NSSpellServer

Inherits From: NSObject

Conforms To: NSObject (NSObject)

Declared In: AppKit/NSSpellServer.h

Class Description

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 that responds to messages asking it to find a misspelled word and to suggest guesses for a 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." Suppose the file containing the code for your delegate is called AcmeEnglishSpellChecker. Then the following might be your program's **main**:

```
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 **spellServer: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 earlier.

Method Types

Registering your service	– registerLanguage:byVendor:
Assigning a delegate	– setDelegate: – delegate
Running the service	– run
Checking user dictionaries	– isWordInUserDictionaries:caseSensitive:

Instance Methods

delegate

– (id)**delegate**

Returns the NSSpellServer’s delegate.

See also: – **setDelegate:**

isWordInUserDictionaries:caseSensitive:

– (BOOL)**isWordInUserDictionaries:**(NSString *)*word* **caseSensitive:**(BOOL)*flag*

Indicates whether *word* is in the user's list of learned words or the document's list of words to ignore. If YES, the word is acceptable to the user. *flag* indicates whether the comparison is to be case-sensitive.

registerLanguage:byVendor:

– (BOOL)**registerLanguage:**(NSString *)*language* **byVendor:**(NSString *)*vendor*

Notifies the NSSpellServer of a language your spelling checker can check. *language* is the English name of a language on NeXT's list of languages. *vendor* identifies the vendor (to distinguish your spelling checker from those that others may offer for the same language). If your spelling checker supports more than one language, it should invoke this method once for each language. Registering a language/vendor combination causes it to appear in the Spelling Panel's pop-up list of spelling checkers.

Returns YES if the language is registered, NO if for some reason it can't be registered.

run

– (void)**run**

Causes the NSSpellServer to start listening for spell-checking requests. This method starts a loop that never returns; you need to set the NSSpellServer's delegate before sending this message.

See also: – **setDelegate:**

setDelegate:

– (void)**setDelegate:**(id)*anObject*

Assigns a delegate to the NSSpellServer. Since the delegate is where the real work is done, this is an essential step before telling the NSSpellServer to run.

See also: – **delegate**, – **run**

Methods Implemented by the Delegate

spellServer:didForgetWord:inLanguage:

– (void)**spellServer:**(NSSpellServer *)*sender*
didForgetWord:(NSString *)*word*
inLanguage:(NSString *)*language*

Notifies the delegate that *word* has been removed from the user's list of acceptable words. If your delegate maintains a similar auxiliary word list, you may wish to edit the list accordingly.

spellServer:didLearnWord:inLanguage:

– (void)**spellServer:**(NSSpellServer *)*sender*
didLearnWord:(NSString *)*word*
inLanguage:(NSString *)*language*

Notifies the delegate that *word* has been added to the user's list of acceptable words. If your delegate maintains a similar auxiliary word list, you may wish to edit the list accordingly.

spellServer:findMisspelledWordInString:language:wordCount:countOnly:

– (NSRange)**spellServer:**(NSSpellServer *)*sender*
findMisspelledWordInString:(NSString *)*stringToCheck*
language:(NSString *)*language*
wordCount:(int *)*wordCount*
countOnly:(BOOL)*countOnly*

Asks the delegate to search for a misspelled word in *stringToCheck*, using *language*, and marking the first misspelled word found by returning its range within the string object. In *wordCount* return by reference the number of words from the beginning of the string object until the misspelled word (or the end-of-string). If *countOnly* is YES, just count the words in the string object; do not spell-check. Send **isWordInUserDictionaries:caseSensitive:** to the spelling server to determine if word exists in the user's language dictionaries.

spellServer:suggestGuessesForWord:inLanguage:

– (NSArray *)**spellServer:**(NSSpellServer *)*sender*
suggestGuessesForWord:(NSString *)*word*
inLanguage:(NSString *)*language*

Gives the delegate the opportunity to suggest guesses for the correct spelling of the misspelled *word*. Return the guesses as an array of NSStrings.