h header file.

Input Mode Palette Control Keys

Defines keys used to describe controls for an input palette.

```
#define kTSInputModePaletteItemTypeKey  CFSTR("tsInputModePaletteItemTypeKey")
#define kTSInputModePaletteItemIconKey   CFSTR("tsInputModePaletteItemIconKey")
#define kTSInputModePaletteItemAltIconKey   CFSTR("tsInputModePaletteItemAltIconKey")
#define kTSInputModePaletteItemStateKey   CFSTR("tsInputModePaletteItemStateKey")
#define kTSInputModePaletteItemEnabledKey   CFSTR("tsInputModePaletteItemEnabledKey")
#define kTSInputModePaletteItemIDKey    CFSTR("tsInputModePaletteItemIDKey")
```

Constants

`kTSInputModePaletteItemTypeKey`
A CFNumber that specifies the type of control (0: push button, 1: toggle button, 2: pull-down menu),

`kTSInputModePaletteItemIconKey`
A CFString that specifies an icon file name. The file should be located in the input method bundle resource directory, so this is just the file name, not full path.

`kTSInputModePaletteItemAltIconKey`
A CFString that specifies an alternate icon file name. The file should be located in the input method bundle resource directory, so this is just the file name, not full path.

`kTSInputModePaletteItemStateKey`
A CFNumber that specifies the state of the control (0: clear or unpressed, 1: checked or pressed, 2: mixed).

`kTSInputModePaletteItemEnabledKey`
A CFBoolean that specifies the enabled state of the control.

`kTSInputModePaletteItemIDKey`
A CFNumber that specifies a `UInt32` tag ID for the control.

Discussion

You use these keys in a CFDictionary that contains control descriptions passed to the functions `TSMInputModePaletteLoadButtons` (page 82) and `TSMInputModePaletteUpdateButtons` (page 83). For information on the structure of the CFDictionary, see the TextServices.h header file.

Individual Input Mode Keys

Defines keys used to identify input modes in an input mode dictionary.

```
#define kTSInputModeListKey CFSTR("tsInputModeListKey")
#define kTSInputModeMenuIconFileKey CFSTR("tsInputModeMenuIconFileKey")
#define kTSInputModeAlternateMenuIconFileKey
        CFSTR("tsInputModeAlternateMenuIconFileKey")
#define kTSInputModePaletteIconFileKey
        CFSTR("tsInputModePaletteIconFileKey")
#define kTSInputModeDefaultStateKey CFSTR("tsInputModeDefaultStateKey")
#define kTSInputModeScriptKey CFSTR("tsInputModeScriptKey")
#define kTSInputModePrimaryInScriptKey
        CFSTR("tsInputModePrimaryInScriptKey")
#define kTSInputModeIsVisibleKey      CFSTR("tsInputModeIsVisibleKey")
#define kTSInputModeKeyEquivalentModifiersKey
        CFSTR("tsInputModeKeyEquivalentModifiersKey")
#define kTSInputModeKeyEquivalentKey
            CFSTR("tsInputModeKeyEquivalentKey")
#define kTSInputModeJISKeyboardShortcutKey
        CFSTR("tsInputModeJISKeyboardShortcutKey")
```

Discussion
If you are developing an input method, you use these keys in a dictionary of input mode information. The Component bundle info.plist should an input mode dictionary key that identifies the dictionary—see "Input Mode Dictionary Key" (page 52). The dictionary should have the form described in the Discussion section of the function `CopyTextServiceInputModeList` (page 14). The function `CopyTextServiceInputModeList` returns an input mode dictionary.

Interfaces

Specify types of text services interfaces.

```
enum {
    kTextService = 'tsvc'
};
```

Constants
`kTextService`
A four-character code identifying a text service of any kind (including input methods). This value is also used to identify non-Unicode TSM documents.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

Discussion
This constant is used in arrays of type `InterfaceTypeList` (page 33). In addition to this constants, the constant `kTSMTEInterfaceType` (`'tmTE'`), from TSMTE.h, is a supported interface type that allows TextEdit to provide automatic inline input support in TextEdit documents.

Language and Script Constants

Specify the language or script is not known or neutral.

```
enum {
    kUnknownLanguage = 0xFFFF,
    kUnknownScript = 0xFFFF,
    kNeutralScript = 0xFFFF
};
```

Constants

`kUnknownLanguage`

Specifies an unknown language.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kUnknownScript`

Specifies an unknown script.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kNeutralScript`

Specifies a neutral script.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

Low-level Routine Selectors

Specify low level routines which are dispatched directly to the Component Manager.

```
enum {
    kCMGetScriptLangSupport = 0x0001,
    kCMInitiateTextService = 0x0002,
    kCMTerminateTextService = 0x0003,
    kCMActivateTextService = 0x0004,
    kCMDeactivateTextService = 0x0005,
    kCMTextServiceEvent = 0x0006,
    kCMGetTextServiceMenu = 0x0007,
    kCMTextServiceMenuSelect = 0x0008,
    kCMFixTextService = 0x0009,
    kCMSetTextServiceCursor = 0x000A,
    kCMHidePaletteWindows = 0x000B,
    kCMGetTextServiceProperty = 0x000C,
    kCMSetTextServiceProperty = 0x000D
};
```

Constants

`kCMGetScriptLangSupport`

Specifies the function `GetScriptLanguageSupport` (page 72); Component Manager call selector 1.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kCMInitiateTextService`

Specifies the function `InitiateTextService` (page 22); Component Manager call selector 2.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kCMTerminateTextService`
: Specifies the function `TerminateTextService` (page 27); Component Manager call selector 3.

  Available in Mac OS X v10.0 and later.

  Declared in `TextServices.h`.

`kCMActivateTextService`
: Specifies the function `ActivateTextService` (page 13); Component Manager call selector 4.

  Available in Mac OS X v10.0 and later.

  Declared in `TextServices.h`.

`kCMDeactivateTextService`
: Specifies the function `DeactivateTextService` (page 16); Component Manager call selector 5.

  Available in Mac OS X v10.0 and later.

  Declared in `TextServices.h`.

`kCMTextServiceEvent`
: Specifies the function `TextServiceEventRef` (page 27); Component Manager call selector 6.

  Available in Mac OS X v10.0 and later.

  Declared in `TextServices.h`.

`kCMGetTextServiceMenu`
: Specifies the function `GetTextServiceMenu` (page 20); Component Manager call selector 7.

  Available in Mac OS X v10.0 and later.

  Declared in `TextServices.h`.

`kCMTextServiceMenuSelect`
: Component Manager call selector 8.

  Available in Mac OS X v10.0 and later.

  Declared in `TextServices.h`.

`kCMFixTextService`
: Specifies the function `FixTextService` (page 18); Component Manager call selector 9.

  Available in Mac OS X v10.0 and later.

  Declared in `TextServices.h`.

`kCMSetTextServiceCursor`
: Component Manager call selector 10.

  Available in Mac OS X v10.0 and later.

  Declared in `TextServices.h`.

`kCMHidePaletteWindows`
: Specifies the function `HidePaletteWindows` (page 21); Component Manager call selector 11.

  Available in Mac OS X v10.0 and later.

  Declared in `TextServices.h`.

`kCMGetTextServiceProperty`
: Specifies the function `GetTextServiceProperty` (page 20); Component Manager call selector 12.

  Available in Mac OS X v10.0 and later.

  Declared in `TextServices.h`.

`kCMSetTextServiceProperty`
: Specifies the function `SetTextServiceProperty` (page 26); Component Manager call selector 13.

  Available in Mac OS X v10.0 and later.

  Declared in `TextServices.h`.

New Low-level Routine Selector

Specifies new low-level routine that are dispatched directly to the Component Manager.

```
enum {
    kCMUCTextServiceEvent = 0x000E
};
```

Constants

`kCMUCTextServiceEvent`
: Component Manager call selector 14.

  Available in Mac OS X v10.0 and later.

  Declared in `TextServices.h`.

Text Service Classes

Specify text service classes supported by TSM.

```
enum {
    kKeyboardInputMethodClass = 'inpm',
    kInkInputMethodClass = 'ink ',
    kCharacterPaletteInputMethodClass = 'cplt',
    kSpeechInputMethodClass = 'voic',
    kOCRInputMethodClass = 'ocr '
};
typedef OSType TextServiceClass;
```

Constants

`kKeyboardInputMethodClass`
: Specifies a text service class for keyboard input methods. Behavior is exclusive. Input methods in this class are normally associated with a Mac ScriptCode or Unicode, although they can be associated with several scripts by adopting the input mode protocol.

  Available in Mac OS X v10.2 and later.

  Declared in `TextServices.h`.

`kInkInputMethodClass`
: Specifies a text service class for Ink input methods. Behavior is additive. Text services in the Ink class do not belong to any given script in the sense that those of the Keyboard class do. Once selected, this kind of text service remains active regardless of the current keyboard script. Although text services in this class are keyboard script agnostic, similar to input methods of the keyboard class they can still profess to produce only those Unicodes that are encoded in the Mac encoding specified in their component description record or their implementation of the `GetScriptLanguageSupport` component call.

  Available in Mac OS X v10.2 and later.

  Declared in `TextServices.h`.

`kCharacterPaletteInputMethodClass`
: Specifies a text service class for Character Palette input methods. Behavior is additive. Text services in the character palette class do not belong to any given script in the same sense that do those of the keyboard class. Once selected, this kind of text service remains active regardless of the current keyboard script. Although text services in this class are keyboard script agnostic, similar to input methods of the keyboard class, they can still produce only those Unicodes that are encoded in the Mac encoding specified in their component description record or their implementation of the `GetScriptLanguageSupport` component call. Unlike input methods in the keyboard class, multiple such text services can be activate in parallel. Mac OS X provides a System user interface to allow the user to both enable and select multiple such input methods.

    Available in Mac OS X v10.2 and later.

    Declared in `TextServices.h`.

`kSpeechInputMethodClass`
: Specifies a text service class for Speech input methods. Behavior is additive. Similar to Character palette class.

    Available in Mac OS X v10.3 and later.

    Declared in `TextServices.h`.

`kOCRInputMethodClass`
: Specifies a text service class for Optical Character Recognition input methods. Behavior is additive. Similar to Character palette class.

    Available in Mac OS X v10.3 and later.

    Declared in `TextServices.h`.

Discussion

Text service classes fall in two categories or behaviors. Text services that belong to some classes are exclusive of one another within a given Mac script code, such input methods of the keyboard class. Input Methods of other classes are additive in nature, regardless of the current keyboard script.

Within a given class and script, exclusive input methods can only be activated one at a time. Input methods in additive classes are keyboard script agnostic and can be active in parallel with other text services in the same class, such as multiple character palettes.

These are the same as the component subtype for the component description.

Text Service Version

Specifies the interface type for version 2.

```
enum {
    kTextServiceVersion2 = 'tsv2'
};
```

Constants

`kTextServiceVersion2`
: The interface type for V2 interfaces

    Available in Mac OS X v10.0 through Mac OS X v10.2.

    Declared in `TextServices.h`.

Text Service Properties

Specify a feature or functionality of a component.

```
typedef OSType TextServicePropertyTag;
enum {
    kTextServiceInputModePropertyTag = kTSMDocumentInputModePropertyTag,
    kIMJaTypingMethodRoman = 'roma',
    kIMJaTypingMethodKana = 'kana',
    kIMJaTypingMethodProperty = kTextServiceJaTypingMethodPropertyTag,
    kTextServiceJaTypingMethodPropertyTag = 'jtyp'
};
```

Constants

`kTextServiceInputModePropertyTag`

Specifies the input mode property for input methods. This property is a CFString object that uniquely identifies which input mode should be made current by a keyboard class input method, if possible. This property tag is identical to the tag `kTSMDocumentInputModePropertyTag` passed to the function `TSMDocumentProperty`. This allows the tag and value to be passed through without interpretation.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kIMJaTypingMethodRoman`

Not recommended. Specify Japanese input in Roman script.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kIMJaTypingMethodKana`

Not recommended. Specify Japanese input in Kana script.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kIMJaTypingMethodProperty`

Deprecated. Specify the typing method as Japanese input. This property is deprecated. Use the tag `kTextServiceInputModePropertyTag` instead.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kTextServiceJaTypingMethodPropertyTag`

Deprecated.Use the tag `kTextServiceInputModePropertyTag` instead.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

Discussion

Text Service Property constants are used in `SetTextServiceProperty` (page 26) and `GetTextServiceProperty` (page 20). The only property that is recommend for you to use is the property `kTextServiceInputModePropertyTag`.

Input modes are either generic (pre-defined by TSM), or specific to an input method. An example of a generic input mode is Katakana input (Japanese) where input in a text field needs to be restricted to that character subset. Another is Roman input mode. This is useful to temporarily provide Roman input from an input method that normally allows text input in another script. The advantage to using Roman input mode over forcing the keyboard script to Roman is that the same user interface for the input method continues to be available to the user, even though the input script changed. An example of a special input mode (input method specific) is Hanin input mode in Traditional Chinese input methods.

To temporarily change the current input mode from whatever it is to a generic one, use the function `GetTextServiceProperty` to obtain the current input mode, then call the function `SetTextServiceProperty` to switch to the generic mode. When done using the generic input mode, you can restore the original input mode.

You can find out what input modes are supported by an input method by calling the function `CopyTextServiceInputModeList`. If the input method does not support a specified input mode, the functions `GetTextServiceProperty` and `SetTextServiceProperty` return the result `tsmComponentPropertyUnsupportedErr`. The function `GetTextServiceProperty` returns the result `tsmComponentPropertyNotFoundErr`.

Text Services Property Values

Define values for the text services input mode property tags.

```
#define kTextServiceInputModeRoman     CFSTR("com.apple.inputmethod.Roman")
#define kTextServiceInputModePassword
        CFSTR("com.apple.inputmethod.Password")
#define kTextServiceInputModeJapaneseHiragana
    CFSTR("com.apple.inputmethod.Japanese.Hiragana")
#define kTextServiceInputModeJapaneseKatakana
    CFSTR("com.apple.inputmethod.Japanese.Katakana")
#define kTextServiceInputModeJapaneseFullWidthRoman
    CFSTR("com.apple.inputmethod.Japanese.FullWidthRoman")
#define kTextServiceInputModeJapaneseHalfWidthKana
    CFSTR("com.apple.inputmethod.Japanese.HalfWidthKana")
#define kTextServiceInputModeJapanesePlaceName
    CFSTR("com.apple.inputmethod.Japanese.PlaceName")
#define kTextServiceInputModeJapaneseFirstName
    CFSTR("com.apple.inputmethod.Japanese.FirstName")
#define kTextServiceInputModeJapaneseLastName
    CFSTR("com.apple.inputmethod.Japanese.LastName")
#define kTextServiceInputModeBopomofo
    CFSTR("com.apple.inputmethod.TradChinese.Bopomofo")
#define kTextServiceInputModeTradChinesePlaceName
    CFSTR("com.apple.inputmethod.TradChinese.PlaceName")
#define kTextServiceInputModeHangul
    CFSTR("com.apple.inputmethod.Korean.Hangul")
#define kTextServiceInputModeJapanese
    CFSTR("com.apple.inputmethod.Japanese")
#define kTextServiceInputModeTradChinese
    CFSTR("com.apple.inputmethod.TradChinese")
#define kTextServiceInputModeSimpChinese
    CFSTR("com.apple.inputmethod.SimpChinese")
#define kTextServiceInputModeKorean
    CFSTR("com.apple.inputmethod.Korean")
```

Constants

`kTextServiceInputModeRoman`

Specifies to restrict output to Roman characters only.

`kTextServiceInputModePassword`
`kTextServiceInputModeJapaneseHiragana`
Specifies to restrict output to Hiragana characters only (no conversion to Kanji, that is, yomi).

`kTextServiceInputModeJapaneseKatakana`
Specifies to restrict output to Katakana characters only (no conversion to Kanji).

`kTextServiceInputModeJapaneseFullWidthRoman`
`kTextServiceInputModeJapaneseHalfWidthKana`
`kTextServiceInputModeJapanesePlaceName`
`kTextServiceInputModeJapaneseFirstName`
`kTextServiceInputModeJapaneseLastName`
`kTextServiceInputModeBopomofo`
Specifies to restrict output to Bopomofo characters only (no conversion to Han).

`kTextServiceInputModeTradChinesePlaceName`
Specifies to restrict output to traditional chines place name.

`kTextServiceInputModeHangul`
Specifies to restrict output to Hangul syllables only (no conversion to Hanja).

`kTextServiceInputModeJapanese`
Specifies unrestricted Japanese output.

`kTextServiceInputModeTradChinese`
Specifies traditional Chinese generic (unrestricted) input mode.

`kTextServiceInputModeSimpChinese`
Specifies simplified Chinese generic (unrestricted) input mode.

`kTextServiceInputModeKorean`
Specifies Korean generic (unrestricted) output (Hanja possible).

Discussion
These values require a cast from the `CFStringRef` data type to an `SInt32` data type before they can be used in text services functions.

Text Services Object Attributes

Specify characteristics of text services.

```
#define kKeyboardInputMethodTypeName         "\pkeyboardinputmethod"
#define kHandwritingInputMethodTypeName      "\phandwritinginputmethod"
#define kSpeechInputMethodTypeName           "\pspeechinputmethod"
#define kTokenizeServiceTypeName             "\ptokenizetextservice"
#define kInteractiveTextServiceTypeName      "\pinteractivetextservice"
#define kInputMethodModeName                 "\pinputmethodmode"
#define kInputMethodModeVariantName          "\pinputmethodvariantmode"
#define kTextServiceModeName                 "\ptextservicemode"
#define kTextServiceNeedsInlineAppMode       "\ptextservicesneedsinlineapp"
#define kTextServiceNeedsGetProtocolMode "\ptextservicesneedsGetProtocol"
#define kTextServiceAnyAppMode               "\ptextservicesanyapp"
```

TSM Document Interfaces

Specify types of TSM document interfaces.

```
enum {
    kTextServiceDocumentInterfaceType = kTextService,
    kTSMTEDocumentInterfaceType   = 'tmTE',
    kUnicodeDocumentInterfaceType = 'udoc',
    };
```

Constants

`kTextServiceDocumentInterfaceType`

A four-character code identifying a TSM document type for traditional (non-Unicode) TSM documents. This is the traditional TSM document type. It does not support Unicode. TSM converts all Unicode produced by input methods to the Mac encoding represented by the current keyboard script (or the Mac encoding specified by the input method producing text.) Full Unicode input sources may not be selectable when this TSM document is active.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSMTEDocumentInterfaceType`

Deprecated. Specifies a TSM document type for TSMTE document (see `kTSMTEInterfaceType` in TSMTE.h). This requests automatic management of inline input sessions by TextEdit (the text engine.) See *Technote TE27 - Inline Input for TextEdit with TSMTE*. This document interface type should no longer be used because TextEdit has been replaced by MLTE.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kUnicodeDocumentInterfaceType`

Specifies a TSM document type for Unicode-savvy applications. TSM pass through all Unicode text unchanged. When this TSM document is active, the full range of input sources is available to the user, such as Unicode keyboard layouts.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

Discussion

These constants are used in arrays of type `InterfaceTypeList` (page 33). TSM Interface types, as of Mac OS X 10.3, are also stored as TSM document properties, so once a TSM document is created, you can easily find out its interface types at document creation.

Unicode Identifiers

Specify constants that identify Unicode components and documents.

```
enum {
    kUnicodeDocument = 'udoc',
    kUnicodeTextService = 'utsv'
};
```

Constants

`kUnicodeDocument`

A four-character code that identifies a Unicode TSM document, for use by Unicode-savvy applications.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kUnicodeTextService`
: Specifies a component type for a Unicode text service.

  Available in Mac OS X v10.0 and later.

  Declared in `TextServices.h`.

Collection Tags

Specify collection tags.

```
enum {
    kInteractiveServicesTag = 'tmin',
    kLocaleIDTag = 'loce',
    kTextInputObjectTag = 'tiot',
    kLocaleObjectRefTag = 'lobj',
    kLocaleRefTag = 'lref',
    kKeyboardInputMethodContextTag = 'kinp',
    kKeyboardLocaleObjectRefTag = 'kilo',
    kHandwritingInputMethodContextTag = 'hinp',
    kHandwritingLocaleObjectRefTag = 'hilo',
    kSpeechInputMethodContextTag = 'sinp',
    kSpeechLocaleObjectRefTag = 'silo',
    kPasswordModeTag = 'pwdm',
    kRefconTag = 'refc',
    kUseFloatingWindowTag = 'uswm',
    kReadOnlyDocumentTag = 'isro',
    kSupportsMultiInlineHolesTag = 'minl',
    kProtocolVersionTag = 'nprt',
    kTSMContextCollectionTag = 'tsmx'
};
```

Constants

`kUseFloatingWindowTag`
: Specifies the use of a bottom-line floating window for an input method.

  Available in Mac OS X v10.0 through Mac OS X v10.2.

  Declared in `TextServices.h`.

Discussion

All the constants in this enumeration, except `kUseFloatingWindowTag`, are reserved for future use.

Input Mode Variants

Specify variant tags for input modes.

```
enum {
    kIM2ByteInputMode = '2byt',
    kIM1ByteInputMode = '1byt',
    kIMDirectInputMode = 'dinp'
};
```

Constants

`kIM2ByteInputMode`

Specifies a double-byte input mode.

Available in Mac OS X v10.0 through Mac OS X v10.2.

Declared in `TextServices.h`.

`kIM1ByteInputMode`

Specifies a single-byte input mode.

Available in Mac OS X v10.0 through Mac OS X v10.2.

Declared in `TextServices.h`.

`kIMDirectInputMode`

Specifies a direct input mode.

Available in Mac OS X v10.0 through Mac OS X v10.2.

Declared in `TextServices.h`.

Discussion

These constants are reserved for future use.

Input Mode - Standard Tags

Specify standard tags for input method modes.

```
enum {
    kIMRomanInputMode = 'romn',
    kIMPasswordInputMode = 'pasw',
    kIMXingInputMode = 'xing',
    kIMHuaInputMode = 'huam',
    kIMPinyinInputMode = 'piny',
    kIMQuweiInputMode = 'quwe',
    kIMCangjieInputMode = 'cgji',
    kIMJianyiInputMode = 'jnyi',
    kIMZhuyinInputMode = 'zhuy',
    kIMB5CodeInputMode = 'b5cd',
    kIMKatakanaInputMode = 'kata',
    kIMHiraganaInputMode = 'hira'
};
```

Discussion

These constants are reserved for future use.

Locale Object Attributes

Specify attributes of a locale object.

```
enum {
    kNeedsInputWindow = 1,
    kHandlesUpdateRegion = 2,
    kHandlesGetRegion = 3,
    kHandlesPos2Offset = 4,
    kHandlesOffset2Pos = 5,
    kInPasswordMode = 6,
    kHandleMultipleHoles = 7,
    kDocumentIsReadOnly = 8
};
```

Discussion
These constants are reserved for future use.

Version Constants

Specify versions of the Text Services Manager.

```
enum {
    kTSMVersion = 0x0150,
    kTSM15Version = kTSMVersion,
    kTSM20Version = 0x0200,
    kTSM22Version = 0x0220,
    kTSM23Version = 0x0230
};
```

Constants

`kTSMVersion`

Specifies the version of the Text Services Manager is 1.5

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kTSM15Version`

Specifies the version of the Text Services Manager is 1.5

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSM20Version`

Specifies the version of the Text Services Manager is 2.0 (Mac OS X v10.0).

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSM22Version`

Specifies the version of the Text Services Manager is 2.2 (Mac OS X 1v0.3).

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSM23Version`

Specifies the version of the Text Services Manager is 2.3 (Mac OS X v10.4).

Available in Mac OS X v10.4 and later.

Declared in `TextServices.h`.

Result Codes

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

Result Code	Value	Description
`tsmComponentNoErr`	0	Component result: no error Available in Mac OS X v10.0 and later.
`tsmUnsupScriptLanguageErr`	-2500	Specified script and language are not supported Available in Mac OS X v10.0 and later.
`tsmInputMethodNotFoundErr`	-2501	Specified input method cannot be found Available in Mac OS X v10.0 and later.
`tsmNotAnAppErr`	-2502	The caller was not an application Available in Mac OS X v10.0 and later.
`tsmAlreadyRegisteredErr`	-2503	The caller is already TSM-initialized Available in Mac OS X v10.0 and later.
`tsmNeverRegisteredErr`	-2504	The caller is not TSM-aware Available in Mac OS X v10.0 and later.
`tsmInvalidDocIDErr`	-2505	Invalid TSM document ID Available in Mac OS X v10.0 and later.
`tsmTSMDocBusyErr`	-2506	Document is still active Available in Mac OS X v10.0 and later.
`tsmDocNotActiveErr`	-2507	Document is not active Available in Mac OS X v10.0 and later.
`tsmNoOpenTSErr`	-2508	There is no open text service component Available in Mac OS X v10.0 and later.
`tsmCantOpenComponentErr`	-2509	Can’t open the component Available in Mac OS X v10.0 and later.
`tsmTextServiceNotFoundErr`	-2510	No text service component found Available in Mac OS X v10.0 and later.
`tsmDocumentOpenErr`	-2511	There are open documents Available in Mac OS X v10.0 and later.
`tsmUseInputWindowErr`	-2512	An input window is being used Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`tsmTSHasNoMenuErr`	-2513	The text service component has no menu Available in Mac OS X v10.0 and later.
`tsmTSNotOpenErr`	-2514	Text service component is not open Available in Mac OS X v10.0 and later.
`tsmComponentAlreadyOpenErr`	-2515	Text service component already open for document Available in Mac OS X v10.0 and later.
`tsmInputMethodIsOldErr`	-2516	The default input method is old-style Available in Mac OS X v10.0 and later.
`tsmScriptHasNoIMErr`	-2517	Script has no (or old) input method Available in Mac OS X v10.0 and later.
`tsmUnsupportedTypeErr`	-2518	Unsupported interface type Available in Mac OS X v10.0 and later.
`tsmUnknownErr`	-2519	Any other error not listed in this table Available in Mac OS X v10.0 and later.
`tsmDefaultIsNotInputMethodErr`	-2524	Current input source is a keyboard layout resource Available in Mac OS X v10.0 and later.
`tsmDocPropertyNotFoundErr`	-2528	Requested TSM document property not found Available in Mac OS X v10.2 and later.
`tsmDocPropertyBufferTooSmallErr`	-2529	Buffer passed for property value is too small Available in Mac OS X v10.2 and later.
`tsmCantChangeForcedClassStateErr`	-2530	Enabled state of a `TextService` class has been forced and cannot be changed Available in Mac OS X v10.2 and later.

Deprecated Text Services Manager Functions

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

Deprecated in Mac OS X v10.5

CloseTextService

Closes a text service component, other than an input method, and disassociates it from the active TSM document. (Deprecated in Mac OS X v10.5.)

```
OSErr CloseTextService (
   TSMDocumentID idocID,
   ComponentInstance aComponentInstance
);
```

Parameters

idocID

The identification number of a TSM document created by a prior call to the `NewTSMDocument` (page 23) function.

aComponentInstance

The component instance created by a prior call to `OpenTextService` (page 76).

Return Value

A result code. See "Text Services Manager Result Codes" (page 66).

Discussion

When a user wants to close an opened text service component, your client application should call the function `CloseTextService`.

If the text service component displays a menu, the Text Services Manager removes the menu from the menu bar. This function is for closing text service components other than input methods. Your application does not need to open or close input methods.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`TextServices.h`

DeselectTextService

Notifies TSM that an input method has been closed. (Deprecated in Mac OS X v10.5.)

```
OSStatus DeselectTextService (
   Component aComp
);
```

Parameters

aComp

Return Value

A result code. See "Text Services Manager Result Codes" (page 66).

Discussion

This API is currently only intended for use by Character Palette class input methods. It allows such an input method to notify TSM that it has been closed by the user as a result of interaction with the input method's own UI, such a palette's close button, instead of via the normal UI provided by the System, such as the Keyboard Menu.

Availability

Not available in CarbonLib.

Available in Mac OS X v 10.2 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`TextServices.h`

GetDefaultInputMethod

Obtains the default input method text service component for a given script and language. (Deprecated in Mac OS X v10.5.)

Not recommended.

```
OSErr GetDefaultInputMethod (
   Component *ts,
   ScriptLanguageRecord *slRecordPtr
);
```

Parameters

ts
: A pointer to the component identifier of the input method text service component that is associated with the script and language combination given in the `slRecord` parameter.

slRecordPtr
: A pointer to a structure of type `ScriptLanguageRecord` (page 33). This structure describes the script and language combination that is associated with the input method text service specified in the `ts` parameter.

Return Value

A result code. See "Text Services Manager Result Codes" (page 66).

Discussion

You should use the function `GetDefaultInputMethodOfClass` (page 71) instead of this one.

The operating system uses `GetDefaultInputMethod` to find out which input method to activate when the user selects a new keyboard script from the Keyboard menu or by Command-key combination, or when an application calls `KeyScript` to change keyboard scripts. Your application should not typically need to call this function.

Version Notes

For systems prior to Mac OS X, in versions of Japanese system software starting with KanjiTalk 7, if the default input method is pre-KanjiTalk 7 and non-TSM-aware, `GetDefaultInputMethod` returns the error `tsmInputMethodIsOldErr`. In that case the `ts` parameter contains the script code of the old input method in its high-order word, and the reference ID of the old input method in its low-order word.

Availability

Not recommended. Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`TextServices.h`

GetDefaultInputMethodOfClass

Obtains the default input method text service component for a given text service class. (Deprecated in Mac OS X v10.5.)

```
OSStatus GetDefaultInputMethodOfClass (
   Component *aComp,
   ScriptLanguageRecord *slRecPtr,
   TextServiceClass tsClass
);
```

Parameters

aComp

On return, a pointer to the component identifier of the input method text service component that is associated with the script and language combination given in the `slRecord` parameter.

slRecPtr

On return, a pointer to a structure of type `ScriptLanguageRecord` (page 33). This structure describes the script and language combination that is associated with the input method text service specified in the `ts` parameter.

tsClass

The text service class whose component and script language record you want to obtain. Pass `kKeyboardInputMethodClass` to specify a keyboard input method. Pass `kInkInputMethod` to specify an Ink input method.

Return Value

A result code. See "Text Services Manager Result Codes" (page 66).

Discussion

Availability

Not available in CarbonLib.

Available in Mac OS X v 10.2 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`TextServices.h`

GetInputModePaletteMenu

Obtains from an input method the menu to display for a pull-down menu on the input mode palette. (Deprecated in Mac OS X v10.5.)

```
ComponentResult GetInputModePaletteMenu (
   ComponentInstance inInstance,
   UInt32 inItemID,
   CFArrayRef *outMenuItemsArray
);
```

Parameters

`inInstance`
The component instance.

`inItemID`
The item ID of the pull-down menu button.

`outMenuItemsArray`
On return, points to an array of menu items. A pull-down menu consists of an array of CFDictionary objects that contain the keys described in `Input Mode Palette Menu Definition Keys` (page 52).

Return Value
Returns a non-null value on successful handling of the call.

Availability
Available in Mac OS X v 10.4 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`TextServices.h`

GetScriptLanguageSupport

Notifies a text service component that it must produce a list of its supported languages and scripts. (Deprecated in Mac OS X v10.5.)

```
ComponentResult GetScriptLanguageSupport (
   ComponentInstance ts,
   ScriptLanguageSupportHandle *scriptHdl
);
```

Parameters

ts

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 76).

scriptHdl

A handle to a structure of type `ScriptLanguageSupport` (page 34). The handle must be either `NULL` or a valid handle. If it is `NULL`, the text service component allocates a new handle. If it is already a valid handle, the text service component resizes it as necessary. `GetScriptLanguageSupport` should produce a list of scripts and languages in this parameter.

Return Value

The return value should contain 0 if the list is correct, or an error value if an error occurred. See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

Text service components must implement a function for this call.

In response to this call, the component should list all its supported scripts and languages, starting with the primary script and language as specified in the `componentFlags` field of its component description structure. The Text Services Manager makes the `GetScriptLanguageSupport` call after a component is opened via the Component Manager function `OpenComponent`. One of the Text Services Manager's uses of this script information is to determine whether to exchange information with the component in the Unicode text encoding.

For example, if a component is associated with a Macintosh script, but includes the `smUnicodeScript` constant in its enumeration of supported constants, then the Text Services Manager determines that the component produces and expects text in the Unicode encoding. Additionally, in this example, the Text Services Manager also concludes that the Unicode characters which the component supports are limited to those encoded in the repertoire of the encoding corresponding to the Macintosh script in the `componentFlags` field of its component description structure. Note that if the script specified in the `componentFlags` field is itself `smUnicodeScript`, the Text Services Manager imposes no restriction on the set of supported characters, and it treats the component as being capable of handling any Unicode character.

Client applications may directly make this call, but the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`TextServices.h`

GetServiceList

Obtains a list of the text service components of a specified type that are currently available. (Deprecated in Mac OS X v10.5.)

```
OSErr GetServiceList (
   SInt16 numOfInterface,
   const OSType *supportedInterfaceTypes,
   TextServiceListHandle *serviceInfo,
   SInt32 *seedValue
);
```

Parameters

numOfInterface

The number of text service interface types supported by your client application.

supportedInterfaceTypes

A pointer to a value of type `InterfaceTypeList` (page 33) specifying the kinds of text services that your program supports. This list helps the Text Services Manager locate text services of the correct interface type.

serviceInfo

A pointer to a handle to a structure of type `TextServiceList` (page 35). If the handle is `NULL`, the Text Services Manager allocates the handle; otherwise, it assumes the handle is a valid text service component list handle, as defined by the `TextServiceListHandle` data type.

seedValue

A pointer to a value that indicates whether the list of text service components returned by `GetServiceList` may have been modified.

Return Value

A result code. See "Text Services Manager Result Codes" (page 66).

Discussion

Your client application is responsible for providing a way—usually a menu—for the user to choose from among all available text service components. To get a list of available text service components to display in a menu, call the `GetServiceList` function. Be sure to filter out input methods, because the Keyboard menu already displays them.

When your application calls `GetServiceList`, the Text Services Manager locates all the text service components of the specified types and creates a text service component list, defined by the `TextServiceList` data type, containing an entry for each of the text service components.

It is possible to register text service components or withdraw them from registration at any time. Once it has compiled a list of text services, the Text Services Manager invokes the `GetComponentListModSeed` function and returns the value in the `modseed` parameter. You can save that value and, the next time you need to draw or regenerate the list of services, call the Component Manager `GetComponentListModSeed` function. If the seed value differs from the one you received from your last call to `GetServiceList`, you need to call `GetServiceList` once more to update the information. Alternately, you can simply call `GetServiceList` each time you need to update the list, although that may be less efficient.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`TextServices.h`

GetTextServiceLanguage

Obtains the current input script and language. (Deprecated in Mac OS X v10.5.)

```
OSErr GetTextServiceLanguage (
   ScriptLanguageRecord *slRecordPtr
);
```

Parameters

slRecordPtr

A pointer to a structure of type `ScriptLanguageRecord` (page 33). Upon completion of the call, this structure describes the language supported by the default (current) text service component for the current keyboard script.

Return Value
A result code. See "Text Services Manager Result Codes" (page 66).

Discussion
Your application should not typically need to call this function.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`TextServices.h`

InputModePaletteItemHit

Informs an input method that a function button on the input mode palette was pressed. (Deprecated in Mac OS X v10.5.)

```
ComponentResult InputModePaletteItemHit (
   ComponentInstance inInstance,
   UInt32 inItemID,
   UInt32 inItemState
);
```

Parameters

inInstance

The component instance.

inItemID

The item ID of the function button pressed on the palette.

inItemState

The new state of the button.

Return Value
Returns a non-null value on successful handling of the call.

Availability
Available in Mac OS X v 10.4 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`TextServices.h`

IsTextServiceSelected

Determines if a text service component is selected. (Deprecated in Mac OS X v10.5.)

```
Boolean IsTextServiceSelected (
   Component aComp
);
```

Parameters

aComp
    The component you want to determine is selected or not.

Return Value
Returns `true` if the component is selected.

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.3 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`TextServices.h`

OpenTextService

Opens a text service component, other than an input method, and associates it with a TSM document. (Deprecated in Mac OS X v10.5.)

```
OSErr OpenTextService (
   TSMDocumentID idocID,
   Component aComponent,
   ComponentInstance *aComponentInstance
);
```

Parameters

idocID
    The identification number of a TSM document created by a prior call to the `NewTSMDocument` (page 23) function.

aComponent

A component identifier for this text service component. You can obtain the component identifier to pass in `aComponent` by comparing the menu item name selected by the user with the component item name found in a `TextServiceInfo` (page 34) structure. You can obtain a `TextServiceInfo` structure by calling the function `GetServiceList` (page 74) and examining the `fServices` field of the `TextServiceList` structure that it produces.

aComponentInstance

Upon completion of the call, a pointer to a component instance. This value identifies your application's connection to a text service component. You must supply this value if you call the text service functions provided by the component directly.

Return Value

A result code. See "Text Services Manager Result Codes" (page 66).

Discussion

This function instructs the Text Services Manager to open a text service component, other than an input method, that a user has chosen and to associate it with a TSM document. The Text Services Manager opens the requested component by calling the Component Manager `OpenComponent` function.

If the specified text service component is already open, the Text Services Manager does not open it again and the `tsmComponentAlreadyOpenErr` error message is returned as a result code. Whether or not the text service is open, the Text Services Manager calls the functions `InitiateTextService` (page 22) and `ActivateTextService` (page 13) for the given text service and returns a valid component instance. Upon completion of the `OpenTextService` call, the selected text service component is initialized and active.

This function is for opening text service components other than input methods. Your application does not need to open or close input methods.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`TextServices.h`

SelectTextService

Selects a text service. (Deprecated in Mac OS X v10.5.)

```
OSStatus SelectTextService (
   Component aComp
);
```

Parameters

aComp

The text service you want to select.

Return Value

A result code. See "Text Services Manager Result Codes" (page 66).

Discussion

This function is intended for use by input methods in text service classes which are additive in nature, that is where the input method can operate in parallel to other input methods in the same class and other additive text service classes. An example of such a class is the Character Palette class. This function is not for use by traditional input methods, such as those that belong to the keyboard input method class.

Availability

Not available in CarbonLib.

Available in Mac OS X v 10.2 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`TextServices.h`

SendAEFromTSMComponent

Sends Apple events from a text service component to a client application. (Deprecated in Mac OS X v10.5.)

```
OSErr SendAEFromTSMComponent (
   const AppleEvent *theAppleEvent,
   AppleEvent *reply,
   AESendMode sendMode,
   AESendPriority sendPriority,
   SInt32 timeOutInTicks,
   AEIdleUPP idleProc,
   AEFilterUPP filterProc
);
```

Parameters

theAppleEvent

A pointer to the Apple event to be sent.

reply

A pointer to the reply Apple event returned by `SendAEFromTSMComponent`.

sendMode

The value that lets you specify one of the following modes specified by corresponding constants: the reply mode for the Apple event, the interaction level, the application switch mode, the reconnection mode, and the return receipt mode. To obtain the value for this parameter, add the appropriate constants. Comprehensive details about these constants are provided in the description of the Apple Event Manager `AESend` function.

sendPriority

The value that specifies whether to put the Apple event at the back of the event queue (set with the `kAENormalPriority` flag) or at the front of the queue (`kAEHighPriority` flag).

timeOutInTicks

The length of time (in ticks) that the client application is willing to wait for the reply or return receipt from the server application before it times out. If the value of this parameter is `kNoTimeOut`, the Apple event never times out.

idleProc

A pointer to a function for any tasks (such as displaying a globe, a wristwatch, or a spinning beach ball cursor) that the application performs while waiting for a reply or a return receipt.

filterProc

A pointer to a function that accepts certain incoming Apple events that are received while the handler waits for a reply or a return receipt and filters out the rest.

Return Value

A result code. See "Text Services Manager Result Codes" (page 66).

Discussion

Component use of Apple events and the function `SendAEFromTSMComponent` is discouraged on Mac OS X. Text service components should use Carbon text input events and the function `SendTextInputEvent` (page 25) on Mac OS X, instead. See "Carbon Porting Notes" below for more details.

The `SendAEFromTSMComponent` function is essentially a wrapper function for the Apple Event Manager function `AESend`. See the description of `AESend` for additional necessary information, including constants for the `sendMode` parameter and result codes.

`SendAEFromTSMComponent` identifies your text service component from the `keyAEServerInstance` parameter in the Apple event specified in the `theAppleEvent` parameter. If a reference constant in a TSM document that corresponds to this parameter is found in the internal data structures of the Text Services Manager, `SendAEFromTSMComponent` adds the reference constant as the `keyAETSMDocumentRefcon` parameter to the given Apple event before sending it to the application.

If the client application is not TSM-aware, `SendAEFromTSMComponent` routes the Apple events to the floating input window to allow bottom-line input.

If your text service component changes the environment in any way—such as by modifying the A5 world or changing the current zone—while constructing an Apple event, it must restore the previous settings before sending the Apple event.

Your text service component should always use the `kCurrentProcess` constant as the target address when it creates an Apple event to send to the Text Services Manager.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Carbon Porting Notes

Note that this function is superseded by the function `SendTextInputEvent` (page 25) on Mac OS X only. With Mac OS X, text service components must be Carbon clients. This is in contrast to Mac OS 8 and 9, where text service components must not be Carbon clients. (This restriction is due to the fact that it is potentially destabilizing for a Carbon-based component to load Carbon in the context of a non-Carbon application.) Therefore, text service components use Carbon text input events and the `SendTextInputEvent` function only on Mac OS X. The function `SendAEFromTSMComponent` must be used by components running on Mac OS 8 and 9.

On any system, the Text Services Manager automatically converts component-originated text input events to the proper form for client applications. On Mac OS X, the Text Services Manager automatically converts component-originated Carbon events to Apple events, if a client application does not provide handlers for Carbon events. Conversely, on Mac OS 8 and 9, the Text Services Manager automatically converts component-originated Apple events to Carbon events and provides these Carbon events to applications, so they have the option of handling them.

Declared In
`TextServices.h`

SetDefaultInputMethod

Sets a default input method to a given script and language. (Deprecated in Mac OS X v10.5.)

Not recommended.

```
OSErr SetDefaultInputMethod (
   Component ts,
   ScriptLanguageRecord *slRecordPtr
);
```

Parameters

ts

The component identifier of the input method to be associated with the script and language combination given in the `slRecord` parameter.

slRecordPtr

A pointer to a structure of type `ScriptLanguageRecord` (page 33). This structure describes the script and language combination to be associated with the input method specified in the `ts` parameter.

Return Value
A result code. See "Text Services Manager Result Codes" (page 66). If the script code and language code specified in the script-language structure are incompatible, `SetDefaultInputMethod` returns the error `paramErr`.

Discussion
You should use the function `SetDefaultInputMethodOfClass` (page 80) instead of this one.

The operating system uses `SetDefaultInputMethod` to associate an input method text service component with a given script and language. The operating system calls this function when the user expresses input method preferences through the Keyboard menu, Keyboard control panel, or other device. The associations made with this function are permanent; that is, they persist after restart. Your application should not typically need to call this function.

Availability
Not recommended. Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`TextServices.h`

SetDefaultInputMethodOfClass

Sets the default input method text service component for a given text service class. (Deprecated in Mac OS X v10.5.)

```
OSStatus SetDefaultInputMethodOfClass (
   Component aComp,
   ScriptLanguageRecord *slRecPtr,
   TextServiceClass tsClass
);
```

Parameters

aComp

The component identifier of the input method to be associated with the script and language combination given in the `slRecord` parameter.

slRecPtr

A pointer to a structure of type `ScriptLanguageRecord` (page 33). This structure describes the script and language combination to be associated with the input method specified in the `ts` parameter.

tsClass

The text service class whose component and script language record you want to obtain. Pass `kKeyboardInputMethodClass` to specify a keyboard input method. Pass `kInkInputMethod` to specify an Ink input method.

Return Value

A result code. See "Text Services Manager Result Codes" (page 66).

Discussion

Availability

Not available in CarbonLib.

Available in Mac OS X v 10.2 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`TextServices.h`

SetTextServiceLanguage

Changes the current input script and language. (Deprecated in Mac OS X v10.5.)

```
OSErr SetTextServiceLanguage (
   ScriptLanguageRecord *slRecordPtr
);
```

Parameters

slRecordPtr

A pointer to a structure of type `ScriptLanguageRecord` (page 33) specifying the new script and language combination.

Return Value

A result code. See "Text Services Manager Result Codes" (page 66).

Discussion

The operating system calls this function when the user switches the keyboard script, so that the Text Services Manager can synchronize the input method with the current keyboard script.

Your application should not typically need to call this function.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`TextServices.h`

TSMCopyInputMethodEnabledInputModes

Obtain the array of the enabled (and visible) input modes for a component. (Deprecated in Mac OS X v10.5.)

```
Boolean TSMCopyInputMethodEnabledInputModes (
   Component inComponent,
   CFArrayRef *outInputModeArray
);
```

Parameters

`inComponent`
The component whose input modes you want to obtain.

`outInputModeArray`
On return, points to an array of the enabled and visible input modes for the specified component. This function is meaningful only for input methods that adopt the input mode protocol. If the component passed is not input mode-savvy, the returned array is `NULL`. It is the responsibility of the caller to release the returned array.

Discussion
You use this function to allow an input method to query the system for the subset of its own input modes that are enabled. This allows you to omit from the component UI any input modes that are disabled by the user or the system. The enabled input modes returned in the array are always visible ones. That is, the array contains those input modes for which `kTSInputModeIsVisibleKey` is true; non-visible input modes are not tracked by the system.

Availability
Available in Mac OS X v 10.3 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`TextServices.h`

TSMInputModePaletteLoadButtons

Notifies the input mode palette of changes to the controls for an input method and replaces the current controls with the new control array. (Deprecated in Mac OS X v10.5.)

```
void TSMInputModePaletteLoadButtons (
   CFArrayRef paletteButtonsArray
);
```

Parameters

paletteButtonsArray

A CFArray that contains descriptions of the controls. Use a CFDictionary to describe each control. See "Input Mode Palette Control Keys" (page 53) for a description of the keys you can supply.

Availability

Available in Mac OS X v 10.4 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`TextServices.h`

TSMInputModePaletteUpdateButtons

Notifies the input mode palette of changes to the controls for an input method and updates the controls. (Deprecated in Mac OS X v10.5.)

```
void TSMInputModePaletteUpdateButtons (
   CFArrayRef paletteButtonsArray
);
```

Parameters

paletteButtonsArray

A CFArray that contains descriptions of the controls. Use a CFDictionary to describe each control. See "Input Mode Palette Control Keys" (page 53) for a description of the keys you can supply.

Discussion

This function updates controls based on the control tag ID. It doe not replace or remove existing controls.

Availability

Available in Mac OS X v 10.4 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`TextServices.h`

TSMSelectInputMode

Sets the specified input method input mode as the current input source. (Deprecated in Mac OS X v10.5.)

```
OSStatus TSMSelectInputMode (
   Component inComponent,
   CFStringRef inInputMode
);
```

Parameters

inComponent
The component whose input mode you want to set.

inInputMode
The input mode you want to set as the current input source.

Discussion

You use this function to allow an input method to select one of its own input modes as the current input source and update the Text Input menu icon in the menu bar. This function is only meaningful for input methods that adopt the input mode protocol.

Availability

Available in Mac OS X v 10.3 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`TextServices.h`

TSMSetInlineInputRegion

Defines a region within a TSM document in which inline input can occur. (Deprecated in Mac OS X v10.5.)

```
OSStatus TSMSetInlineInputRegion (
   TSMDocumentID inTSMDocument,
   WindowRef inWindow,
   RgnHandle inRegion
);
```

Parameters

inTSMDocument
The identification number of a TSM document created by a prior call to the `NewTSMDocument` (page 23) function.

inWindow
A reference to the window that contains the inline input session. You can pass `NULL` for this parameter to indicate the window that currently has user focus.

inRegion
The current inline input region. This region should be in coordinates local to the port associated with the window specified in the `inWindow` parameter. The region must be recomputed each time the text content of the inline input session changes (such as after an Update Active Input Area event) and when the region moves for other reasons (such as window resizing or scrolling). If you pass `NULL` for this parameter, the Text Service Manager defaults to intercepting mouse events for the window's entire content region.

Return Value

A result code. See "Text Services Manager Result Codes" (page 66).

Discussion
The `TSMSetInlineInputRegion` function informs the Text Services Manager of the region occupied by an inline input session. If certain mouse events (such as clicks and mouse-moved events) occur within this region, the Text Services Manager forwards these events to the current input method, so the component can respond to the user's actions.

When a mouse-moved event occurs inside an inline input region (as registered via the `TSMSetInlineInputRegion` function), the Text Services Manager promotes the `kEventMouseMoved` event, after it has been received by the applicable control or window, to the window-specific `kEventWindowCursorChange` event. The Text Services Manager first delivers the `kEventWindowCursorChange` event to the active input method, then, if it is not handled, to any other active text services. If the event has still not been handled, the Text Services Manager finally passes the event to the application's `kEventWindowCursorChange` event handler, if any. This event-dispatching process gives applications that need to see the low-level mouse-moved events a chance to see these events first, while providing a mechanism for text services and applications to act on these events without conflict. After completing this process, the Carbon Event Manager converts any `kEventWindowCursorChange` event that remains unhandled to a "classic" mouse-moved event for `WaitNextEvent` clients.

If an application does not call this function, when an input method is active the Text Services Manager by default intercepts mouse events in the entire content region of the window that currently has user focus.

Availability
Available in CarbonLib 1.1 and later.

Available in Mac OS X v 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`TextServices.h`

Unsupported Functions

This section lists functions that are unsupported and you should no longer use. Table B-1 provides information on what you should do in place of using these functions.

Table B-1 Porting notes for unsupported functions

Unsupported functions	Porting notes
`InitTSMAwareApplication`	Applications are automatically considered TSM-aware and therefore have no need to call this function.
`CloseTSMAwareApplication`	Not needed due to the Mac OS X event model.
`TSMEvent`	Not needed due to the Mac OS X event model.
`TSMMenuSelect`	Not needed due to the Mac OS X event model.
`SetTSMCursor`	Instead use the function `TSMSetInlineInputRegion` (page 84).
`NewServiceWindow`	Instead use the Window Manager function `CreateNewWindow` (of class `kUtilityWindowClass`).
`CloseServiceWindow`	Instead use Window Manager functions for working with utility windows.
`GetFrontServiceWindow`	Instead use Window Manager functions for working with utility windows.
`FindServiceWindow`	Instead use Window Manager functions for working with utility windows.
`TextServiceEvent`	Instead use the function `TextServiceEventRef` (page 27).
`UCTextServiceEvent`	Instead use the function `TextServiceEventRef` (page 27).
`TextServiceMenuSelect`	Not needed.
`SetTextServiceCursor`	Not needed.
`NewCServiceWindow`	Instead use Window Manager functions for working with utility windows.

Document Revision History

This table describes the changes to *Text Services Manager Reference*.

Date	Notes
2005-11-09	Added a hyperlink.
2005-08-11	Corrected typographical errors.
2005-07-07	Updated for Mac OS X v10.4.
2005-06-16	Updated discussion for the function NewTSMDocument.
2003-10-15	Updated documentation with information on Carbon events and other items added for Mac OS X version 10.3.
	Added documentation for the Carbon events relevant to the Text Services Manager, including the following: "Carbon Event Class for TSM Document Access" (page 39), "Carbon Events for TSM Document Access" (page 39), "Carbon Event Parameters for General TSM Events" (page 46), "Carbon Event Parameters for TSM Document Access" (page 47), "Attribute Bits for TSM Document Access Carbon Events" (page 38), and "Attribute Masks for TSM Document Access Carbon Events" (page 38). TSM-related Carbon events and event parameters are declared in the header file CarbonEvents.h.
	Added documentation for the functions: "CopyTextServiceInputModeList" (page 14), "SelectTextService" (page 77), and "IsTextServiceSelected" (page 76).
	Added documentation for "TSM Document Interfaces" (page 61), the "TSM Document Interface Type" (page 32), "Text Services Property Values" (page 60), "Input Mode Dictionary Key" (page 52), and "Individual Input Mode Keys" (page 54).
	Added information to "Document Property Tags" (page 50), "Text Service Classes" (page 57), "Text Service Properties" (page 59), and "Version Constants" (page 65).
2003-02-12	Added documentation for the functions `SetDefaultInputMethodOfClass` (page 80), `GetDefaultInputMethodOfClass` (page 71), `TSMSetDocumentProperty` (page 31), `TSMGetDocumentProperty` (page 29), and `TSMRemoveDocumentProperty` (page 30).
	Moved unsupported functions to an appendix.
	Updated formatting to improve document appearance and search capability.

Index