Apple Event Registry: Word Services Suite

Working Software, Inc.

Beta 3 Draft

Working Software, Inc.
PO Box 1844
Santa Cruz, CA 95060
(408) 423–5696
http://www.working.com

Writers: Michael D. Crawford and Amr Eissa
Developmental Editor: Wendy Krafft
Illustrator: Janet Anders
Production Supervisor: Theresa Fennern

Contents

Figures and Tables

The Word Services suite

The Word Services suite contains definitions of Apple event constructs that are used in operations involving spell-checking and grammar-checking. The suite can be used for any service that must examine and change the text in a document. The suite should be supported by applications that let the user edit text, such as word processors, drawing programs, spreadsheets, or electronic mail packages. The suite should also be supported by applications that perform spell-checking, grammar-checking, bibliography, envelope addressing or hyphenation tasks.

Introduction to the Word Services suite

The Word Services suite defines Apple event constructs that allow an application to have its text inspected and modified by programs that operate on text, such as spellers or grammar checkers. The Word Services suite is based on the Core suite and contains Apple event constructs that are not included in the Core suite.

Word processor in this document means any application that allows text editing, not just programs sold as word processors. The term *client* is used interchangeably with *word processor*.

Speller here refers to spelling and grammar checkers, or any other program that acts as a server in this protocol. The term *server* is used interchangeably with *speller*.

Word Services is intended to be simple for client programs to implement. A client developer may choose to leave out optional features of the suite to save work. The burden of supporting the suite falls on the server developer. This is of great benefit to the developer community as it is expected that there will be far more client applications than server applications.

The terms*may* or *can* are used to indicate that a feature is optional. For example, "A client may support background highlighting." The term *must* is used to indicate a mandatory feature: "If the speller receives an error code in response to setting the background highlighting, it *must* display the questioned text within its own window." There is much more flexibility allowed to client applications than to servers.

Overview of the Word Services suite

The Word Services suite defines Apple events for

- applying a word service to one or more blocks of text
- Registering the location, menu item, and icon of the server application
- replacing text upon user confirmation
- spell-checking a single word
- starting an interactive spelling session
- sending text to a server for interactive checking

The Word Services suite extends definitions of the three object classes—cApplication, cChar, and cText—that are defined in the Core suite.

Applications that support the Word Services suite

The following types of applications support the Word Services suite as servers:

- spellers
- grammar checkers
- hyphenators
- envelope addressers
- natural language translators
- indexers
- bibliography generators

Typical client applications for the Word Services suite

The following types of applications are clients of applications that support the Word Services suite:

- word processors
- page layout applications
- database applications
- spreadsheets
- electronic mail applications
- graphic applications that allow text entry
- personal information management applications

Relationship of the Word Services suite to the Core suite

The Word Services suite is designed to be easy for a word processor to implement. To support the protocol, the word processor must support a small subset of the Core suite, particularly the Get Data and Set Data events, the formRange key form, and the formAbsolutePosition key form, including positions relative to the end of the container.

Sample source code

The sample source code displayed here is from the Word Services Software Development Kit. The SDK contains the complete source code to a simple word processor called Writeswell Jr.™, as well as a debugging version of a speller, a dictionary, a tutorial on the sample code and this document. The code is written in ThinkC, but should be readily portable to MPW or other C compilers.

The Word Services SDK is available on many online services.

Usage scenarios

There are four basic scenarios for using the Word Services suite: *server-interface interactive, server-interface batch, client-interface interactive,* and *client-interface batch.* Each mode of operation has its own advantages. In server-interface interactive mode, the user interface is provided by the server and the client does not have to create one. The server-interface batch provides simple control flow.

The client-interface interactive mode provides an integrated look of the user interface and is designed for developers who want to create their own interface. The client-interface batch is simple to implement. Applications can also use the Word Services suite to spell-check a single word directly.

Each client or server may support one or more of the scenarios. It is not expected that every application will provide for each kind of interaction.

There are two main styles of operation: *server-interface* and *client-interface*. In the server-interface style of operation, the user interface is provided by the server. Misspelled words are presented with a list of suggested spellings in a dialog box drawn by the speller. The user indicates whether to skip or replace the misspelled word within the dialog box. If the word processor supports it, the speller can request that the misspelled word be highlighted in a window provided by the word processor, thus displaying the word in its own style and in the context of the whole document. If background highlighting is not supported, the speller displays the word in its own window, possibly with some simple formatting.

In the client-interface style of operation, the speller acts as a background server. The word processor sends text to the speller, which sends back a report that contains a list of all the words that were misspelled. The word processor then draws its own dialog box and requests the speller to provide a list of suggested alternatives.

Both server-interface and client-interface styles of operation have two modes of spell-checking: *batch* or *interactive.* Batch mode spell-checks a range of text (typically either the current selection or the whole document) upon an explicit user command.

In interactive mode, spell-checking is performed while the user types. The word processor parses words and sends them to the speller. If the word is correct, the speller does not respond. If the word is incorrect, an error signal is generated, such as SysBeep. The user can either bring up a dialog box to do the replacement or ignore the signal (either because the word is correct or because the word has already been fixed or deleted).

Note that the interactive mode can be slow. The time it takes to generate an Apple event, do a MultiFinder switch (switch to the speller), receive the event, look up the word in the dictionary, and then switch back to the word processor might be unacceptable to the user, especially if the text is sent one character at a time by the application.

For this reason, interactive checking is normally done asynchronously; the word processor sends the Apple event and continues without switching out. During the next normal switch, the speller starts the word lookup, but allows itself to be switched back out before it is done (if the lookup takes too long). There are several opportunities to check each word, as each typed character returns from a separate WaitNextEvent call.

Applications employing either style of interactive operation must be able to deal with typical cases, such as

- the user typing text after a misspelling and before reacting to an error signal
- the user typing an incorrect word while the speller identifies an earlier misspelling
- the user moving the mouse and typing bits of words in different places
- the user pressing the backspace key
- the user clicking the mouse in the middle of a word and changing it

Required support for a Word Services client

The Word Services client *must* support *at least* a minimal set of Core suite Apple events and object model types, although for other purposes it should support them all. The Apple events a Word Services client *must* support are as follows:

- Get Data
- Get Data Size
- Set Data

The key forms that a Word Services client *must* support are as follows:

- formAbsolutePosition (including positions relative to the beginning of the container and positions relative to the end of the container)
- formPropertyID
- formRelativePosition
- formRange

The recommended but optional properties and element classes that the Word Services client should support are as follows:

- pBatchMenuString
- pInteractiveMenuString
- pLockTransactionID
- cWord
- cParagraph

Batch checking

This section discusses the following types of batch checking:

- Predatory server-interface batch checking
- Server-interface batch checking
- Client-interface batch checking

Predatory server-interface batch checking

Predatory server-interface batch checking allows spell-checking of documents created by word processors that do not support the Word Services protocol. As long as the word processor supports the Core suite sufficiently, its documents are processed by the Word Services server. This also allows a user to drag a document icon to the speller in the Finder. The speller's response to an 'odoc' event (Open Document event) is to locate the creator and have it open the document.

To perform predatory server-interface batch checking, the user launches the speller and then opens a word processor document from the file menu of the speller. The speller launches the application that created the document and uses the Get Structure event to locate the text fields within the document. The speller then sends a Get Data event to retrieve the text, which can be a range of characters, a paragraph, or whole text. The speller than sends the Set Data events to set a range of characters, when it needs to replace them. The range specifier consists of two object specifiers, one for the start and one for the end of the range.

For predatory checking to work, the speller explores the word processor's object to locate the text items. Locating text items is an easy task when performed in a simple text editor window, but it is quite complicated when attempted in a highly structured document.

Server-interface batch checking

To initiate server-interface batch checking, the user selects the text to be spell-checked. This is done either by shift-clicking (clicking the mouse button while holding down the shift key) several cells in a spreadsheet or text fields in a drawing, highlighting a range of text in a word processor, or selecting the spell-checking menu option from within the application.

For each text field to be checked, the word processor creates an object specifier. Note that the object specifier refers to an object in the word processor's own document and that each object is equivalent to a cText object. (These objects must contain paragraphs or characters. Also, if you do a Get Data on the object itself, all the text within it is returned.)

To locate the speller for the first time, the word processor uses the PPCBrowser function, which returns the information about the application (in this case, the speller) that the user has selected in the target ID record. (For more information about the PPCBrowser, see the Apple Event Manager chapter of *"Inside Macintosh: Interapplication Communication."*)

After locating the speller, the word processor uses the Get Data event to obtain pBatchMenuString, pInteractiveMenuString, and pLocation properties from the speller. The pLocation property is an alias record that contains the creator code of the server, as well as its filesystem location. The word processor must save all these properties in its own preference file or within its own resource file for future use. A separate resource file is recommended, because it allows the user to save the preferences whenever an application is upgraded, and also allows each user to have their own set of preferences when an application is shared over the network. The next time the user requests spell-checking, the word processor simply checks the alias record to locate the speller.

Once the word processor has located the speller, it checks the creator code to see whether the speller is already running. The word processor connects directly to the speller if it is running. If the speller is not running, the word processor can use the alias record to launch the speller (provided that the speller file already exists on the machine). This can be done by sending the Open Selection event, with the alias record as a parameter, to the Finder.

After establishing a connection with the speller, the word processor sends a Batch Process My Text event to the speller. The keyDirectObject parameter to the event is either a single object specifier or a list of several object specifiers (or possibly a specifier that indicates it is the head of a list of object specifiers kept by the word processor). Once the reply is received, the word processor resumes its normal event processing. The particular service that is performed on the text depends on the server to which the word processor connects to. The server can be a grammar checker, a speller, or another program.

The key to easy implementation of the protocol is that the word processor need only send one event, then forget about it and handle other events normally. The whole operation involves an extended sequence of Apple events and replies, but the logic to drive the process is entirely in the speller. The word processor need only respond passively to the Apple events; there is no spell-checking mode that it must enter. Of course, the word processor must support the necessary events, classes, and key forms from the Core suite.

Client-interface batch checking

Client-interface batch checking is useful to word-processor developers who want to present their own interface, and to server developers who want to write a server that can be accessed from a remote machine.

To perform client-interface batch checking, the user selects text to be checked either by shift-clicking several cells in a spreadsheet or text fields in a drawing, highlighting a range of text in a word processor, or selecting the "Check Whole Document" option from the menu. The user then selects a service, such as "Check Spelling," from the Services menu.

For each text field to check, the word processor creates an object specifier. Next, the word processor retrieves the alias record to the selected service from its preference file. Then, the word processor checks to see whether the speller is currently running; if it is, the word processor sees its creator code in the list of PPC ports by using the IPCListPorts function. If the speller is not running, the word processor launches the speller from the saved location by using the Open Selection event from the Finder Suite or by calling LaunchApplication directly if the application is on the same machine.

The word processor sends the speller a Batch Process My Text event with the keyDirectObject parameter consisting of either a single object specifier or a list of several object specifiers. The keyFacelessMode parameter of the Batch event is of typeBoolean with a TRUE value. After sending the event, the word processor waits for the reply from the speller before resuming its normal event processing. On receiving the Batch Process My Text event, the speller either sends errAEInTransaction as an error code in reply if it is busy, or records the object specifiers and the address of the sender, and then returns a reply indicating no error. (Refer to the section "Communicating with the Speller" for details regarding interaction between the word processor and the speller.)

Interactive checking

This section discusses the following types of interactive checking:

- Server-interface interactive checking
- Client-interface interactive checking

Server-interface interactive checking

To perform server-interface interactive checking, the user requests that interactive checking be turned on (generally by selecting a menu option). A word processor can have a preference setting that can automatically turn on interactive checking whenever the word processor is launched.

The first time the user requests interactive checking, the word processor locates the speller by using the PPCBrowser function, which returns the information about the application (in this case the speller) that the user has selected in the target ID record. (For more information about the PPCBrowser, see the Apple Event Manager chapter of *"Inside Macintosh: Interapplication Communication."*)

After locating the speller, the word processor uses the Get Data event to obtain pBatchMenuString, pInteractiveMenuString, and pLocation properties from the speller. The pLocation property is an alias record that contains the creator code of the server, as well as its file system location. The word processor must save all these properties in its own preference file or within its own resource file for future use. A separate resource file is recommended, because it allows the user to save the preferences whenever they upgrade the application, and also allows each user to have their own set of preferences when an application is shared over the network. The next time the user requests spell-checking, the word processor simply checks the alias record to locate the speller.

Once the word processor has located the speller, it checks the creator code to see whether the speller is already running. The word processor connects directly to the speller if it is running. If the speller is not running, the word processor can use the alias record to launch the speller (provided the speller file already exists on the machine). This can be done by sending the Open Selection event, with the alias record as a parameter, to the Finder.

After connecting with the speller, the word processor sends an Interactively Process Text event to the speller and waits for a reply. If the reply is received without an error, the word processor starts sending words to the speller. As the user finishes typing each word, the word processor sends it to the speller in the Check Word Interactively event. (Refer to the section “Communicating with the Speller” for details regarding interaction between the word processor and the speller.)

Client-interface interactive checking

This scenario is similar to the server-interface interactive scenario, except that when the user requests that a questioned error be looked up, the speller sends a Query Replace to the word processor that provides the user the option to replace the text.

To perform client-interface interactive checking the user turns on interactive checking by selecting the menu option. A word processor can also have a preference setting that automatically turns on interactive checking whenever the word processor is launched. When the word processor is launched the first time, it locates the speller by using the PPCBrowser function. The word processor than uses the Get Data event to obtain pBatchMenuString, pInteractiveMenuString, and pLocation properties from the speller. The pLocation property is an alias record that contains the creator code of the server, as well as its file system location. The word processor must save all these properties in its own preference file or within its own resource file for future use. A separate resource file is recommended, because it allows the user to save the preferences whenever an application is upgraded, and also allows each user to have their own set of preferences when an application is shared over the network. The next time the user requests spell-checking, the word processor simply checks the alias record to locate the speller.

The word processor checks the creator code to see whether the speller is running. It connects directly to the speller if it is running, otherwise it uses the alias record to launch the speller (provided the speller file already exists on the machine). To launch the speller it sends the Open Selection event with the alias record as a parameter to the Finder. After connecting with the speller, the word processor sends an Interactively Process My Text event with the keyFacelessMode parameter as TRUE, and waits for the reply. If the reply is received without an error, the word processor starts sending each word to the speller (as the user finishes typing it) in a Check Word Interactively event. Refer to the next section for details regarding interaction between the word processor and the speller.

Communicating with the speller

To communicate with a speller, a word processor first locates the speller by checking the creator code in an alias record to see whether the speller is running. If the speller is running, the word processor connects directly to it. Otherwise, it communicates with the speller by sending the Open Selection event (with the alias record as a parameter) to the Finder. Once the word processor has connected to the speller, it sends the appropriate Apple Event.

In the server-interface batch mode the word processor sends a Batch Process My Text event. When the speller receives the event, it records the object specifiers and the address of the sender, then returns a reply indicating that there is no error. If the speller is busy, it sends errAEInTransaction as an error code in the reply. The word processor receives the reply from the speller and resumes its normal event processing.

The speller then makes itself the foreground application by calling AEInteractWithUser and by displaying a dialog box (or another type of user interface). Note that the dialog box must be movable, allowing the user to view the text in the word processor's window.

For each object specifier in the list of object specifiers sent by the word processor:

- The speller uses a Set Data event on pLockTransactionID for the object to prevent other Apple events from changing the text while the text is being worked on. All subsequent events sent by the speller must have the transaction ID.
- The speller makes new object specifiers from the original object specifier. The new object specifiers pick out paragraphs or ranges of characters within the original object specifier.
- The speller sends a Get Data Size event to find the amount of text within the object. Note that Apple Events cannot contain more than 64K of data - even though this limit may change in the future, a particular speller might have a limited amount of memory. Also, placing a parameter on an Apple Event causes *two* copies of the data to exist temporarily - the parameter's descriptor, and the data within the event itself.

- The speller sends a Get Data event to retrieve the text. The speller can get the whole text object, paragraphs within it, or ranges of characters.
- The speller sends a Set Data event to set the pBackgroundHilite property of a ranges of characters that it wishes the user to change.
- If the word processor does not support background highlighting, it returns an error code to the Set Data event. If the speller receives an error code, it creates a window to display the text itself. Also, it should allow the user to select a preference of whether the text is shown in the speller or the word processor window.
- The speller sends Set Data events to set a range of characters when it needs to replace them. The range specifier consists of two object specifiers for a single character. Each character specifier uses formAbsolute position to give the offset from the end of the container.
- The speller uses a Set Data event on the pLockTransactionID property of the text field, setting it to kAEAnyTransaction. This allows other events to access the object.

When the speller is done, it restores the word processor to the foreground of the screen and quits.

In the client-interface batch mode, communication between the word processor and the speller is similar to that in the server-interface batch mode. However, the speller does not send events to highlight and replace the text directly. Instead, it sends “Query Replace” events to the word processor that include an object specifier for the text in question, a list of replacement strings, and a text string that explains what the error is. After connecting with the speller, the word processor sends a Batch Process My Text event. If the speller is busy, it returns a reply with errAEInTransaction as an error code. Otherwise, the speller records the object specifiers and the address of the sender, and then returns a reply indicating that there is no error. Upon receiving the reply, the word processor resumes its normal event processing.

If the Batch Process My Text contains a single object specifier, the speller uses that specifier as a container for a table of object specifiers that is kept in the word processor. Each element of the table is an object specifier that refers to the actual text. Using the table instead of sending the object specifiers directly allows a large number of text blocks to be specified without exceeding the 64K byte limit on the data size that can be sent through an Apple event.

The word processor then displays a dialog box (or some other user interface). The dialog box should be movable to allow the user to view the text in the word processor's window.

For each object specifier in the list of object specifiers sent by the word processor:

- The speller uses a Set Data event on pLockTransactionID for the object. This is done to prevent other Apple events from changing the text while it is being worked on.

- The speller makes new object specifiers from the original object specifier. The new object specifiers specify paragraphs or ranges of characters within the original object specifier.
- The speller sends a Get Data Size event to find the amount of text within the object. (Note that Apple events cannot contain more than 64K of data, and a particular speller might have a limited amount of memory.)
- The speller sends a Get Data event to get the text. It can get the whole text object, paragraphs within it, or ranges of characters.
- If a range of characters must be changed, the speller sends a Query Replace event to the word processor. The Query Replace event includes the keyAEData parameter (a list of zero or more strings that can be used as possible replacements), and the keyErrorString parameter (a message string that indicates the nature of the error). The word processor displays the message and allows the user to either select one of the suggested words, type in a replacement, or skip the error.
- The speller then uses a Set Data event on the pLockTransactionID property of the text field and sets it to kAEAnyTransaction so that other events can access the objects.

When the speller is done, it once again makes the word processor the foreground application.

In server-interface interactive checking, after connecting with the speller, the word processor sends an Interactively Process Text event and then waits for reply. If the speller is busy, it sends errAEInTransaction as an error code in the reply, otherwise it returns a reply with no error. If the word processor receives a reply without an error, it starts sending each word to the speller. As the user finishes typing each word, the word processor sends it to the speller by the Check Word Interactively event. (In general, typing a word break character, such as a space or punctuation mark, causes a word to be sent).

Every separate word is sent, including any punctuation, because the speller might have the ability to check capitalization after periods, or it might be a grammar checker that can rigorously check punctuation.

Each word is sent asynchronously, with no reply requested. Because it can take some time for the speller to check the word, waiting for replies causes long pauses. Not using replies reduces overhead. If the speller reports a misspelled word, it does so on its own with the Notification Manager.

Each event sent to the speller contains the following parameters:

- keyAEDirectObject —a typeObjectSpecifier that specifies the word that is sent. It must be of formAbsolutePosition relative to the beginning of the container, rather than the end, as is used in batch checking, because the end of the container changes as the user types.
- keyAEData —the actual text of the word that is sent.

If the speller finds a spelling error, it should generate a beep (SysBeep) or blink the menu bar to alert the user. The speller should use the Notification Manager to set an icon blinking in the menu bar, because the user might choose to ignore the spelling error (it might be a correct word that is not in the dictionary). The user can then select the Check Word option from the word processor's menu. The word processor must send the Process Last Error event to the speller. If there is a selection, the Check Word item is grayed out, so that the user is forced to choose the Check Spelling or Check Grammar items. The word processor can optionally allow Check Word if a single word is selected.

Upon receiving the Process Last Error event, the speller sends a Get Data event back to the word processor to get the text of the erroneous word again. This recall procedure is performed because something might have occurred to invalidate the object specifier for the word (text might have been added before the word, or the user might have deleted the word).

If the returned text does not match the text that was originally questioned, the speller should generate an alert and return control to the word processor. If the text does match, the speller should show its dialog box. If the user wants to replace a word, the speller uses a Set Data event to perform the replacement.

A word processor can provide a menu option that, when selected by the user, allows the speller to automatically replace words whenever it finds a spelling error without consulting the user; it searches for the correct word and then uses the Set Data event to do the replacement.

In client-interface interactive checking, after connecting with the speller, the word processor sends the speller an Interactively Process Text event with the keyFaceLessMode parameter as TRUE. Note that the TransactionID for the Interactively Process My Text event should have a unique value, because the TransactionID is used by subsequent Query Replace events. (Do not set the kAENeverInteract flag in the event. The speller might need to interact, such as to prompt a user to locate a dictionary file). On receiving the event, the speller records the address of the sender and then returns a reply. The speller sends errAEInTransaction as an error code to the word processor if it is busy, otherwise the reply does not contain any error. If the word processor receives the reply without an error, it starts sending words to the speller. As the user finishes typing each word, the word processor sends it to the speller in the Check Word Interactively event. (In general, typing a word break character, such as a space or punctuation mark, causes a word to be sent.) Every separate word is sent, including any punctuation.

Each Check Word Interactively event sent to the speller contains the following parameters:

- keyAEDirectObject —a typeObjectSpecifier that specifies the word that is sent. It must be of formAbsolutePosition relative to the beginning of the container, rather than the end, as is used in batch checking, because the end of the container changes as the user types.
- keyAEData —the actual text of the word that is sent.

Each word is sent asynchronously, without requesting a reply from the speller. This is done because the speller can take some time to check the word; thus, waiting for replies results in pauses. If the speller finds a misspelled word, it should generate a beep or blink the menu bar to alert the user, or use the Notification Manager to set an icon blinking in the menu bar, because the user might ignore the word (it might be a correct word that is not in the dictionary), or manually correct the word.

The user can then select the Check Word option from the word processor's menu. The word processor then sends a Process Last Error event to the speller. Upon receiving the event, the speller sends a Get Data event back to the word processor to get the text of the erroneous word again. This is important because something might have occurred to invalidate the object specifier for the word (text might have been added before the word, or the user might have deleted it). If the returned text does not match the text that was originally questioned, the speller should show an alert, then return control to the word processor. If there is a match, the speller sends a Query Replace event to the word processor. The Query Replace event contains the object specifier for the erroneous word, as well as any suggested replacements, and a text string that explains what the error is. The word processor can then display a dialog box that allows the user to edit the text.

A word processor can also provide the user with a menu option that, when selected, allows the speller to automatically replace words whenever it finds a spelling error without consulting the user.

Enhancement options

The Word Services Suite is very flexible. The word processor developer may provide basic functionality with little labor, or the developer may choose to provide a richer interface at the expense of a moderate amount of work.

This section discusses the following enhancement options:

- Examples of menus
- Installing a server
- Handling menus
- Checking a selection
- Multiple languages
- Highlighting
- Text locking
- Error handling
- Read-Only documents

Examples of Menus

It is possible that a user might have a number of different servers on his or her machine, each with a menu item for interactive and batch service mode (a particular speller might not support both modes). For a richer interface, the word processor can be programmed to request the pMenuIcon property to get a typeSmallIcon object that it can use as a small icon for the menu items.

Figure 1 is a suggested layout for such a menu. In this example, new services are appended to the end. (This is just a suggestion.) There are many alternative to using this menu. For example, services can be selected from a scrolling list. Alternatively, minimal support might be provided by going to the PPCBrowser every time checking is requested.

■ **Figure 1** A suggested layout for the menu

If the client does not support interactive checking, it can have a much simpler menu as in Figure 2. The "Check Selection" item may be eliminated as well if the word processor handles this automatically.

■ **Figure 2** A simple menu

If the client program allows for completely customizable menus, there need not be a separate Services menu at all. Instead, the "Get New Service" and "Remove Services" items may be incorporated into the dialog that allows menu editing, and each service may be placed on any menu that the user desires.

Installing a server

There are several ways on might initially locate a server. The user may launch the server and then select it from the client using the PPCBrowser dialog.

Alternatively, the user may locate the server's application file using a Standard File open dialog. The client program will then launch the speller.

These two methods have the disadvantage that there is no way to easily filter out non-Word Services applications. A better way to locate the server might be to have the user launch it, and then send a Get Data event to ask for the menu strings from each application that is running. Non-servers will return an error code. Word Services Servers will return at least one of the strings. The client must check whether the server is already installed, and then add the server to its menu.

Handling menus

When a word processor locates a speller, it sends Get Data events to request the pBatchMenuString, pInteractiveMenuString, and pLocation properties from the speller.

The strings returned are added to the word processor's menu. By doing so, the user can have separate items for each service. The strings, aliases and icons are saved in the preference file for future use.

Example code for handling the preferences is in the Word Services SDK. It should be emphasized that this code is an example only - the developer may want to use a different method.

The preference file contains a record that records the type of service for each menu item—either batch service, interactive service, or no service. (If there is no service, then this menu item is not in use).

```
typedef struct {
     …
     short     serviceType[ kMaxServices ];
} WWJrPrefs, **WWJrPrefsHdl;

typedef enum {
     kNoService = 0,
     kBatchService,
     kInteractiveService
} ServiceType;
```

When the menu is built, a global array is used that stores the resource ID for each service menu item.

Because menu icons are specified by resource id within the menu item, rather than as handles to blocks of memory, the preferences file must be the current resource file when MenuSelect is called:

```
Boolean DoMouseDown( EventRecord *eventPtr )
{
     WindowPtr  theWindow;
     Boolean         result = true;
     short           curFile;
     short           deskPart;
     …

     deskPart = FindWindow( eventPtr->where, &theWindow
);
     switch( deskPart ){
          …
          case inMenuBar:
               FixMenuMarks();
               curFile = CurResFile();
               UseResFile( gPrefFileRefNum );
               /* Make SICN resource accessible */
               result = DoMenuCommand(
                         MenuSelect( eventPtr->where )
);
               UseResFile( curFile );
               FixMenuMarks();
               break;
```

When a server creates an alias record to reply to the Get Data event for the pLocation property, it must place its own creator code in the userType field of the alias record (this field is not used by the alias manager, and may be used as desired by applications). When the service is selected from the menu, the client gets the creator code from the alias record. The client then uses the Process Manager to see if the server is already running. (The FindAProcess function iterates through the processes using GetNextProcess and GetProcessInfo.) If it is not, the word processor launches the speller by sending the Open Selection event, with the alias record as a parameter, to the Finder.

```
/* See if the speller is out there */
signature = (*aliasHdl)->userType;

if ( !FindAProcess( signature, &psn, &pInfo,
          (FSSpecPtr)NULL, (StringPtr)NULL ) ){

     err = LaunchSpeller( aliasHdl );
     if ( err ){
          return err;
     }
```

Checking a selection

Figure 1 shows a menu in which the user may choose between selecting the whole document or checking a selection by toggling the check mark on the "Check Selection Only" menu item. The Word Processor developer may choose instead to always check a selection if a selection exists, so that the menu item is unnecessary, as in Figure 2. (If there is no selection, the whole document should be checked).

The client program has complete control over which text is processed. It is up to the client to prepare the list of object specifiers for the text. If the user wishes to check a particular cell in a spreadsheet, then the spreadsheet will send an object specifier for that cell to the server.

Multiple languages

Spellers already exist in the French, German, and Turkish languages.

Handling multiple languages in a document is quite a tricky problem. There is no reliable way for a word processor to be sure that a user is writing in a particular language. It is helpful to give the user some means to declare the language that a range of text is in, but many client applications will not provide for this, and the user might not care to go to the trouble of declaring the language when creating the document. This problem exists with conventional OEM and document-based spell-checkers. C'est la vie.

The server must be able to receive text that is in an unexpected language. This might be as simple as reporting each word as misspelled and allowing the user to skip it. A speller may be fancy enough to handle multiple languages and switch between them as the language changes.

A simple client implementation may reply to Get Data events from the server with blocks of text that are of typeChar. There is no way the speller can determine the language in this case.

If the client allows the user to declare the language of the text, then replies to Get Data events should be of typeIntlText. This data type includes the country and script code along with the text. If the speller cannot understand the language then it should ignore the text or display a helpful message.

In this case it is the client's responsibility to specify each block of a particular language with a different object specifier. If a sentence is mostly English with a single word of German in the middle, then there will be three object specifiers given in the Batch event: one specifying the first part of the sentence, the second specifying the single German word, and a third to specify the remainder of the sentence.

While servers such as spellers are very language dependent, some servers may not care at all. For this reason, the client should request checking for all text blocks, and leave it up to the server to decide whether it wishes to deal with them.

Highlighting

When a speller suggests that a word be replaced, it is helpful to highlight the word in the original document so that the user may see the word in context with its original font and style.

A word processor can optionally support background highlighting by handling the pBackgroundHilite property of characters. If the property is true, then the character is highlit even when its window is in the background. If the property is false, then the character is not highlit when the window is in the background.

This property is analagous to the pFont and pStyle properties of characters - it may be applied to a range of characters to highlight a range of text.

The speller sends Set Data events to set background highlighting of ranges of characters to True or False. If the value is true, the specified characters are highlighted just as they are when selected by the mouse. An implied side effect of setting this property is that the highlit text is scrolled into view within the document window.

The normal way to highlight text is to set the user's selection just as if it had been selected with the mouse. A flag should exist in the code that highlights the selected text. In most applications, the updating code checks to see whether the window it accesses is in the front and, if it is not, the updating code does not display the selection.

If there are any characters with a pBackgroundHilite value of True, the selection is displayed. That is, setting any range of characters to true will select the characters and then set a global flag that causes the selection display code to still highlight while in the background.

An ideal implementation uses a two-stage highlight, similar to the way the Macintosh Programmer's Workshop (a development environment for writing Macintosh software) and MacApp (an object-oriented programming framework) do, with a dim outline while in the background and a regular highlight while in the foreground.

Support for background highlighting is optional. If the word processor does not support background highlighting, it returns an error code to the Set Data event. If the speller receives an error code in response to setting the background highlighting, it *must* display the questioned text within its own window.

If a client supports background highlighting, it may optionally also support the highlighting of disjoint ranges. This is helpful to grammar checkers, that may want to point out (for example) that two verbs disagree in tense by highlighting each of them.

If the client does not support disjoint ranges of highlighting, then a Set Data event that requests that a character be highlighted that is not contiguous with the current highlit range will unhighlight the original range. (This is the normal behavior of most word processing engines such as TextEdit). If it does support disjoint highlighting, then the original highlit range will remain. Thus it is the responsibility of the server request old ranges be turned off before a new one is turned on.

If a client supports disjoint ranges, then the pCanDisjointHilite property of its cApplication object is true. If it does not support them, then either its property is false, or attempting to read the property will return an error.

Text locking

A word processor may optionally support text locking. The lock property exists to make the Word Services suite more reliable. The lock property is not necessary as long as the user does not allow more than one application to access the document at the same time.

If a client supports text locking, a word processor must first resolve any object specifier that refers to a cText object, or a property or element of a cText object. Then the client checks whether a lock owner exists for the cText object.

The pLockTransactionID property holds the transaction ID for a transaction that has exclusive access to the cText object. The pLockTransactionID property has the value kAnyTransactionID if there is no lock owner. If a lock owner exists, the word processor should extract the transaction ID from the Apple event that is attempting to operate on the cText object. If the transaction ID does not match, the word processor should return errAEInTransaction rather than performing the requested action.

An implication of the pLockTransactionID property is that if its value is not kAnyTransactionID, the property cannot be set by an event whose transaction ID matches does not match its value.

A word processor might implement the pLockTransactionID property by keeping a list of locks, with the contents of the list being the transaction IDs and pointers to the cText objects. Because there are a few (usually zero or one) locks in existence, this method is easier than adding a transaction ID field to the data structure that implements the cText objects.

A speller must still operate on a word processor that does not support text locking. The speller can try to get data from the pLockTransactionID property. If an error is returned to the read, the pLockTransactionID property is not implemented and the speller goes on as if it had received the lock.

It is difficult to determine the value to use for the transaction ID. Begin Transaction cannot be used, because it locks out all other transactions. The speller can make up a value, but there is a risk that the value is not unique. A reasonable convention is for the speller to use the value of ticks at the time it sends the event to set the property as the transaction ID. The probability of two different applications sending the Set Data events at the same time is low.

Error handling

The Batch Process My Text event is a "Fire and Forget" technique. The client (a word processor of any sort) sends a single Batch Process My Text event and then continues its event loop, only responding to the Core events initiated by the server (spelling- or grammar-checker), which directs the rest of the process.

A client-interface batch mode spell-checking session initiated by a Fire and Forget method allows easy handling of error conditions. If the speller cannot continue, it should stop sending events to the word processor. If the word processor encounters an error, it should return error replies to the speller, which should then stop spell-checking. The worst possibility that can happen under these circumstances is that a range of text might stay locked.

In any case, the speller should stop its work if it receives an error reply from the word processor, unless the error is received from an attempt to carry out an optional operation, such as setting the background highlighting.

Read-Only documents

Read-Only documents present a special problem to Word Services servers. The server cannot change the text by sending Set Data events.

Client applications should still allow Word Services processing of read-only documents because some services will not need to change the text. An envelope addresser is an example of such a service.

Some documents may appear to be read-only because it is impractical to allow the text to be changed through Apple Events. A terminal emulator might display a document that is open in a text editor on a mainframe host. Although the document can be edited, such editors usually require keystroke commands to be typed by the user.

A reasonable way to handle this is to use the Check Word event to display a list of misspelled words in a separate window. The user may ask for guesses for each word in the list, and then type the corrections in herself.

The terminal emulator does this by sending each word in the window to the speller in its own Check Word event. Words that return "False" are kept in a window. If the user requests guessing for a particular word, then the terminal emulator will send a Check Word again, this time requesting some guesses, which it then displays.

Related implementation details

This section discusses the importance of range specifiers and methods for checking multiple blocks of text.

Checking multiple blocks of text

Most documents do not contain all of their text in a single block. Even a simple word processor document may have headers, footers, and footnotes. Databases and spreadsheets may have thousands of text blocks. Even with a single text flow, it is more efficient if the word processor can have it checked as a sequence of paragraphs rather than a single large text block.

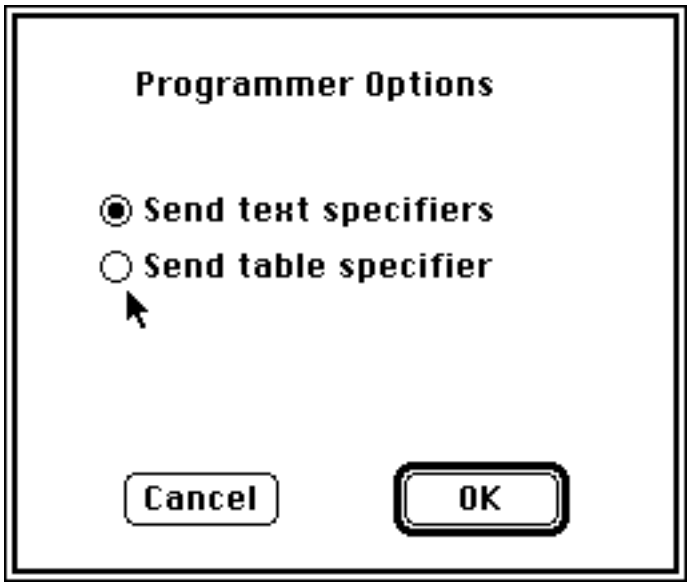
Word Services allows client applications to use two methods of specifying the text that is to be checked - the "Send Text Specfiers" method of the "Send Table Specifier" method.

The Send Text Specifiers method is simpler to use. It works well when there are a small number of separate text blocks, as in a word processor or drawing program. It does not work well when there are a large number of separate blocks because the parameter to the Batch Process My Text event will exceed the limit on the size of an Apple Event.

The Send Table Specifier method is more complicated to use but has no limitations on the number of text blocks that can be checked with one Batch Process My Text event.

The Word Services SDK includes a word processor that can be set to use either method. This is for the edification of programmers - the user should not normally see this. It is also useful for testing servers. Though Writeswell Jr. has just one text block, it can use either method by choosing a setting in the Programmer Options dialog:

■ **Figure 3** Programmer Options from Writeswell Jr.



Any client application *may* use one or both methods as appropriate. Any server application *must* support both methods.

The direct object to the Batch Process My Text Event may be one of two types. The server identifies the method used from the type of direct object that the client has sent.

The Send Text Specifiers method uses a typeAEList that contains one or more object specifiers for the text to be checked. *If there is only one text block to be checked, then the direct object must be a list with one object specifier as an element.*

The Send Table Specifier method uses a typeObjectSpecifier for the direct object. This object specifier refers to a table that is kept in the client application. The elements of this table are object specifiers to the text that is to be checked.

The main difference between these two methods is that the Send Text Specifiers is an explicit method, and the Send Table Specifier is implicit. The first case sends the specifiers directly in the Batch event. The second case informs the server that it must ask for them one at a time. The object specifiers that are in the table are the same as might otherwise be used as elements of the list in the Send Text Specifiers method.

One might ask why the table is used at all - why doesn't the server just ask for the first text block in the frontmost window, then the second, and so on. This is because the table contains specifiers for the blocks of text *that are to be checked*. This might not be all of the text blocks in the document. Also, the actual object specifiers are needed to construct subranges for the Set Data events when highlighting and replacing.

Client applications do not need to actually maintain an actual table of object specifiers in memory. There will likely be some list of text blocks that are to be checked. When an element of the table is requested by the server, the client will create the object specifier and send it back in the reply.

Once the server has an object specifier for a particular text block, it will use it the same way no matter what method was used to obtain it.

Range specifiers

The key to supporting the Word Services suite is to support formRange and end-of-container formAbsolute key forms. Specifying a position relative to the end of the container allows the text to be changed by several successive Set Data events without having to recalculate the new offset after every change. You can tell that a formAbsolutePosition key is relative to the end of the container because it is negative; -1 is the last object in the container. Figure 4 illustrates the concept of specifying the offset relative to the end of the container.

■ **Figure 4** Ranges with offset relative to the end of the container

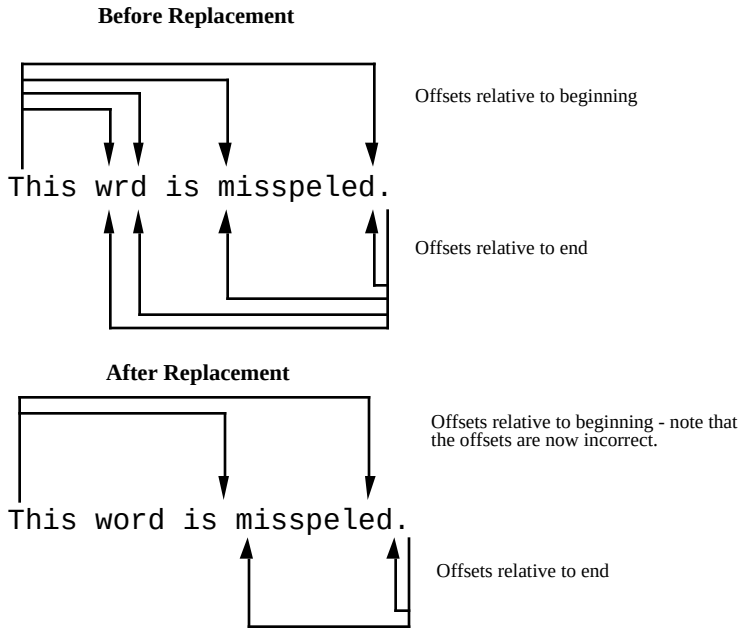
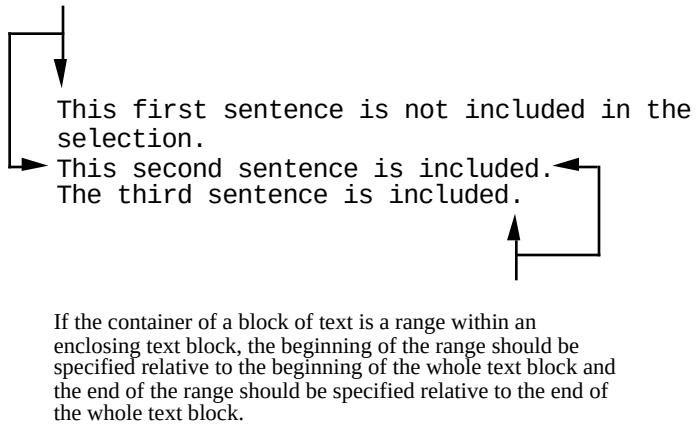


Figure 5 illustrates a text block that is a range within an enclosing text block. The text block container is a range whose start is relative to the beginning of the container and whose end is relative to the end of the container.

■ **Figure 5** Text block that is a range within an enclosing text block



The formRange key form is not clearly described in the Apple Event Manager chapter of *"Inside Macintosh: Interapplication Communication."* The documentation is unclear about how to resolve formRange specifiers.

Note that to specify a range of characters, you use a formRange object specifier that consists of two formAbsolutePosition specifiers, which specify a single character at the beginning and at the end of the range. When an object accessor receives a formRange key, the selectionData is a descriptor of type ‘rang.’ You can coerce this to typeAERecord, and then use the AEGetKeyDesc to extract the keyAERangeStart and the keyAERangeStop object specifiers. If you don’t coerce to typeAERecord, the call to AEGetKeyDesc fails because it cannot recognize that the ‘rang’ data type is really an AERecord. You must then call the AEResolve on each of the two specifiers to get the beginning and end of the range.

Note that this is a recursive call; your object accessors have already been called by AEResolve. Therefore, all object accessor must be reentrant and must not change any global variables. Also, object accessors must be conservative in their use of the stack; placing large amounts of data in local variables may cause the stack to collide with the heap.

Specifying sub-ranges of text

The word processor sends the object specifier, for one or more whole containers of text, to the speller. The speller adds descriptors to the original object specifiers to create new object specifiers, and sends them back to the word processor. The object specifiers must stay valid if the length of the text within them changes. Keeping object specifiers valid is not a problem if specifiers are sent for whole cText objects. However, if the user selects some arbitrary run of text within a document, the word processor cannot specify the selection by giving the whole range relative to the beginning of the document. The beginning of the range must therefore be given relative to the beginning of the document, and the end must be given relative to the end of the document: spell-check the text starting with the 29th character from the beginning of the first text field of window 'foo', and ending with the 37th character from the end of the first text field of window 'foo'.

Note that the object specifier sent by the word processor must be one that the word processor can itself resolve. The word processor can specify the object in any way it chooses; it can specify windows by name, location, or any other acceptable ID. The innards of the object specifier are not inspected by the speller.

Other facilities

This section discusses another facility of the Word Services suite; specifically, how to spell-check a single word.

Spell-check a single word

Script writers need a simple method to spell-check a single word and possibly get a list of guesses. This is also useful for applications that have read-only documents, such as terminal emulators, and for other purposes such as getting a list of synonyms to a given word.

To spell-check a single word, send the speller a Check Word event, with a typeText value in its direct object. The reply to this event has typeBoolean in its direct object, which is TRUE in value if the word is correct. Note that the reply has an error only when there was an error in handling the Apple event; if the word is misspelled, the reply has no keyErrorNumber parameter, and has a FALSE value in its direct object.

If the word processor wants to guess at correct spellings, it may use the Guess Word event. The keyWSGuessCount parameter is a typeInteger value that contains the maximum number of guesses desired. The speller might not be able to return as many guesses as are requested. The parameter can be left off entirely. Any guesses are returned in the keyAEData parameter to the reply as a typeAEList of typeText objects.

The actual behaviors of Check Word and Guess Word are not specified for applications that are not spelling checkers. A thesaurus, for example, might use Guess Word to supply synonyms as the "guesses" in response to each event. The behavior is up to each individual server.

Apple events defined in the Word Services suite

The Apple events defined in the Word Services suite are described in the following sections. Table 1 lists these Apple events.

Table 1 Apple events defined in the Word Services suite

Apple event name	Requested action
Batch Process My Text	Apply a word service to one or more blocks of text
Check Word	Spell-check a single word
Guess Word	Get guesses to a misspelled word
Check Word Interactively	Check a word during an interactive session
Interactively Process Text	Start an interactive spelling session
Query Replace	Replace text upon user confirmation

Batch Process My Text—apply a word service to one or more blocks of text

This event specifies one or more blocks of text for a word service. The event does not actually send the text to the server, but instead sends either a single object specifier for a cText object or a list of several object specifiers. The keyDirectObject parameter allows the text to be specified in two ways: either by sending object specifiers for the text explicitly, in which case the direct object to the batch event is a list (a descriptor of typeAEList) that contains a single element, or by sending an object specifier for a table, in which case the direct object is a descriptor of typeObjectSpecifier. After receiving the Batch Process My Text event, the Word Services server uses the Get Data event to get the text, and the Set Data event to replace the text in the document window. The server also attempts to use a Set Data event to highlight text in the client's window. If this event fails, the server shows the text in its own window.

Event Class kWordServicesClass

Event ID kWSBatchCheckMe

Parameters

keyClientAddress

Description:	This identifies the client word processor. If the keyClientAddress parameter is present, its value is used as the address of the word processor. This is meant to allow a third program, such as a scripting application, to instruct the speller to check some other application's document.
Descriptor Type:	typeTargetID
Required or Optional?	Optional

keyDirectObject

Description:	This specifies the text to be checked. There are two ways that the text can be specified. If the descriptor is of typeAEList, it must be a list of items of typeObjectSpecifier. Each object specifier is expected to resolve to an object of class cText in the *client's* own application; the server does not resolve the object. Instead, it sends the object specifier back to the client. If the keyDirectObject parameter is an object specifier, it is an object specifier to a list of object specifiers that is maintained by the word processor. Each element in this list specifies a cText object that is to be spell-checked. The spell-checker can use the object specifier as a container to request the first element of the list by asking for the typeObjectSpecifiers that are

contained in the list. The first element is always specified using formAbsolutePosition (give me the first element). Succeeding elements are specified using formRelativePosition (give me the next element) or formAbsolutePosition.

Descriptor Type:	typeObjectSpecifier or typeAEList
Required or Optional?	Required

keyFacelessMode

Description:	This specifies that the client-interface mode is selected. If the keyFacelessMode parameter is TRUE, then spell-checking is done in client-interface mode. The word processor should use a unique transaction ID for the batch event; this transaction ID is used by the subsequent Query Replace events. (Do not set the kAENeverInteract flag in the batch event. The speller might need to do some interaction, such as prompting the user to locate a dictionary file.)
Descriptor Type:	typeBoolean
Required or Optional?	Optional (default value: false)

Reply Parameters

keyErrorNumber

Description:	The result code for the event.
Descriptor Type:	typeLongInteger
Required or Optional?	Optional (The absence of a keyErrorNumber parameter in the reply indicates that the event was handled successfully.)

Result Codes

errAEEventFailed	–10000	The Apple event handler failed when attempting to handle the Apple event.
errAEInTransaction	–10011	Could not handle this Apple event because it is not part of the current transaction.

Notes A client application can choose either of the two options to specify the text (that is to be checked) in the keyDirectObject parameter. The options are referred to as the "Send Table Specifier" and "Send Text Specifiers" methods of specifying text in the above section "Checking multiple blocks of text".

Check Word—spell-check a single word

This Apple event requests spell-checking for a single word. The action is not specified if the application that receives this event is not a speller. There are other uses for the Check Word event, in addition to performing a spell-check. One example is looking up synonyms in a thesaurus.

Event Class kWordServicesClass

Event ID kWSCheckWord

Parameters

keyDirectObject

Description:	This is the text of a single word to be checked.
Descriptor Type:	typeIntlText
Required or Optional?	Required

keyWSGuessCount

Description:	This is the maximum number of guesses that are desired if the word is incorrect.
Descriptor Type:	typeInteger
Required or Optional?	Optional (If it is not present, a default value of 0 is used.)

Reply Parameters

keyAEDirectObject

Description:	Is TRUE if the word was correct or FALSE if it was not.
Descriptor Type:	typeBoolean
Required or Optional?	Required

keyErrorNumber

Description:	The result code for the event.
Descriptor Type:	typeLongInteger
Required or Optional?	Optional (The absence of a keyErrorNumber parameter in the reply indicates that the event was handled successfully.)

Guess Word—Get Guesses for a Misspelled Word

This Apple event requests guessing for a single word. An optional parameter to the Guess Word event gives the maximum number of guesses to the word that can be returned as a list in the reply. The action is not specified if the application that receives this event is not a speller. There are other uses for the Guess Word event, in addition to performing a spell-check. One example is looking up synonyms in a thesaurus.

Event Class kWordServicesClass

Event ID kWSGuessWord

Parameters

keyDirectObject

Description:	This is the text of a single word to be guessed.
Descriptor Type:	typeIntlText
Required or Optional?	Required

keyWSGuessCount

Description:	This is the maximum number of guesses that are desired if the word is incorrect.
Descriptor Type:	typeInteger
Required or Optional?	Optional (If it is not present, a default value of 0 is used.)

Reply Parameters

keyAEDirectObject

Description:	This contains a list of possible replacements for an incorrect word.
Default Descriptor Type:	typeAEList (a list of typeText objects)
Required or Optional?	Reguired. . If any guesses were requested but none were available, the keyAEResult parameter will be an empty list. Thus, this parameter has the default value of an empty list. Even though the parameter is present, it can have fewer guesses than were requested.

keyErrorNumber

Description:	The result code for the event.
Descriptor Type:	typeLongInteger
Required or Optional?	Optional (The absence of a keyErrorNumber parameter in the reply indicates that the event was handled successfully.)

Check Word Interactively—check a word during an interactive session

This Apple event sends words to a server that has already received the Interactively Process Text event and is prepared to receive words and check them as they arrive.

Event Class kWordServicesClass

Event ID kWSCheckInteractive

Parameters

keyAEData

Description:	The actual text of the word that is sent.
Descriptor Type:	typeIntlText
Required or Optional?	Required

keyDirectObject

Description:	A typeObjectSpecifier that specifies the word that is sent, relative to the *beginning* of its container. It must not be sent relative to the end, because the end can change as the user types.
Descriptor Type:	typeObjectSpecifier
Required or Optional?	Required

Reply Parameters

keyErrorNumber

Description:	The result code for the event.
Descriptor Type:	typeLongInteger
Required or Optional?	Optional (The absence of a keyErrorNumber parameter in the reply indicates that the event was handled successfully.)

keyErrorString

Description:	A character string that describes the error, if any, that occurred when the event was handled.
Descriptor Type:	typeIntlText
Required or Optional?	Optional

Interactively Process Text—start an interactive spelling session

This Apple event requests the speller to start an interactive text-checking session. No data is sent in this event; it is used to ensure that the speller is both available and capable of interactive spell-checking. If a reply is received with no error code, the speller can start sending words for processing with the Check Word Interactively Apple event.

Event Class kWordServicesClass

Event ID kWSStartInteractive

Parameters

keyFacelessMode

Description:	This parameter specifies that client-interface mode is used. If the keyFacelessMode parameter is TRUE, spell-checking is done in client-interface mode. The word processor should use a unique transaction ID for the event; this transaction ID is used by the subsequent Query Replace events. Do not set the kAENeverInteract flag in the event. The speller might need to do some interaction, such as to prompt the user to locate a dictionary file.
Descriptor Type:	typeBoolean
Required or Optional?	Optional (default value: FALSE)

Reply Parameters

keyErrorNumber

Description:	The result code for the event.
Descriptor Type:	typeLongInteger
Required or Optional?	Optional (The absence of a keyErrorNumber parameter in the reply indicates that the event was handled successfully.)

keyErrorString

Description:	A character string that describes the error, if any, that occurred when the event was handled.
Descriptor Type:	typeIntlText
Required or Optional?	Optional

Query Replace—replace text upon user confirmation

The speller sends this event to inform the word processor that a range of text might need replacement.

Event Class kWordServicesClass

Event ID kWSQuery Replace

Parameters

keyAEData

Description:	This is a list of possible replacements for the identified text. There can be zero, one, or more items in the list.
Descriptor Type:	typeAEList (of typeIntlText objects)
Required or Optional?	Optional (default value: a list of one null string)

keyDirectObject

Description:	This is an object specifier for the text that is questioned by the speller. The word processor should highlight this text in the document window and scroll it into view in such a way that it is visible behind any dialog it might show.
Descriptor Type:	typeIntlText
Required or Optional?	Required

keyErrorString

Description:	This is a human readable string that explains what sort of error in the text has been found. For example, “Incorrect spelling,” or “Split infinitive.”
Descriptor Type:	typeAEList (of typeIntlText objects)
Required or Optional?	Optional (default value: a list of one null string)

keyTransactionIDAttr

Description:	This is the transaction ID that was supplied in the Batch Process My Text or Interactively Process Text event. The transaction ID is supplied as an argument to the AECreateAppleEvent system call.
Descriptor Type:	typeLongInteger
Required or Optional?	Required

Reply Parameters

keyErrorNumber

Description:	The result code for the event.
Descriptor Type:	typeLongInteger
Required or Optional?	Optional (The absence of a keyErrorNumber parameter in the reply indicates that the event was handled successfully.)

keyErrorString

Description:	A character string that describes the error, if any, that occurred when the event was handled.
Descriptor Type:	typeIntlText
Required or Optional?	Optional

Object classes defined in the Word Services suite

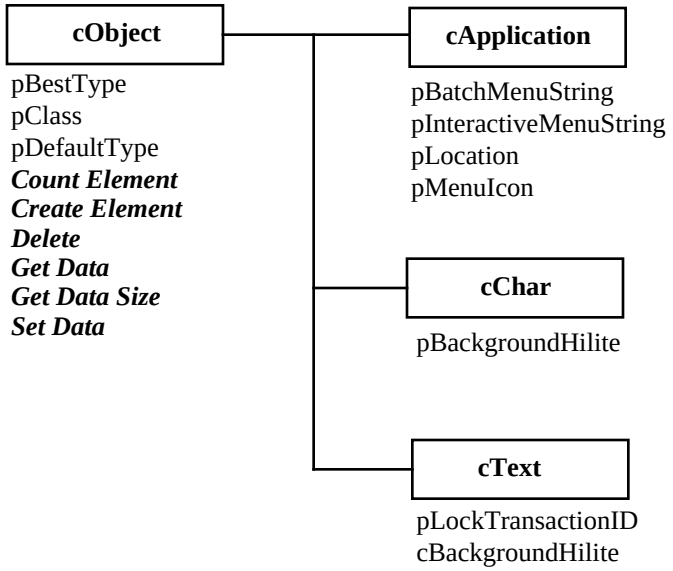
The Apple event object classes defined in the Word Services suite are described in the following sections. Table 2 lists these object classes.

Table 2 Apple event object classes defined in the Word Services suite

Object class ID	Description
cApplication	A standard Macintosh application *Properties:* pBatchMenuString, pInteractiveMenuString, pLocation, pMenuIcon, pColorMenuIcon, pCanDisjointHilite *Element Classes:* none
cChar	Text characters *Properties:* pBackgroundHilite, pBestType, pClass, pDefaultType *Element Classes:* none
cText	A series of characters *Properties:* pBestType, pClass, pDefaultType, pLockTransactionID, pBackgroundHilite

Figure 6 illustrates the inheritance hierarchy for the object classes defined in the Word Services suite. For each object class, a list is provided of the properties, element classes, and Apple events that have not been inherited from object classes higher in the inheritance hierarchy.

■ **Figure 6** Object inheritance hierarchy for the Word Services suite



cApplication—a standard Macintosh application

The cApplication object class is an extension to the existing cApplication object class. Five properties have been added. The first properties contain text that a word processor can display in its menu to list the Batch and Interactive services provided by a speller. The third property provides an alias record to the application so that it can be launched again. The following two properties allow applications to make different menu items more distinguishable. The sixth property allows a server to ask a client if it supports disjoint background highlighting. Not all of the Core suite cApplication properties are listed here, because some of them do not have any relevance to the Word Services suite.

Some of these properties are possessed by servers so that clients may query them. Others are possessed by clients for servers to query.

Superclass cApplication (Core suite)

Default Descriptor Type typeIntlText

Properties

pBatchMenuString

Description:	This property is a text string suitable for display as a menu item. It describes the service that is performed by the speller in response to receiving a Batch Process My Text event. The value of the property is a string such as "Check Spelling", "Check Grammar", or "Translate to Greek". This is a property of a server.
Object Class ID:	cIntlText
Inherited?	No
Modifiable or Nonmodifiable?	Nonmodifiable

pInteractiveMenuString

Description:	This property is a text string suitable for display as a menu item. It describes the service that is performed by the speller in response to receiving an Interactively Process My Text event. The value of the property is a string such as "Check Spelling Interactively", "Check Grammar Interactively", or "Translate to Greek on the Fly". This is a property of a server.
Object Class ID:	cIntlText
Inherited?	No
Modifiable or Nonmodifiable?	Nonmodifiable

pLocation

Description:	This property is an alias record that the user can save and use later to launch the application again. The application must set the userType field of the alias record to its own signature. (The userType is set to 0 when the Alias Manager creates the alias record; the value that can be stored there is left up to the application, and is not interpreted by the Alias Manager). Storing the signature in this field allows the application to receive Apple events by creator code without having to manually resolve the Alias Manager record, and then call GetFileInfo to get the application file's Finder information. It also allows client applications to use the Process Manager to see if the server is already running. This is a property of a server.
Object Class ID:	typeAlias
Inherited?	No
Modifiable or Nonmodifiable?	Nonmodifiable

pMenuIcon

Description:	The value of this property is a small icon that is placed in the menu along with the interactive or batch menu strings. It is identical to the small icon that the speller shows in the Finder. Word processors that take advantage of this use it to make the different menu items more distinguishable. This property can be useful in cases in which a user has two different spellers. This is a property of a server.
Object Class ID:	typeSmallIcon
Inherited?	No
Modifiable or Nonmodifiable?	Nonmodifiable

pColorMenuIcon

Description:	The value of this property is a small color icon that is placed in the menu along with the interactive or batch menu strings. It can be displayed with the same code the displays the monochrome small icon. The Menu Manager will use the color version if it is present when the menu is displayed on a color monitor. This is a property of a server.
Object Class ID:	typeColorIcon
Inherited?	No
Modifiable or Nonmodifiable?	Nonmodifiable

pCanDisjointHilite

Description:	The value of this property is a true if the application is a Word Services client that can highlight more

		than one separate range of text. This is useful to some servers such as grammar checkers that may wish to point out (for example) that two verbs in a paragraph disagree in tense. This is a property of a client.
	Object Class ID:	typeBoolean
	Inherited?	No
	Modifiable or Nonmodifiable?	Nonmodifiable

Element Classes None

Apple Events

Apple events from the Core suite:

Get Data	Inherited from cObject
Get Data Size	Inherited from cObject

cChar—text characters

The cChar object class has a property that highlights a range of characters even when they are in a window that is in the background. The cChar object class is an extension of the cChar object class defined in the Core suite.

Superclass cText (Core suite)

Default Descriptor Type typeIntlText

Properties

pBackgroundHilite

Description:	Indicates whether the character is to remain highlighted while the containing window is in the background. If pBackgroundHilite is TRUE, the character stays highlighted while a window is in the background. Setting this property to TRUE also implies that the character is scrolled into view within its window.
Object Class ID:	typeBoolean
Inherited?	No
Modifiable or Nonmodifiable?	Modifiable

Element Classes None

Apple Events

Apple events from the Core suite:

Get Data	Inherited from cObject
Set Data	Inherited from cObject

Notes The pBackgroundHilite allows a Word Services server to highlight a suspected word in the original document with all its font, style, and other attributes, rather than showing the text in its own window. Ordinarily, this property is implemented by setting a text selection, with a special case that leaves the highlighting of the selection on when the window is moved to the background. It is not necessary for an application developer to change the user selection to show the highlighting (the highlighting is for display purposes only). Instead, the user selection can be preserved and implemented in a different way.

Support for this property is optional. If a word processor does not support this property, it should return an error code when the speller tries to set it. The speller must then display the text in its own window. Support for more than one range of highlighted text is also optional. If the word processor cannot show disjointed selections, it should turn off any existing highlighting when the property is set to TRUE on a new range of text.

The server can determine whether disjoint ranges are supported by trying to set the property for two ranges and then using the Get Data event to obtain the value of the property for each range. If disjointed ranges are not supported, the value of the property is FALSE for the first range that was set. The server must set the property to FALSE when it has finished examining a particular range of characters.

Note that you set this element by using a Set Data event, where the data to be set has a Boolean value. The direct object is a formRange specifier that specifies a range of characters within the cText object to be highlighted.

There are additional properties and Apple events defined in the Core suite that can be used by an application that supports the Word Services suite. Only the essential properties and Apple events required by an application to support the Word Services suite are listed here.

cText—a series of characters

This is an extension of cText object class, as defined in the Core suite. A text locking property has been added, which allows a transaction to have exclusive access rights to an object.

Superclass cObject (Core suite)

Default Descriptor Type typeIntlText

Properties

pBestType

Description:	The descriptor type that contains the most information from objects of this object class.
Object Class ID:	cType
Inherited?	Yes, from cObject
Modifiable or Nonmodifiable?	Nonmodifiable

pClass

Description:	The four-character class ID for the object class.
Object Class ID:	cType
Inherited?	Yes, from cObject
Modifiable or Nonmodifiable?	Nonmodifiable

pDefaultType

Description:	The default descriptor type for the object class.
Object Class ID:	cType
Inherited?	Yes, from cObject
Modifiable or Nonmodifiable?	Nonmodifiable

pLockTransactionID

Description:	The pLockTransactionID property holds the transaction ID for a transaction that has exclusive access to the object. If an event with a different ID attempts to read, write, or modify the object, the word processor should return errAEInTransaction as a result. If there is no lock owner, the property has the value kAnyTransactionID.
Object Class ID:	typeLongInteger
Inherited?	No
Modifiable or Nonmodifiable?	Modifiable

Apple Events

Apple events from the Core suite:

Get Data	Inherited from cObject
Set Data	Inherited from cObject

Notes

There are additional properties and Apple events defined in the Core suite that can be used by an application that supports the Word Services suite. Only the essential properties and Apple events required by an application to support the Word Services suite are listed here.

Primitive object classes defined in the Word Services suite

Table 3 lists the primitive Apple event object classes (classes with no properties and only one element) defined in the Word Services suite.

Table 3 Primitive object classes defined in the Word Services suite

Object class ID	Descriptor type of element	Description
(There are no primitive object classes currently defined in the Word Services Suite)		

Key forms defined in the Word Services suite

Table 4 lists the key forms defined in the Word Services suite. The italicized words in each example correspond to the key (the portion of the object specifier record that distinguishes an object from other objects of the same class in the same container). For more information about keys and key forms, see the Apple Event Manager chapter of *"Inside Macintosh: Interapplication Communication."*

Table 4 Key forms defined in the Word Services suite

Key form constant	Description
formAbsolutePosition	Specifies the position of an element in relation to the beginning or end of its container (for example, "word *5* of . . . "), or specifies one or more elements with a constant defined in the Apple Event Manager chapter of *"Inside Macintosh*: *Interapplication Communication,"* such as kAEFirst (for example, "the *first* word in paragraph 12 . . . ") or kAEAll (for example, "*all* the words in paragraph 12 . . . "). Note that the end-of-container position is required.
formName	Specifies an element by its name (for example, "the document *named 'MyDoc' "*).
formPropertyID	Specifies a property of an object by its four-character property ID (for example, *"the font* of word 1").
formRange	Specifies a list of elements between two other elements (for example, "the words *between 'Wild' and 'Zanzibar,' inclusive*").
formRelativePosition	Specifies an element immediately before or after a container (for example, "*the next* word *after* the words whose style is bold").
formTest	Specifies one or more elements that pass a test; values of one or more properties or elements are tested (for example, "the first paragraph *that is centered and that begins with the word 'Wild' "*).

Constants defined in the Word Services suite

Table 5 lists the constants defined in the Word Services suite.

Table 5 Constants defined in the Word Services suite

Constant	Value
keyClientAddress	'Cadr'
keyWSGuessCount	'Gcnt'
kWordServicesClass	'WSrv'
kWSBatchCheckMe	'Btch'
kWSCheckInteractive	'CkIn'
kWSCheckWord	'CkWd'
kWSGuessWord	'Gess'
kWSQueryReplace	'QRep'
kWSStartInteractive	'SInt'
pBackgroundHilite	'pBgH'
pBatchMenuString	'pBMs'
pInteractiveMenuString	'pIMs'
pLocation	'pALc'
pLockTransactionID	'pLID'
pCanDisjointHilite	'pDjH'

Index