The Word Services Suite

The Word Services suite defines Apple event constructs for such services as spell checking and grammar checking. The suite can be used for any service that needs to examine and change the text in a document. The suite should be supported by applications that allow text to be edited by a user, such as word processors, drawing programs, spreadsheets, or electronic mail programs, as well as spell checking, grammar checking, bibliography and hyphenation programs.

By: Michael D. Crawford

Working Software, Inc.

(408) 423-5696

AppleLink D1620

February 20, 1992

Introduction to the Word Services suite
The Apple Event Word Services Protocol is a protocol to allow any application to have its text inspected and modified by programs that operate on text such as spelling checkers or grammar checkers. It is designed to be a simple extension of the Core Suite which word processors will find easy to implement.

"Word Processor" in this document means any application that allows text editing, not just programs sold as word processors. The word processor is a client in this protocol.

"Speller" here refers to spelling and grammar checkers, or any other program that acts as a server in this protocol.

Overview of the Word Services suite

The Word Services suite defines Apple events to

- Apply a word service to one or more blocks of text.
- Replace text upon user confirmation.
- Spell check a single word.
- Start an interactive spelling session.
- Send text to a server for interactive checking.
- Check possible errors during interactive checking.

The Word Services suite extends object classes for

- cText
- cChar
- cApplication

Applications that should support the Word Services suite

The following types of applications should support the Word Services suite:

- Spelling checkers
- Grammar checkers
- Hyphenators
- Natural language translators
- Indexers

- Bibliography programs
- Any application that can take text from a document and change it.

Typical client applications for the Word Services suite

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

- Word Processors
- Page Layout applications
- Database
- Spreadsheet
- Electronic Mail
- Graphics
- Personal Information Management
- Any application that allows a user to edit text

Important Information About This 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, particular the GetData and SetData events, the formRange key form, and the formAbsolutePosition key form, including positions relative to the end of the container.

All a word processor need do to get its text spellchecked is to send a single BatchProcessMyText event with one or more object specifiers for the text to be checked, then continue its event loop, in which the word processor will respond to events sent by the speller.

Usage scenarios

There are four different basic scenarios - *faceful batch*, *faceful interactive*, *faceless batch*, and *faceless interactive*. One may also request that single words be checked directly.

There are two main styles of operation. In the faceful style, the speller provides the user interface. Misspelled words are presented in a dialog drawn by the speller, perhaps with a list of guesses, and the user may be allowed to skip or replace the misspelled word. If the word processor supports it, the speller may request that the misspelled word be highlighted in the word processors own window, thus displaying the word in its own style and in the context of the whole document.

If background highlighting is not supported, then the speller will display the word in its own window, possibly with some simple formatting.

Faceless spellers act as background servers. The word processor will send some text to the speller, which will send back a report listing which words were misspelled. The word processor will then draw its own dialog, and may request lists of guesses from the speller.

In either style of operation, one can have two modes of spellchecking. Batch mode will spellcheck a range of text (typically the current selection, or the whole document) upon an explicit user command. Interactive checking does spellchecking as the user types. The word processor must parse word breaks and send whole words to the speller. If the word is correct, the speller does not respond. If the word is incorrect, an error signal of some sort is given (usually a SysBeep). The user may ignore the signal (either because she knows the word is correct, or she just deletes and corrects the word herself), or bring up a dialog to do the replacement.

Note that the interactive mode can be slow. The time it takes to generate the Apple event, do a Multifinder switch, receive the event, look up the word in the dictionary and switch back may be unacceptable to the user, especially if the text is sent a character at a time.

For this reason interactive checking will normally be done asynchronously – the word processor will send the Apple event, but will continue without switching out. During the next normal switch, the speller will start the word look up, but should allow itself to be switched back out before it is done if the look up takes too long (it will get several opportunities to run for each word, as each character typed will return from a separate WaitNextEvent call).

Both programs should be able to deal with the cases of the user typing some text after the misspelling before reacting to the signal, typing another incorrect word while one is pending, moving the mouse and typing bits of words in different places, backspacing, clicking the mouse in the middle of a word and changing it – for the most part, the speller will just ignore the word, but a fancier speller could try to queue up mistakes, or request the whole word that the cursor just left.

Performing Faceful Batch Mode Checking

The user will select some text to be checked. This may be done by shift-clicking several cells in a spreadsheet or text fields in a drawing, or highlighting a range of text in a word processor, or selecting "Check Whole Document" from the menu.

The word processor will create an object specifier for each text field to be checked. 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 (basically, it must be something that can contain paragraphs or characters. Also if one does a GetData on the object itself, all of the text within it will be returned).

The word processor will check to see if it has saved the location of the speller (that is, an alias record). The speller may be currently running; if it is, then the word processor will see its creator code in the list of PPC ports by using the IPCListPorts function. If it is not, then it can launch 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.

If the word processor has not saved the speller’s location (or cannot find it) the word processor will locate the speller with the PPCBrowser. This location can then be saved so this step can be avoided in the future.

The word processor can optionally use Get Data to get the pBatchMenuString and pInteractiveMenuString from the speller, and save them to add to its menu. The word processor can also use Get Data to get the pLocation property from the speller - this is an alias record that it can save in its Preference file to associate with the menu strings. These properties may or may not be defined for each server implementation.

The word processor will send the speller a "Batch Process My Text" event. The keyDirectObject parameter to the event will either be a single object specifier or a list of several object specifiers (or possibly some kind of specifier that indicates that it is the head of a list of object specifiers that is kept by the word processor). Once the reply is received the word processor will resume its normal event processing.

Note that this is the key to easy implementation of the protocol for word processors - they need only send one event, then forget about it and handle events normally. Of course, they must support the necessary events, classes, and key forms from the Core Suite.

When the speller receives the "Batch" event it will record the object specifiers and the address of the sender, then return a reply indicating that there is no error. If the speller is busy, it will send errAEInTransaction as an error code in the reply.

The speller will then make itself the foreground application by calling AEInteractWithUser. The word processor will become the second from the front.

The speller will then display a dialog (or some other sort of user interface). The dialog 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 will Set Data on pLockTransactionID for the object. This is to prevent other Apple Events from changing the text while it is being worked on.
>
> The speller will make new object specifiers from the original. The new object specifiers will specify paragraphs or ranges of characters within the original.
>
> The speller may send Get Data Size events to find the amount of text within the object. Note that Apple Events may not contain more than 64K of data, and a particular speller may have a limited amount of memory.
>
> The speller will send Get Data events to get the text. It may get the whole text object, or paragraphs within it, or ranges of characters.
>
> The speller may send Create Element and Set Data events to create and set the cBackgroundHilite elements in the word processor's application object. The value of the cBackgroundHilite is a range of characters (objects of class cChar). If there are no cBackgroundHilite elements then there is no text shown highlighted in the background. If there is a range then the specified characters are highlighted just as they are when selected by the mouse. The normal way to do this would be to set the user's selection just as if they dragged the mouse over it, and have a flag in the code that displays the selection. In most applications, the updating code checks to see if the window is in the front, and if it is not, does not display the selection. If there is one or more cBackgroundHilite objects, then the selection is displayed. An ideal implementation will use a two stage hilite similar to the way MPW and MacApp do, with a dim outline while in the background and a regular hilite while in front.

The speller will send Delete events to remove cBackgroundHilite elements when it is done. It need not delete and create each element each time it hilights a new word; instead it can use Set Data to change the value of an existing element.

(If the word processor does not support background highlighting, it should return an error code to the SetData event. If the speller receives an error code, it should create a window to display the text itself. Also, it should allow the user to choose whether the text is shown in the speller or the word processor.)

The speller will send SetData events to set a range of characters when it needs to replace them. The range specifier will consist of two object specifiers for single characters. Each character specifier will use formAbsolute position to give the offset **from the end of the container**.

The speller will Set Data on the propTransactionID property of the text field, setting it to kAEAnyTransaction. This allows the object to be accessed by any other events.

When the speller is done, it will make the word processor the foreground application again.

This scenario illustrates several key points:

1. To operate the protocol, the word processor need only send one Apple Event on its own initiative. The whole operation will involve an extended sequence of apple events and replies, but the logic to drive this is entirely in the speller. The word processor need only respond passively to the Apple Events. There is no "spellchecking mode" that it needs to enter, with the exception that if a cBackgroundHilite element gets set, the screen updating code needs to show the selection while in the background, preferably in a "dim" hilite mode similar to MPW.

2. The particular service that is performed on the text depends on which server that the word processor connects to. This may be a grammar checker, a spelling checker, or some other sort of program. The word processor should allow each server its own menu item (possibly two, if it supports both interactive and batch mode), and should store the location of each.

3. If there is no saved speller location, the user must launch the speller before starting the session. Most likely a naive user will need to cancel the PPCBrowser, launch the speller from the Finder, switch back to the word processor, and select "check spelling" again. It sure would be great if there were a way to register the kind of service an application provides.

4. The formRange key form is not very clearly described in the Object Support Library documentation. When an object accessor is given a formRange, it must coerce the selection data to typeAERecord. The data is of type 'rang', which is identical to typeAERecord, but AEGetKeyDesc does not recognize 'rang'. (The Object Support Library apparently contains a coercion handler for this purpose). Use AEGetKeyDesc on the coerced selection data with the keywords keyAERangeStart and keyAERangeStop to get the object specifiers for the beginning and ends of the range. You must then call AEResolve with each specifier to get a token for the beginning and end of the range - note that this is a recursive call to AEResolve, and therefore a recursive call to the object accessor itself.

5. There is a problem when creating the first cBackgroundHilite object. Note that there are no initial elements to give the location to insert the new element at. Instead, one passes the object specifier for the cText object that is the container, and the keyAEPosition field contains the value kAEBeginning. Subsequent items can be created by using the constants kAEEnd, or kAEAfter.

Performing "Predatory" Faceful Batch Checking

In this mode, the user launches the speller, and then opens a word processor document from the file menu of the speller. The speller will launch the application that created the document, and use Get Structure events to locate the text fields within the document, and then use Get and Set Data as above to check and replace the text.

This is primarily meant to allow spellchecking for 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 can be processed by a Word Services speller. This also would allow a user to drag a document icon onto the speller in the Finder. The speller's response to an ODOC event will be to locate the creator have it open the document.

For predatory checking to work, the speller will have to explore the word processor's object in order to locate the text items. This would be fairly simple for a simple text editor window, but could be quite complicated for a highly structured document.

Performing Faceful Interactive Checking

The user will request that interactive checking be turned on. Note that a word processor could have a preference setting to turn interactive checking on automatically when the word processor is launched.

The word processor will locate the speller as above, and send it an "Interactively Process Text" event, and await a reply. If reply is received without an error, the word processor will send each word to the speller as the user completes typing it in a "Check Word Interactively" event. (In general, typing a word break character such as a space or punctuation should cause a word to be sent).

Every separate word should be sent. This includes any punctuation, as the speller may have the ability to check capitalization after periods, or may be a grammar checker that will rigorously check punctuation.

Each word is sent asynchronously, with no reply requested. It may take some time for the speller to check the word; waiting for replies would cause obnoxious pauses. Not using replies will reduce overhead. If the speller wishes to complain of some error it should do so on its own with the notification manager.

Each event sent to the speller will contain the following parameters:

- keyAEDirectObject - a typeObjectSpecifier that specifies the word that is sent, relative   to the *beginning* of the container.
- keyAEData - the actual text of the word that is sent

If the speller finds an error it should make some noise 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 as the user may just ignore the error (it might be a correct word that is not in the dictionary) or correct it herself.

The user may then select "Check Word" from the word processor's menu. If there is a text selection, the word processor must use the "Batch Process My Text" event to check the whole selection. (The user may not be checking the last word that was beeped at). If there is no selection, the word processor must send the "Process Last Error" event to the speller. (Alternatively, if there is a selection, the "Check Word" item may be greyed out, so the user is forced to choose the "Check Spelling" or "Check Grammar" items).

Upon receiving the "Process Last Error" event the speller will send a Get Data event back to the word processor, to get the text of the erroneous word again. This is important because something may have occurred to invalidate the object specifier for the word (text may have been added before the word, or the user may 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 the text does match, then the speller should show its dialog. If the user wants to replace a word, the speller will use Set Data to do the replacement.

The speller could also just replace words automatically when it finds an error, without consulting the user, by doing the verification when it finds the error, then using Set Data to do the replacement.

Note that the object specifier sent 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 is changing as the user types.

If the word is verified, then the speller may display its dialog, and get a command to replace the word from the user, which it does by using Set Data.

Performing Faceless Interactive Checking

This is basically similiar to the faceful interactive scenario, except that when the user requests that a questioned error be looked up, a Query Replace event will be used to allow the user to decide whether to replace the text.

Performing Faceless Batch Checking

This scenario is basically similiar to the faceful mode. However, the speller does not send events to hilight and replace the text directly. Instead, it sends "Query Replace" events to the word processor, which include an object specifier for the text in question, a list of replacement strings, and a text string that explains what the message is.

Note that the word processor *must* allow the user to type in her own replacements, and that the text that needs replacing could extend outside the range which is specified. This is because the error may be an error in grammar that ranges over the whole paragraph, and the correction that is needed may be different from what the grammar checker guesses.

If this is too complex, a reasonable solution would be to just cancel out of the "Query Replace" dialog to have the user replace the text in the document. The user could then select the remaining text in the document and have it checked.

Spell checking Single Words

Script writers may want to have a very simple means to spellcheck single words, and possibly get a list of guesses back.

To do this, send the speller a "Check Word" event, with a typeText value in its direct object. The reply to this event will have a typeBoolean in its direct object, which will be "true" in value if the word is correct. Note that the reply should only have an error if there was an error in handling the Apple Event; if the word is incorrect, the reply will have no keyErrorNumber parameter, and will have a "false" value in its direct object.

If the word processor wants some guesses for correct spellings, it should add a keyWSGuessCount parameter to the Check Word event. This will be a typeInteger value that will contain the maximum number of guesses or desires. (The speller may not be able to return as many as are requested). The value may be zero, or the parameter may be left off entirely if guesses are not desired. Any guesses will be returned in the keyAEData parameter to the reply as a typeAEList of typeText objects.

Note that guessing can be time consuming. The user should have an option to have guessing on every error, or only on request. If a word is incorrect, and the user requests a guess, then the same word should be sent again.

Assumptions

Handling Exceptions

The "Fire and Forget" method of starting a faceless batch mode spellchecking session allows error conditions to be handled easily. If the speller cannot continue, it should just 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 spellchecking.

The worst that would happen here 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 do an optional operation, like setting the background highlighting.

Requirements for Supporting Word Services

The word processor *must* support *at least* a minimal set of Core Suite Apple Events and object model types. (It *should* support them all for other purposes, though).

The Apple Events it must support are

- Get Data Size
- Get Data
- Set Data
- Create Element (to create the cBackgroundHilite objects)
- Delete (to delete the cBackgroundHilite objects)

The key forms that it must support are

- formPropertyID
- formAbsolute position, including
    - positions relative to the beginning of the container,
    - positions relative to the end of the container

- formRange

The key form that it must support if the "list head" specifier is used is:
- formRelativePosition

The properties that it should support (recommended, but optional) are:

- pColor
- pFont
- pPointSize
- pTextStyles

These are all properties of the cChar class, defined in the Text Suite. A word processor that did not allow formatting (such as a simple spreadsheet or database) would not be expected to support them, but regular word processors and page layout programs would be.

The word processor will send the object specifier for one or more whole containers of text to the speller. The speller will add descriptors to the original object specifiers to make new ones, and send them back to the word processor. The object specifiers must stay valid if the length of the text within them changes. This is not a problem if specifiers are sent for whole cText objects, but 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 be given relative to the beginning of the document, and the end must be relative to the end of the document ("spellcheck 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'").

Of course, the object specifier that the word processor sends must be one that it can resolve itself. How it chooses to specify the object is completely up to the word processor - it can specify windows by name, location, or whatever. The innards of the object specifier are not inspected by the speller.

Apple events defined in the Word Services suite

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

▪ **Table X-1** Apple events defined in the Word Services suite

Name	**Requested action**
BatchProcessMyText	Apply a word service to one or more blocks of text.
QueryReplace	Replace text upon user confirmation
CheckWord	Spell check a single word
InteractivelyProcessText	Start an interactive spelling session
CheckWordInteractively	Check a word during an interactive session

BatchProcessMyText – Apply a word service to one or more blocks of text.

This event specifies one or more blocks of text to apply a service to. It 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. After the event is sent, the word services server may bring itself to the front, and use Get Data events to actually get the text, and Set Data events to replace the text in the document window. The server may also attempt to use Set Data to highlight text in the client's window; if this fails, then the server will show the text in its own window.

Event Class	kWordServicesClass
Event ID	kWSBatchCheckMe

Parameters

keyDirectObject

Description:	This is the object specifier for the text to be checked. 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 -that is, the server does not resolve the object, instead it sends the object specifier back to the client.
Descriptor Type:	typeObjectSpecifier or typeAEList.
Required or Optional?	Required

keyListHead

Description:	This flags the use of a list head specifier in the direct object. If the keyListHead parameter has a value of true, then the keyAEDirectObject parameter is an object specifier 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 spellchecked. The spellchecker may 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).

Succeeding elements are specified using formRelativePosition (give me the next).

This allows the word processor to request the processing of a large number of blocks of text without having to worry about overflowing the 64K packet size limit. There is the cost that the word processor will need to maintain the list. A word processor need not literally keep a list of object specifiers; it needs only to know which items are selected for spellchecking, so it can create the object specifiers as they are requested.

Descriptor Type:	typeBoolean
Required or Optional?	Optional (default value: false)

keyFacelessMode

Description:	This specifies that faceless mode is to be used. If the keyFacelessMode parameter is true, then spellchecking is done in faceless mode. The word processor should use a unique transaction ID for the batch event; this transaction ID will be used by the subequent "Query Replace" events. (Do not set the kAENeverInteract flag in the batch event. The speller may need to do some interaction, such as to prompt the user to locate a dictionery file).
Descriptor Type:	typeBoolean
Required or Optional?	Optional (default value: false)

keyWantCleanReport

Description: This specifies that the word processor wants to know what text has been checked. If the keyWantCleanReport parameter is true, then the speller will send Set Data events to set the "pCleanText" of ranges of text.

This allows a word processor that supports the property to mark the text as "clean". When the user types into that range of text, it is marked as "dirty". When spellchecking is requested by the user, the word processor may request spellchecking only for the dirty blocks by sending only their objects specifiers, rather than the whole document. This can speed up spellchecking.

The concept of "clean" and "dirty" text might be completely meaningless for some sorts of services. A service that simply counted words would not get accurate results if it did not get all the text each time. There ought to be a way to allow the user to choose whether this optimization gets used at all.

Descriptor Type:	typeBoolean
Required or Optional?	Optional (default value: false)

keyClientAddress

Description:	This identifies the client word processor. If the keyClientAddress parameter is present, then 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 tell the speller to check some other word processor's document.
Descriptor Type:	typeSessionID
Required or Optional?	Optional

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	description

QueryReplace – Replace text upon user confirmation

The speller sends this event to inform the word processor that a range of text may need to be replaced.

Event Class	kWordServicesClass
Event ID	kWSQueryReplace

Parameters

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 may show.
Descriptor Type:	typeIntlText
Required or Optional?	Required

keyAEData

Description:	This is a list of possible replacements for the questioned text. There may 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)

keyErrorString

Description:	Description: This is a human readable string that explains what sort of error has occurred. 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

CheckWord – Spell check a single word

This event requests that a single word be spellchecked. An optional parameter gives the maximum number of possible replacements to the word that may be returned in a list in the reply.

If the application that receives this event is not a spelling checker, the action is not defined. There may be some other use to this than just checking spelling, like 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:	typeText
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 will be used.

Reply Parameters

keyAEResult

Description:	This contains a list of possible replacements to an incorrect word.
Default Descriptor Type:	typeAEList (a list of typeText objects)
Required or Optional?	Optional. It is only present if a non-zero amount of guesses were requested. If guesses were requested, but non were available, it may either be an empty list, or may be entirely absent. (Thus it has the default value of an empty list). Even if it is present, it may not contain as many guesses as 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.)
keyAEDirectObject		
	Description:	This contains a true value if the word was correct, or false if not.
	Descriptor Type:	typeBoolean
	Required or Optional?	Required

InteractivelyProcessText – Start an interactive spelling session

This apple event requests a speller to start up an interactive text checking session. No data is sent in this event; it is used to make sure that the speller is available and capable of interactive spellchecking. If a reply is received with no error code, then the speller may start sending words for checking with the CheckWordInteractively event.

Event Class kWordServicesClass

Event ID kWSStartInteractive

Parameters

keyFacelessMode

	Description:	This specifies that faceless mode is to be used. If the keyFacelessMode parameter is true, then spellchecking is done in faceless mode. The word processor should use a unique transaction ID for the event; this transaction ID will be used by the subequent "Query Replace" events. (Do not set the kAENeverInteract flag in the event. The speller may need to do some interaction, such as to prompt the user to locate a dictionery 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

CheckWordInteractively – Check a word while typing

This apple event sends words to a server that has already received the InteractivelyProcessText event and is prepared to receive words and check them as they come in.

Event Class	kWordServicesClass
Event ID	kWSCheckInteractive

Parameters

keyDirectObject

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

keyAEData

Description:	the actual text of the word that is sent
Descriptor Type:	typeIntlText
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 X-2 lists these object classes.

■ **Table X-2** Apple event object classes defined in the Word Services suite

Object class ID	Description	
cText	Description	
	Properties:	pLockTransactionID
	Element Classes:	cBackgroundHilite
cChar	Description	
	Properties:	pCleanText
	Element Classes:	none
cApplication	Description	
	Properties:	pBatchMenuString, pInteractiveMenuString, pMenuIcon, pLocation
	Element Classes:	none

Figure X-1 illustrates the inheritance hierarchy for the object classes defined in the Word Services suite. Listed for each object class are the properties, element classes, and Apple events that have not been inherited from object classes higher in the inheritance hierarchy.

■ **Figure X-1** Object inheritance hierarchy for the Word Services suite

[Hierarchy illustration here]

cText—series of characters

This is an extension to the existing cText object class. We add an element that specifies text that is shown highlighted while the window containing the text is in the background.

Superclass cAbstractObject

Default Descriptor Type

typeIntlText

Properties

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, then the property has the value "kAnyTransactionID".

When a word processor has resolved an object specifier that refers to a cText object, or a property or element of a cText object, it should check to see whether a lock owner exists for the cText object. If it does, then the word processor should extract the transaction ID from the Apple Event. If the transaction ID does not match, then the word processor should return errAEInTransaction rather than returning the result.

An implication of this property is that if its value is not kAnyTransactionID, it may not be set except by an event whose transaction ID matches its value.

A word processor might implement this property by maintaining a list of locks, with the contents of the list being the transaction ID's and pointers to the cText objects. Since there should be at most a few (usually zero or one) locks in existence, this way might be easier than adding a transaction ID field to the data structure that implements the cText objects.

A speller can still operate on a word processor that does not support this property. It can try to Get Data from the property. If an error is returned to the read, then the property is not implemented, and the speller will go on as if it had gotten the lock. The lock property exists to make the Word Services Suite more reliable, but is not strictly necessary, if the user takes care not to allow more than one other program to access her document at a time.

There is a problem in determining the value to use for the transaction ID. We cannot use Begin Transaction, as it will lock out all other transactions. The speller can simply make one up, but there is a danger that it is not unique. I would rather not add a new Apple Event that the word processor must support to supply a unique ID.

A reasonable convention would be 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 getting to send Set Data events during a single tick is fairly small.

Object Class ID:	typeLongInteger
Inherited?	No
Modifiable or Non-modifiable?	Modifiable

Element Classes

cBackgroundHilite

Description: These elements are the sets of cChar objects contained in the cText container that are shown highlighted while the containing window is in the background. Ordinarily a user's selection is not highlighted when a window is not in front, but the text specified by the value of the property will be.

This element exists to allow a Word Services server to highlight a suspect word in the original document, rather than showing it's own window. This way the suspect word can be displayed with all of its font, style, and other attributes, in the context of the original document.

Ordinarily this element will be implemented by setting a text selection, with a special case that leaves the highlighting of the selection on when the window is backgrounded. If you desire, you do not have to change the user's selection to show the highlighting (the highlighting is for display purposes only), but instead you could preserve the user selection, and implement this a different way. Also, this property is entirely optional. If it is not supported in a client, the server will display the sample text in its own window.

Support for this element is entirely optional. If a word processor does not support it, it should return an error code when the speller tries to set it. The speller must then show the text in its own window. Support for more than one element is also optional. If the word processor cannot show disjoint selections, it should return an error when the speller tries to set any elements other than the first.

Deleting all of the elements turns off the background hilighting.

Note that one sets this element by using a Set Data event, where the data to be set is a typeObjectSpecifier that uses a formRange to specify a range of characters within the cText object to highlight. The object specifier for the element itself will use a formAbsolutePosition to specify which cBackgroundHilite element to set.

Inherited?	No
Modifiable or Non-modifiable?	Modifiable
Key Forms:	formAbsolutePosition

Apple Events

Apple events from the Core suite:

Get Data	Inherited from cObject

cChar—text characters

This is an extension to the existing cChar object class. We add a property
that allows a range of text to be marked as "clean" so that a word processor may choose
not to have it spellchecked again in the future.

Superclass cText (Core suite)

Default Descriptor Type

typeIntlText

Properties

pCleanText

Description:	the pCleanText property of a character is "true" if it has been spellchecked since it was last edited. It does not make sense to have a clean character (though it is an ideal to strive for), but ranges of characters may be marked clean. This is a property possessed by the word processor that may be set by the speller, but only if the word processor requested it in its "Batch Process My Text" event. When the user requests spellchecking, the word processor can send object specifiers for the dirty text blocks. When a user edits a clean text block, the word processor can mark the block as dirty.
Object Class ID:	typeBoolean
Inherited?	No.
Modifiable or Non-modifiable?	Modifiable

Apple Events

Apple events from the Core suite:

Set Data	Inherited from cObject

cApplication—a standard Macintosh application

This is an extension to the existing cApplication object class. We add two properties that contain text a word processor can display in its menu to list the service provided by a speller, as well as a property that gives an alias record to the application so it may be launched again.

Superclass cAbstractObject

Default Descriptor Type

typeIntlText

Properties

pBatchMenuString

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

pInteractiveMenuString

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

pMenuIcon

Description:	The value of this property is a small icon that may be placed in the menu along with the interactive or batch menu strings. It should be identical to the small icon that the speller shows in the Finder. Word processors that take advantage of this can use

it to make the different menu items more distinguishable (a user might have two different spelling checkers).

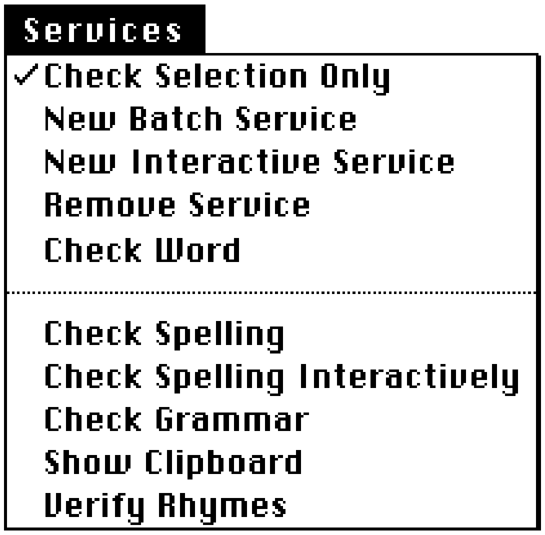
Object Class ID:	typePixelMap
Inherited?	No
Modifiable or Non-modifiable?	Non-modifiable

Notes

When the word processor has located a new speller, it should ask for its pBatchMenuString and pInteractiveMenuString properties. The strings returned should be added to the word processor's menu, and should be saved for next time as well. This way the user can have separate items for each service they may want. It is quite possible that they may have a number of different "spellers" on their machine, each with a menu item for interactive and batch service (a particular speller might not support both modes). The word processor should also have a menu item for "New Service" to locate a speller it does not already know about, and a "Delete Service" item to remove a service that is no longer needed.

If the word processor wants a richer interface, it can also request the propMenuIcon property to get a typePixelMap object that it may use as a small icon for the menu items.

Here is a suggested layout for such a menu. New services are just appended to the end. This is just a suggestion! There are many ways this could be done - services could be selected from a scrolling list, or minimal support might be provided by going to the PPCBrowser every time checking is requested.



pLocation

Description:	This property is an alias record that one may save, and use to launch the application again at a later time.
Object Class ID:	typeAlias
Inherited?	No.
Modifiable or Non-modifiable?	Non-modifiable

Notes Word processors should get this property from spellers when they first locate them, and save the alias record as a resource in their preferences file. When the user chooses the menu item associated with this alias, the word processor should use IPCListPorts to see if the speller is already running. If it is not, then the word processor should send the Open Selection event to the Finder to launch the speller.

Primitive object classes defined in the Word Services suite

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

▪ **Table X-3** Primitive object classes defined in the Word Services suite

Object class ID	Descriptor type of element	Description
cBackgroundHilite	typeObjectSpecifier	cBackgroundHilite object is an object specifier for a range of text that is shown hilighted, even when the containing window is in the background.

Descriptor types defined in the Word Services suite

There are no new descriptor types in this suite. *(I leave the table in, in case we want to add types in a later draft).*

▪ **Table X-4** Descriptor types defined in the Word Services suite

Descriptor type	Description
typeThis	description
typeAEThat	description

Key forms defined in the Word Services suite

There are no new key forms in this suite. *(I leave the table in, in case we want to add key forms in a later draft).*

▪ **Table X-5** Key forms defined in the Word Services suite

Key form constant	Description
formRange	Specifies a list of elements between two other elements (for example, “the words *between ‘Wild’ and ‘Zanzibar,’ inclusive*”)

Comparison operators defined in the Word Services suite

Table X-6 lists the comparison operators defined in the Word Services suite. There are no new comparison operators in this suite. *(I leave the table in, in case we want to add comparison operatorsin a later draft).*

▪ **Table X-6** Comparison operators defined in the Word Services suite

Comparison operator constant	Operator	Meaning
kAEBeginsWith		`'bgwt'` The value of the first operand begins with the value of the second operand (for example, the string “operand” begins with the string “opera”).

Constants defined in the Word Services suite

Table X-7 lists the constants defined in the Word Services suite.

▪ **Table X-7** Constants defined in the Word Services suite

Constant	Value
kWordServicesClass	'WSrv'
kWSBatchCheckMe	'Btch'
kWSQueryReplace	'QRep'
kWSCheckInteractive	'CkIn'
kWSCheckWord	'CkWd'
pBackgroundHilite	'pBgH'
pBatchMenuString	'pBMs'
pInteractiveMenuString	'pIMs'
pLocation	'pALc'
pLockTransactionID	'pLID'
pCleanText	'pCTx'
keyListHead	'Lhed'
keyClientAddress	'Cadr'
keyWSGuessCount	'Gcnt'
keyWantCleanReport	'ClRp'

Change History

Changes made since 0.7

1. Cleaned up typos galore.

2. Changed keyAEDirectObject to keyAEResult in reply for CheckWord event.

Changes made since 1.0a1

1. The spec has been formatted with the registry template.

2. Property names such as "propLocation" have been changed to "pLocation".

Changes made since 1.0d5
1. I changed the background hilighting from a property of the application to an element of a cText object. There may be zero or more hilit ranges. An application may choose to support multiple elements (so that grammar checkers may do disjoint hilighting), one element (for the normal sort of text selection), or none at all, if it is not feasible to show a selection when in the background. The highlighting is removed by deleting all of the elements.

2. I added a new, optional paramater to the Batch event. The keyword is "keyListHead". It the parameter is present and has the boolean value of "true", then the direct object is a table of object specifiers that are maintained by the word processor. The elements in this table are specifiers for the cText objects that are to be spellchecked.

3. I corrected the note about specifying a range within a text field to be checked. The beginning of the range must be relative to the beginning of the text field, and the end must be relative to the end, rather than both relative to the end.

4. I added an optional parameter to the Batch event, keyClientAddress, which allows a different program to be specified as the target for spellchecking. This would allow a scripting program to tell a speller to spellcheck a word processing document.

5. I added the propMenuIcon to the speller cApplication object. The word processor

may request the data for this property to display alongside the menu strings.

6. I added the propLockTransactionID property to the cText object. This allows a speller to request exclusive access to a text field.

7. The constants were changed to match the final Apple Event Registry. (keyAETheData changed to keyAEData, etc.)

8. Added the "Query Replace" event that is used by the speller to request text replacement in the word processor during faceless checking.

9. Added the definition of the "Interactively Check Word" event.

10. Added the "propCleanText" property to the cChar object class to allow ranges of characters to be marked as clean, so that a word processor can choose not to re-spellcheck them. This is only done if requested in the keyWantCleanReport parameter of the "Batch Process My Text" event.