Spell Checking

February 9, 2004

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

Simultaneously published in the United States and Canada.

Contents

Introduction to Spell Checking

Spell Checking describes Cocoa's spell-checking facilities.

Who Should Read This Document

You should read this document if you need to implement spell checking in your application or if you wish to make your spelling checker available as a service to other applications.

Organization of This Document

This document contains the following articles:

- "Spell Checker" (page 7) describes the basic features of the spell check facility.
- "Checking Text Spelling" (page 9) explains how to use the spell check facilities from an application.
- "Creating a Spell Server" (page 11) explains how to make your particular spelling checker a service that's available to any application

Spell Checker

The NSSpellChecker class gives any application an interface to the Cocoa spell-checking service. To handle all its spell checking, an application needs only one instance of NSSpellChecker. It provides a panel in which the user can specify decisions about words that are suspect.

The spell checker facility also provides lists of potential word completions culled from its spelling dictionary for the text-completion system in Mac OS X version 10.3 and later.

Checking Text Spelling

To check the spelling of some text, an application performs the following actions:

- Includes in its user interface a menu item (or a button or command) by which the user will request spell checking.
- Makes the text to be checked available as an NSString object.
- Creates an instance of the NSSpellChecker class and sends it a `checkSpellingOfString:startingAt:` message.

For example, you might use the following statement to create a spell checker:

```
range = [[NSSpellChecker sharedSpellChecker] checkSpellingOfString:aString
 startingAt:0];
```

The `checkSpellingOfString:startingAt:` method checks the spelling of the words in the specified string beginning at the specified offset (this example uses 0 to start at the beginning of the string) until it finds a word that is misspelled. Then it returns an NSRange to indicate the location of the misspelled word.

In a graphical application, whenever a misspelled word is found, you'll probably want to highlight the word in the document, using the NSRange that `checkSpellingOfString:startingAt:` returns to determine the text to highlight. Then you should show the misspelled word in the Spelling panel's misspelled-word field by calling `updateSpellingPanelWithMisspelledWord:`. If `checkSpellingOfString:startingAt:` does not find a misspelled word, you should call `updateSpellingPanelWithMisspelledWord:` with the empty string. This causes the system to beep, letting the user know that the spell check is complete and no misspelled words were found. None of these steps is required, but if you do one, you should do them all.

The object that provides the string being checked should adopt the following protocols:

Protocol	Description
NSChangeSpelling	A message in this protocol (`changeSpelling:`) is sent down the responder chain when the user presses the Correct button.
NSIgnoreMisspelledWords	When the object being checked responds to this protocol, the spell server keeps a list of words that are acceptable in the document and enables the Ignore button in the Spelling panel.

The application may choose to split a document's text into segments and check them separately. This is necessary when the text has segments in different languages. Spell checking is invoked for one language at a time, so a document that contains portions in three languages requires at least three checks.

Dictionaries and Word Lists

The process of checking spelling makes use of three references:

- A dictionary registered with the system's spell-checking service. When the Spelling panel first appears, by default it shows the dictionary for the user's preferred language. The user may select a different dictionary from the list in the Spelling panel.
- The user's "learn" list of correctly-spelled words in the current language. The NSSpellChecker updates the list when the user presses the Learn or Forget buttons in the Spelling panel.
- The document's list of words to be ignored while checking it (if the first responder conforms to the NSIgnoreMisspelledWords protocol). The NSSpellChecker updates its copy of this list when the user presses the Ignore button in the Spelling panel.

A word is considered to be misspelled if none of these three accepts it.

Matching a List of Ignored Words with the Document It Belongs To

The NSString being checked isn't the same as the document. In the course of processing a document, an application might run several checks based on different parts or different versions of the text, but they all belong to the same document. The NSSpellChecker keeps a separate "ignored words" list for each document that it checks. To help match "ignored words" lists to documents, you should call `uniqueSpellDocumentTag` once for each document. This method returns a unique arbitrary integer that will serve to distinguish one document from the others being checked and to match each "ignored words" list to a document. When searching for misspelled words, pass the tag as the fourth argument of `checkSpellingOfString:startingAt:language:wrap:inSpellDocumentWithTag:wordCount:`. (The convenience method `checkSpellingOfString:startingAt:` takes no tag. This method is suitable when the first responder does not conform to the NSIgnoreMisspelledWords protocol.)

When the application saves a document, it may choose to retrieve the "ignored words" list and save it along with the document. To get back the right list, it must send the NSSpellChecker an `ignoredWordsInSpellDocumentWithTag:` message. When the application has closed a document, it should notify the NSSpellChecker that the document's "ignored words" list can now be discarded, by sending it a `closeSpellDocumentWithTag:` message. When the application reopens the document, it should restore the "ignored words" list with the message `setIgnoredWords:inSpellDocumentWithTag:`.

Creating a Spell Server

The NSSpellServer class gives you a way to make your particular spelling checker a service that's available to any application. A service is an application that declares its availability in a standard way, so that any other applications that wish to use it can do so. If you build a spelling checker that makes use of the NSSpellServer class and list it as an available service, then users of any application that makes use of NSSpellChecker or includes a Services menu will see your spelling checker as one of the available dictionaries.

To make use of NSSpellServer, you write a small program that creates an NSSpellServer instance and a delegate of the server that responds to messages asking it to find a misspelled word and to suggest guesses to correct the misspelled word. Send the NSSpellServer `registerLanguage:byVendor:` messages to tell it the languages your delegate can handle.

The program that runs your spelling checker should not be built as an Application Kit application, but as a simple program. Suppose you supply spelling checkers under the vendor name "Acme" and the file containing the code for your delegate is called AcmeEnglishSpellChecker. Then the following might be your program's main function:

```
void main()
{
    NSSpellServer *aServer = [[NSSpellServer alloc] init];
    if ([aServer registerLanguage:"English" byVendor:"Acme"]) {
        [aServer setDelegate:[[AcmeEnglishSpellChecker alloc] init]];
        [aServer run];
        fprintf(stderr, "Unexpected death of Acme SpellChecker!\n");
    else {
        fprintf(stderr, "Unable to check in Acme SpellChecker.\n");
    }
}
```

Your delegate is an instance of a custom subclass. (It's simplest to make it a subclass of NSObject, but that's not a requirement.) Given an NSString, your delegate must be able to find a misspelled word by implementing the method `spellServer:findMisspelledWordInString:language:wordCount:countOnly:`. Usually, this method also reports the number of words it has scanned, but that isn't mandatory.

Optionally, the delegate may also suggest corrections for misspelled words. It does so by implementing the method `spellServerS:suggestGuessesForWord:inLanguage:`.

Service Availability Notice

When there's more than one spelling checker available, the user selects the one desired. The application that requests a spelling check uses an NSSpellChecker object, and it provides a Spelling panel; in the panel there's a pop-up list of available spelling checkers. Your spelling checker appears in that list if it has a service descriptor.

A service descriptor is an entry in a text file called services. Usually it's located within the bundle that also contains your spelling checker's executable file. The bundle (or directory) that contains the services file must have a name ending in ".`service`" or ".`app`". The system looks for service bundles in a standard set of directories.

A spell checker service availability notice has a standard format, illustrated in the following example for the Acme spelling checker:

```
Spell Checker: Acme
Language: French
Language: English
Executable: franglais.daemon
```

The first line identifies the type of service; for a spelling checker, it must say "Spell Checker:" followed by your vendor name. The next line contains the English name of a language your spelling checker is prepared to check. (The language must be one your system recognizes.) If your program can check more than one language, use an additional line for each additional language. The last line of a descriptor gives the name of the service's executable file. (It requires a complete path if it's in a different directory.)

If there's a service descriptor for your Acme spelling checker and also a service descriptor for the English checker provided by a vendor named Consolidated, a user looking at the Spelling panel's pop-up list would see:

```
English (Acme)
English (Consolidated)
French (Acme)
```

Illustrative Sequence of Messages to an NSSpellServer

The act of checking spelling usually involves the interplay of objects in two classes: the user application's NSSpellChecker (which responds to interactions with the user) and your spelling checker's NSSpellServer (which provides the application interface for your spelling checker). You can see the interaction between the two in the following list of steps involved in finding a misspelled word.

- The user of an application selects a menu item to request a spelling check. The application sends a message to its NSSpellChecker object. The NSSpellChecker in turn sends a corresponding message to the appropriate NSSpellServer.
- The NSSpellServer receives the message asking it to check the spelling of an NSString. It forwards the message to its delegate.

- The delegate searches for a misspelled word. If it finds one, it returns an NSRange identifying the word's location in the string.
- The NSSpellServer receives a message asking it to suggest guesses for the correct spelling of a misspelled word, and forwards the message to its delegate.
- The delegate returns a list of possible corrections, which the NSSpellServer in turn returns to the NSSpellChecker that initiated the request.
- The NSSpellServer doesn't know what the user does with the errors its delegate has found or with the guesses its delegate has proposed. (Perhaps the user corrects the document, perhaps by selecting a correction from the NSSpellChecker's display of guesses; but that's not the NSSpellServer's responsibility.) However, if the user presses the Learn or Forget buttons (thereby causing the NSSpellChecker to revise the user's word list), the NSSpellServer receives a notification of the word thus learned or forgotten. It's up to you whether your spell checker acts on this information. If the user presses the Ignore button, the delegate is not notified (but the next time that word occurs in the text, the method `isWordInUserDictionaries:caseSensitive:` will report `YES` rather than `NO`).
- Once the NSSpellServer delegate has reported a misspelled word, it has completed its search. Of course, it's likely that the user's application will then send a new message, this time asking the NSSpellServer to check a string containing the part of the text it didn't get to previously.

Document Revision History

The table below describes the revisions to *Spell Checking*.

Date	Notes
February 9, 2004	Rewrote introduction, lightly edited all articles, and added an index.
November 12, 2002	Revision history was added to existing topic. It will be used to record changes to the content of the topic.

Index

U