11

Regular Expressions in Microsoft Word

Regular expressions can be very useful when carrying out searches in Microsoft Word or when carrying out search and replace operations. However, the regular expression syntax used in Microsoft Word differs significantly from that in most programming languages. Microsoft Word refers to regular expressions as wildcards and, unfortunately, has many nonstandard features. So if you have been using regular expressions in other languages or tools, you may have significant adjustments to make, in part because of nonstandard syntax in Word and in part because of the fairly limited functionality. There is, for example, no lookbehind or lookahead functionality.

In this chapter, you will learn the following:

- How to access regular expression functionality in Microsoft Word
- The regular expression functionality supported in Microsoft Word, and its differences from standard regular expression syntax
- How to use Microsoft Word wildcards in a range of example scenarios

The User Interface

The interface to use regular expressions in Microsoft Word is straightforward. To use regular expressions (wildcards) in Word 2003, simply select the Edit menu; then select Find. Alternatively, you can use the equivalent keyboard shortcut Ctrl+F. The Find and Replace dialog box opens. If you have not used regular expressions or other advanced search functionality in Word, the appearance expected is shown in Figure 11-1.

To access regular expression functionality, click the More button. Further options are displayed, as shown in Figure 11-2.

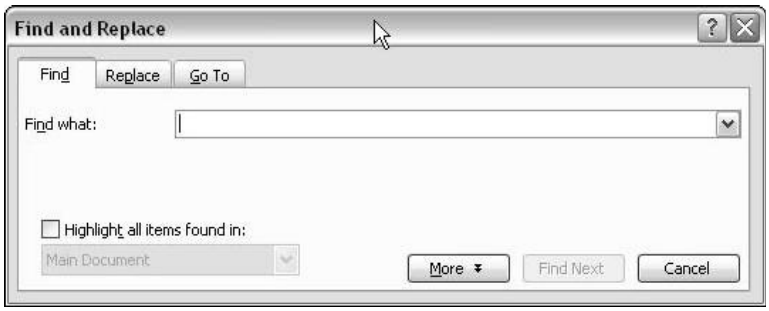

Figure 11-1

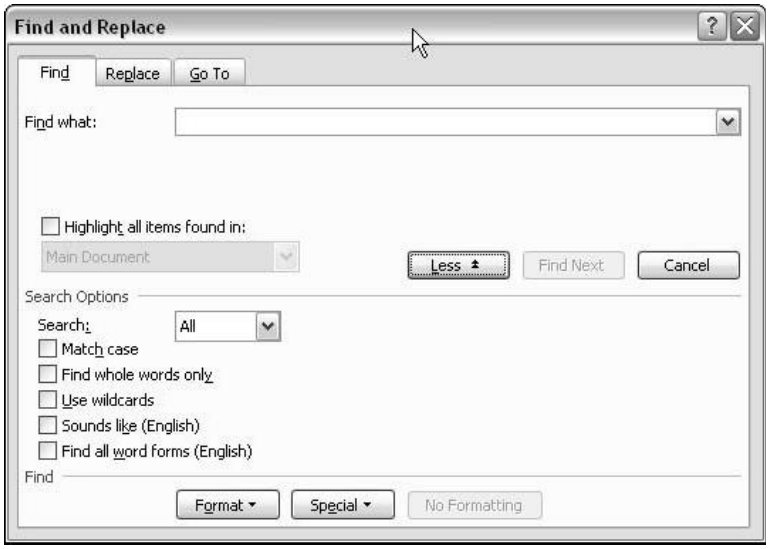

Figure 11-2

To use Word regular expression functionality, click the Use Wildcards check box.

Try It Out Word Wildcards

1. With Microsoft Word open, open the test file `Apple.doc`.

   The content of the test file is shown here:

```
The skin of the apple is green.

The apple's flavor was fantastic.

Apples grow on trees.
```

2. Type Ctrl+F to open the Find and Replace dialog box. If necessary, click the More button to display the Use Wildcards check box.

3. Check the Use Wildcards check box. When you click the Use Wildcards check box, the Match Case and Find Whole Words Only check boxes are grayed out. The Sounds Like and Find Whole Words Only check boxes cannot be checked at the same time as the Use Wildcards check box. Those three options, although displayed as check boxes in Word, function more like a set of radio buttons in a Web form, because you can select only one option at any one time.

4. Enter the pattern **apple** in the Find What text box, and click Find Next three times, inspecting the highlighted text after each click.

Figure 11-3 shows the appearance after clicking Find Next twice in Step 4. After the third click, a dialog box should display, indicating that the end of the file has been reached: "Word has finished searching the document."

How It Works

The matching of the character sequence `apple` in the first line is straightforward. The character sequence `Apple` in the third line does not match, because the default behavior in Microsoft Word, when wildcards are used, is to match case sensitively. The character sequence `Apple` fails to match the pattern `apple` because the first character of the character sequence is an uppercase `A`,and the first character of the pattern is a lowercase `A`.

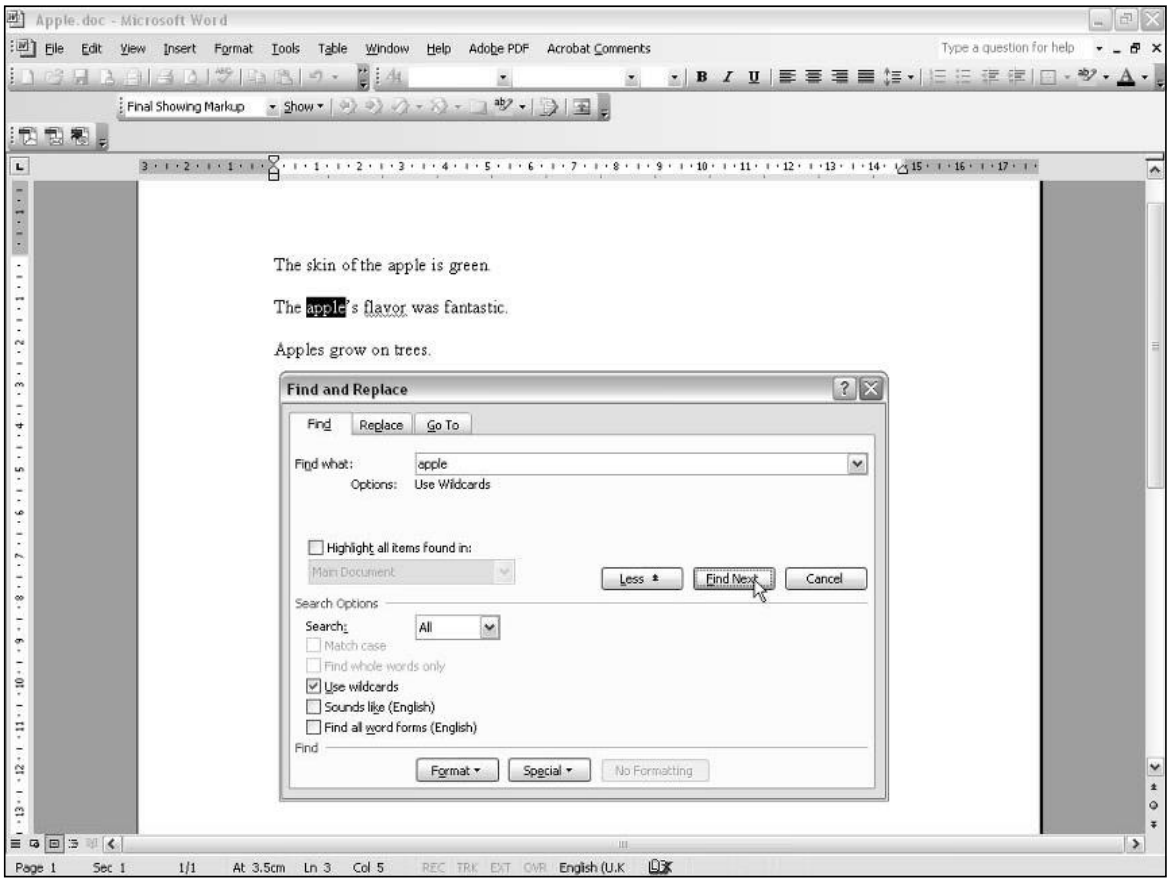

Figure 11-3

Metacharacters Available

The following table shows the metacharacters available in Word 2003. The functionality relating to what each metacharacter does in Word is described in more detail in the following sections of this chapter.

Metacharacter	Description
?	Nonstandard usage (in Word, it is not a quantifier). Matches a single character. Broadly equivalent to the dot metacharacter (`.`) in standard syntax.
*	Nonstandard usage. Matches zero or more characters. Broadly equivalent to `.*` in standard syntax.
<	Matches the position between a nonalphabetic character and a following alphabetic character. Effectively, a beginning-of-word position metacharacter.
>	Matches the position between an alphabetic character and a following nonalphabetic character. Effectively, an end-of-word position metacharacter.
[...]	Character class delimiters.
!	Used in character classes as the character class negation metacharacter.
{n,m}	Quantifier syntax.
@	Quantifier. Nonstandard syntax, to match one or more occurrences of the preceding character or group. Equivalent to the + metacharacter in standard syntax.

Quantifiers

Microsoft Word does not use the `?`, `*`, or `+` metacharacters as quantifiers in the way typical of most regular expression implementations. Instead, the `?` and `*` metacharacters are used in the same way as they are in DOS filenames.

The `?` metacharacter signifies a single character and is equivalent to patterns such as `.` or `\w` in most regular expression implementations.

So the pattern:

```
so?t
```

in Microsoft Word will match `soft` and `sort`. However, the `?` metacharacter in Word does not carry the notion of optionality. The pattern:

```
sa?t
```

will match `salt` but does not match `sat`, because there is no character between the `a` and `t` in `sat`. The `?` metacharacter in Word always matches exactly one character.

The * metacharacter in Word signifies zero or more characters and is roughly equivalent to .* or \w* in most implementations. Notice that while the ? metacharacter does not have the notion of optionality, the * metacharacter does signify an optional character. So the pattern:

```
se*t
```

will match set (the letter between the e and t is optional), seat, sect, and sent. However, the effect of the * metacharacter in Word may occasionally surprise you with unexpected matches when it matches zero characters. For example, the pattern:

```
s*t
```

will match words such as sit and sat, but will also match the character sequence st in words such as first and best. Figure 11-4 shows an example with the st character sequence in first matching. The test text is as follows:

```
sit
sat
first
```

The pattern is s*t.

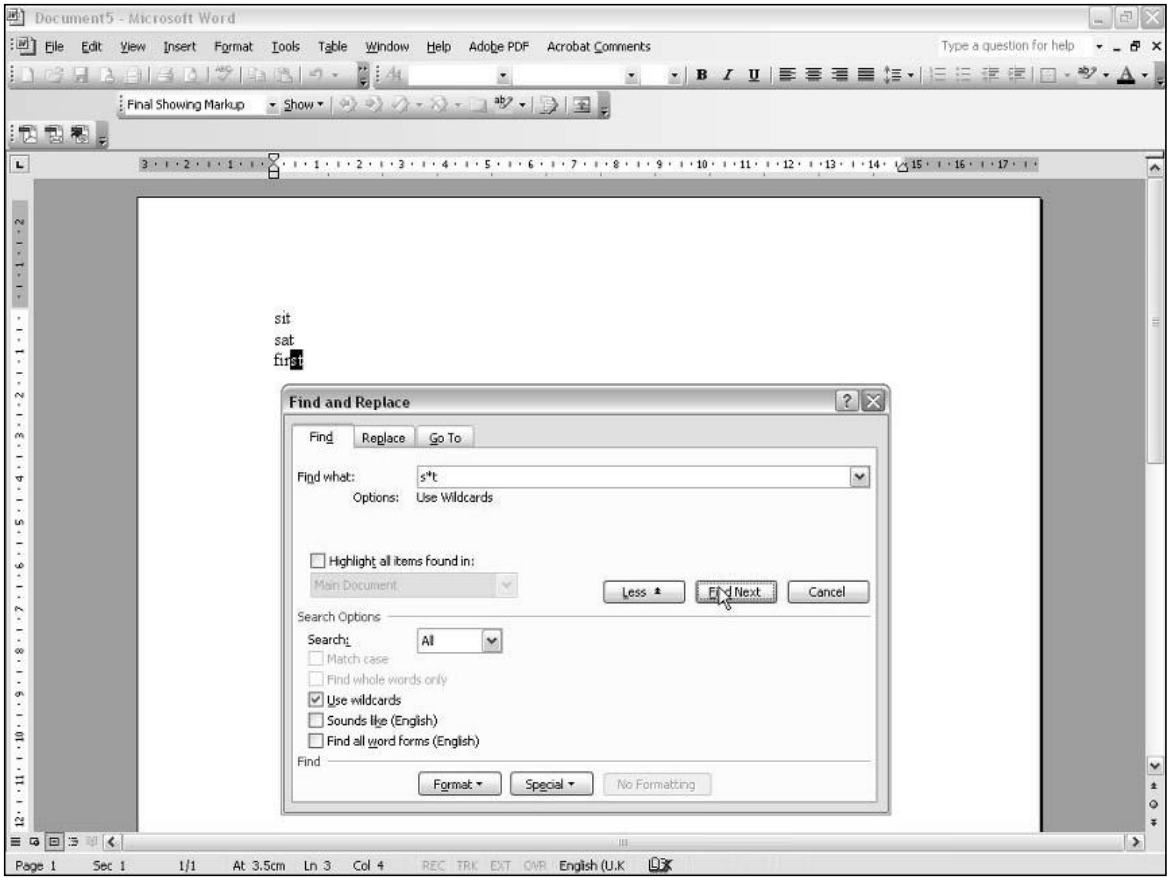

Figure 11-4

If you are carrying out a search and replace, such undesired matches can cause significant oddities in the text following the replacement operation.

> **The use of the `*` metacharacter in Word can lower the specificity of a search substantially. So replacement operations using the `*` metacharacter are, in general, to be avoided.**

The following table summarizes the support for quantifiers in Microsoft Word:

Metacharacter	Comment
`?`	The `?` metacharacter in Word is *not* a typical quantifier. It matches a single character and is not a quantifier at all.
`*`	The `*` metacharacter in Word is *not* a typical quantifier. It matches zero or more characters.
`@`	The `@` metacharacter.
`{n,m}`	The `{n,m}` syntax is a quantifier in Word.

There is no quantifier in Word equivalent to the zero or more occurrences `?` metacharacter in most regular expression implementations. Neither is there a quantifier that matches zero or more occurrences as the `*` metacharacter does in most regular expression implementations. In Word, the `{n,m}` syntax cannot be used to provide equivalent functionality.

The @ Quantifier

The only true quantifier metacharacter in Microsoft Word is the `@` metacharacter. It is equivalent to the `+` metacharacter in most implementations, matching one or more occurrences of the character or group that precedes it.

Try It Out — The @ Quantifier

The test file is `Hot.doc`, whose content is shown here:

```
hot
hoot
host
holt
shoot
ht
```

1. Open `Hot.doc` in Microsoft Word, and open the Find and Replace dialog box using the Ctrl+F keyboard shortcut.
2. Check the Use Wildcards check box, and type the pattern **ho@t** in the Find What text box.
3. Click Find Next three times. After each click, inspect the text that is highlighted.

   Figure 11-5 shows the appearance after Step 3, when the Find Next button has been clicked three times. After the first click, the character sequence `hot` is highlighted. After the second click, the character sequence `hoot` is highlighted. After the third click, the character sequence `hoot` in `shoot` is highlighted. Notice that the character sequence `ht` on the final line is not highlighted if you click the Find Next button again, because zero occurrences of `o` between the `h` and `t` of `ht` does not match the `o@` component of the pattern.
4. Create a new blank Word document, and type the test text **AABABABAAAAAAA** in it.
5. If necessary, open the Find and Replace dialog box, and check the Use Wildcards check box.
6. Type the pattern **A(AB)@** in the Find What text box; click the Find Next button; and inspect the highlighted text, as shown in Figure 11-6.

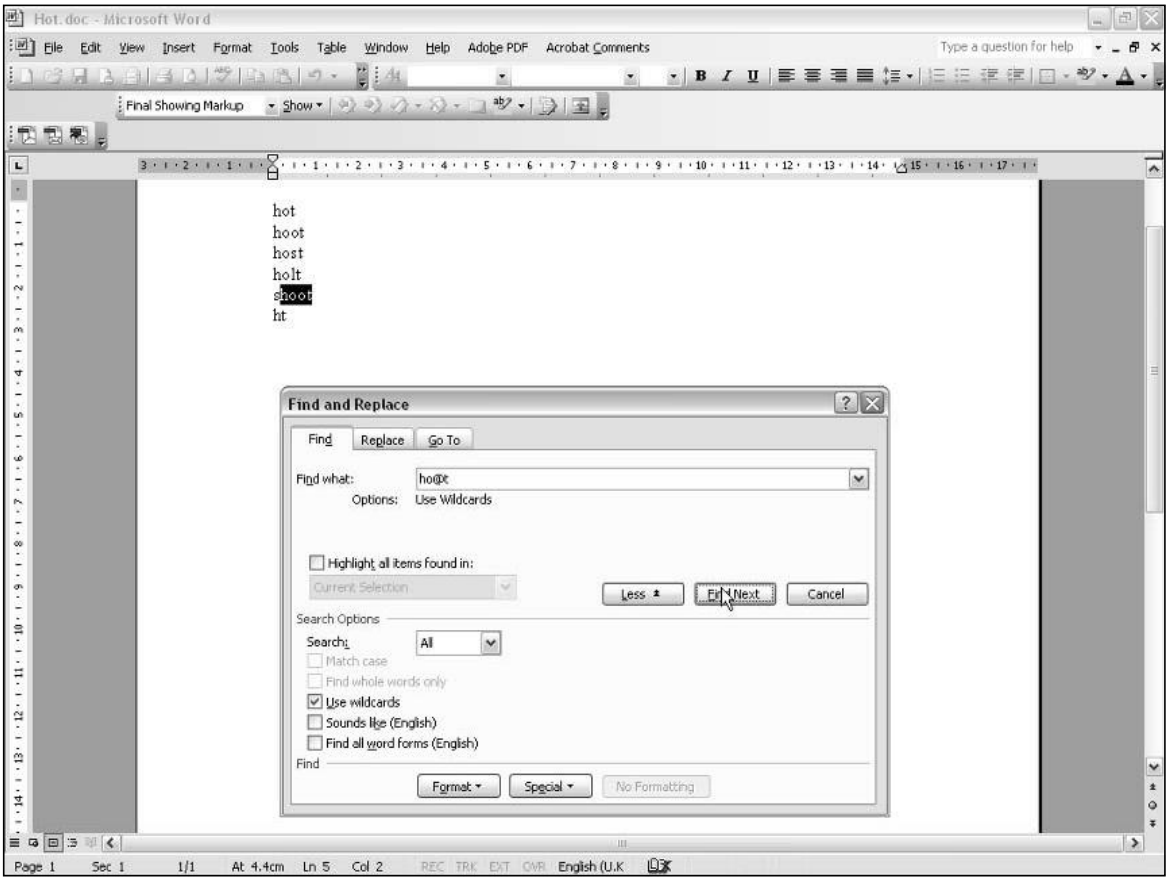

Figure 11-5

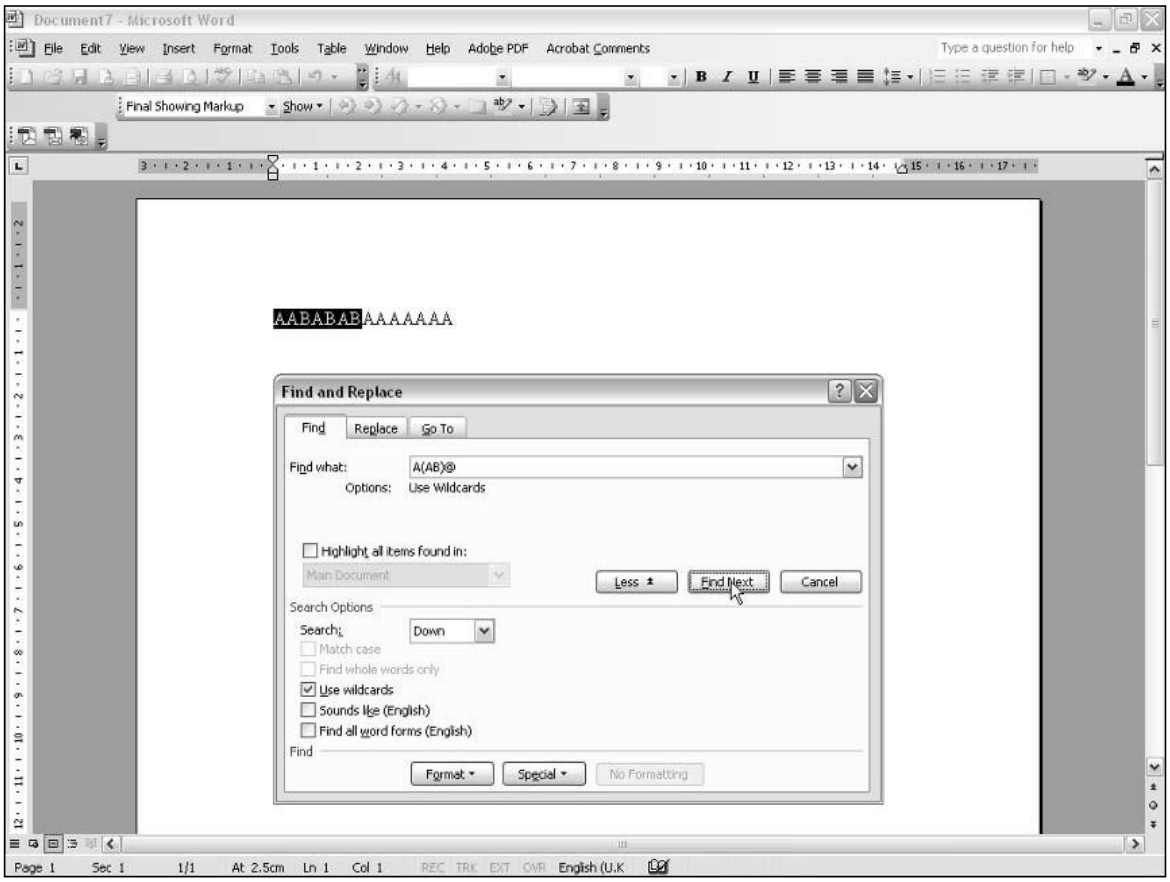

Figure 11-6

How It Works

The pattern `o@` matches one or more occurrences of the lowercase `o`. So in the test document `hot.doc`, the first click matches `hoot` because two occurrences of lowercase `o` match the pattern `o@`. Similarly, the second click matches `hot` because one occurrence of lowercase `o` in the test text matches the pattern `o@`. The third click matches `hoot` in `shoot` for the same reason as the first match.

When the pattern `A(AB)@` is used, the `@` quantifier refers to the preceding group, not simply a single character. In the example, the `(AB)@` component of the pattern matches three occurrences of the character sequence `AB`.

The {n,m} Syntax

The `{n,m}` quantifier syntax is available in Word but, unfortunately, cannot provide quantifiers equivalent to the `?` and `*` metacharacters in typical regular expression syntax.

Some limitations exist in how you can use the {n,m} syntax in Word. You cannot, for example, use `{0,}` or `{0,*}` to create a quantifier exactly equivalent to the `*` metacharacter in typical regular expression syntax. In fact, you can't use `0` as the first numeric digit inside curly braces at all. The lack of support for `0` as a minimum occurrence constraint means that neither the `?` nor `*` metacharacters can be mimicked using the `{n,m}` syntax.

Try It Out The {n,m} Syntax

The test document, `Zeros.doc`, is shown here:

```
AB1
AB10
AB100
AB1000
AB10000
```

1. Open `Zeros.doc` in Word, and open the Find and Replace dialog box using Ctrl+F.
2. Enter the pattern **AB10{1,100}** in the Find What text box, and click Find Next four times, inspecting the text highlighted after each click.

   Figure 11-7 shows the text's appearance after the first time that the Find Next button is clicked.
3. Edit the pattern in the Find what text box to `AB10{3,100}`, click the Find Next button three times, and inspect the text that is highlighted each time.

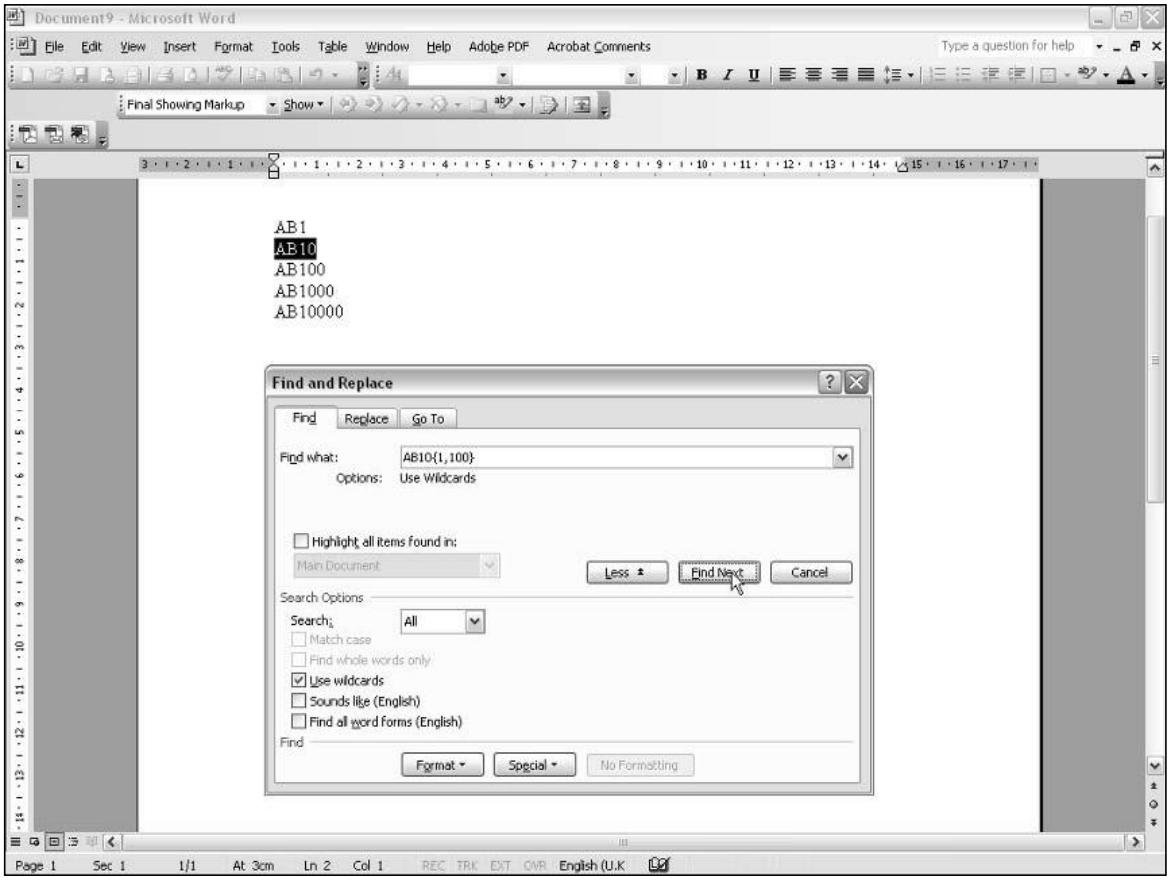

Figure 11-7

How It Works

The `{n,m}` syntax works as in other implementations. The `n` specifies the minimum number of occurrences that match. The `m` specifies the maximum number of occurrences that match.

When the pattern is `AB10{1,100}`, the first match is the character sequence `AB10`, because the first three characters of the pattern match literally, and there is one occurrence of the numeric digit 0, which fits with the minimum occurrence constraint.

The other matches for the pattern `AB10{1,100}` occur because the number of occurrences of the numeric digit `0` is between `1` and `100`, the minimum occurrence constraint and maximum occurrence constraint, respectively.

When the pattern is altered to `AB10{3,100}`, the first match is `AB1000`, which has three occurrences of the numeric digit `0`. The character sequences `AB1`, `AB10`, and `AB100` have too few occurrences of the numeric digit `0`.

Modes

The choice of modes in Microsoft Word is nil. There is no way to use wildcards in a case-insensitive way.

Applying the pattern:

```
Staff
```

to the test text, `Staff.doc`

```
Staff is an archaic word for a long walking stick.

The Staff party takes place on Friday.

I spoke yesterday with senior staff.
```

will find a match on the lines containing:

```
Staff is an archaic word for a long walking stick.
```

and

```
The Staff party takes place on Friday.
```

but will find no match on the line:

```
I spoke yesterday with senior staff.
```

because the pattern `Staff` matches only a sequence of characters that begins with an uppercase `S`.

There is a workaround to achieve case-insensitive matching: Use a character class for each literal character in the pattern to be matched. That approach can be tedious, but it allows things to be done that cannot be done in Word any other way. The test text, `Star.doc`, is shown here:

```
sTar

star

Star

STAR
```

Notice how the case of some characters differs among the sample lines.

Try It Out Character Classes to Match Case Insensitively

1. Open `Star.doc` in Microsoft Word, and use the Ctrl+F keyboard shortcut to open the Find and Replace dialog box.
2. Check the Use Wildcards check box, and enter the pattern **star** in the Find What text box.
3. Click the Find Next button, and inspect the results.
4. Click the Find Next button again, and inspect the results, as shown in Figure 11-8.

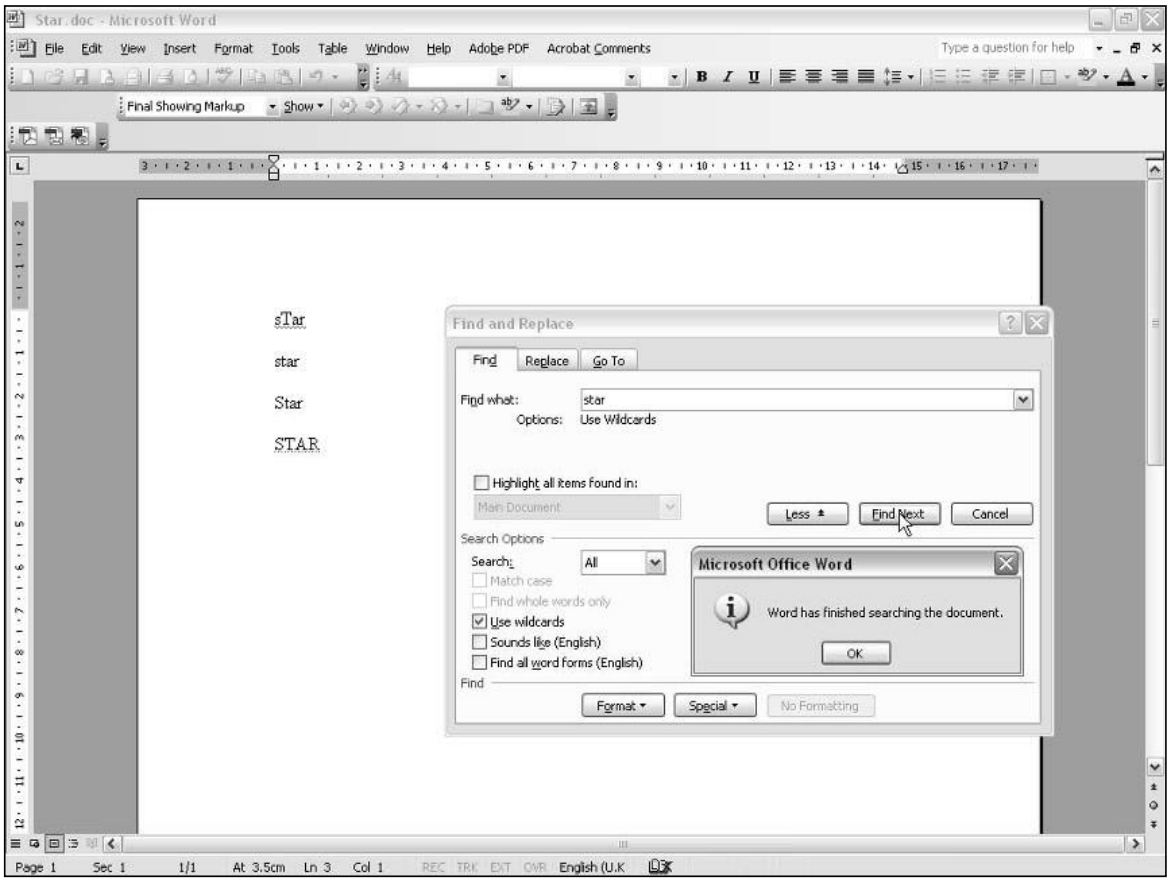

Figure 11-8

Notice that the line containing `star` (all lowercase) is the first (and only) line that matches. When you click Find Next again, the message window shown in Figure 11-8 is displayed.

5. Edit the pattern in the Find What text box to [Ss][Tt][Aa][Rr].

6. Ensure that the cursor is at the beginning of the test document. Click the Find Next button four times.

   Figure 11-9 shows the text's appearance after clicking Find Next once in Step 6.

How It Works

The pattern `star` matches case sensitively. The character sequence `sTar` does not match because its second character is an uppercase `T`, which does not match the corresponding character in the pattern, a lowercase `t`, when matching is case sensitive.

The character sequence `star` on the second line matches because each character in the pattern matches the corresponding character in the test character sequence.

The character sequence `Star` does not match because the first character is uppercase. The character sequence `STAR` does not match the pattern `star` because matching will fail on the first character, which is uppercase.

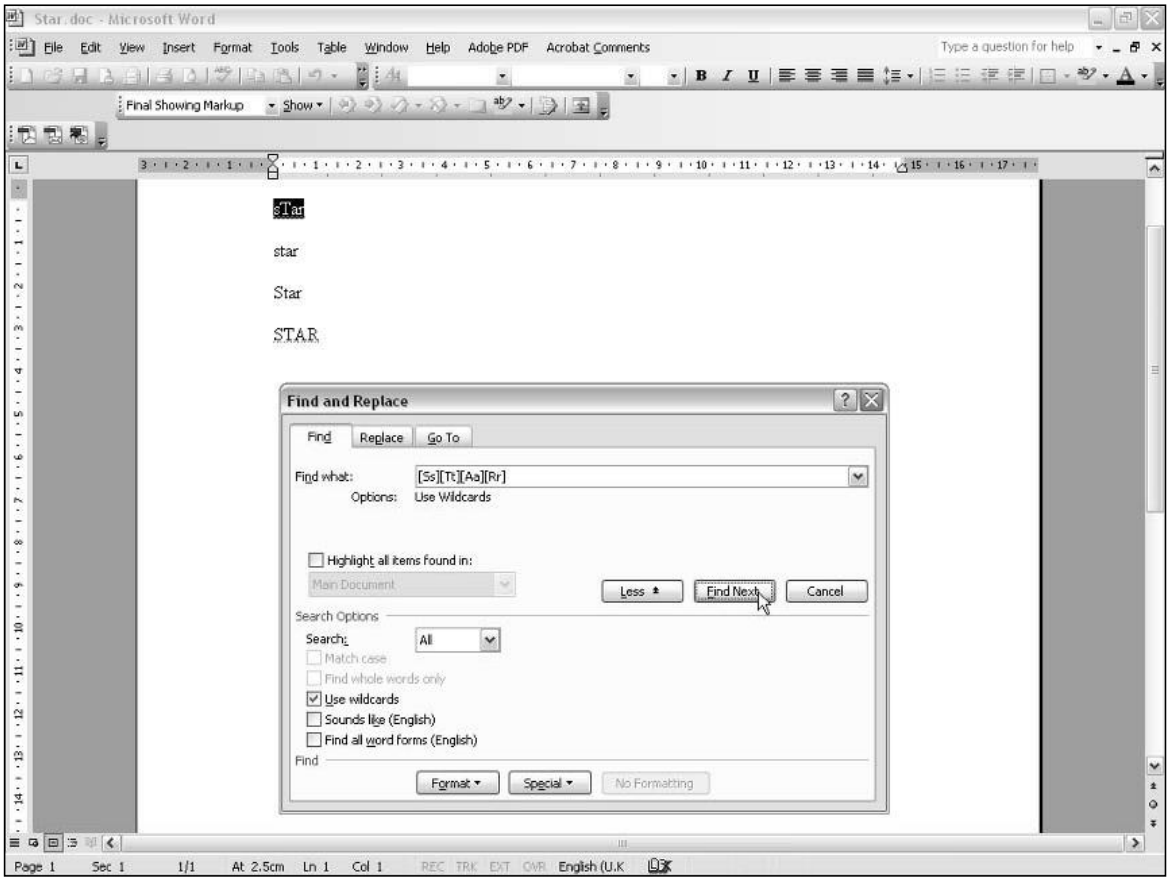

Figure 11-9

When the pattern is `[Ss][Tt][Aa][Rr]`, all of the test character sequences will match, because for each character in a test character sequence, both uppercase and lowercase options are provided.

Character Classes

Microsoft Word provides support for character classes. The most important variation from standard syntax is that the `!` metacharacter is used to negate a character class (in place of the `^` character typically used in other implementations).

Back References

Back references can be used successfully in Microsoft Word by grouping character sequences using paired parentheses in the normal way. An example using back references is provided later in this chapter.

Lookahead and Lookbehind

Microsoft Word does not support lookahead or lookbehind.

Lazy Matching versus Greedy Matching

Another difference between the default behavior of most regular expression implementations and frequent behavior of the regular expression implementation in Microsoft Word is that Word tends to default to lazy matching. Most other regular expression implementations default to greedy matching (with many languages providing a special syntax to be used for lazy matching).

However, Word conducts greedy matching in some circumstances. Recall the pattern `A(AB)@` in the example earlier in this chapter. It matched the maximum available occurrences.

Word usually matches as few characters as possible, consistent with matching a pattern. Let's look at an example to make the concept clearer. The test file, `ABC123.doc`, is shown here:

```
ABC123

ABC123456

ABC123456
```

Try It Out — Lazy Matching in Microsoft Word

1. Open the file `ABC123.doc` in Microsoft Word, and use the Ctrl+F keyboard shortcut to open the Find and Replace dialog box.
2. Check the Use Regular Expressions check box, and enter the pattern `*` in the Find What text box.

3. Click the Find Next button, and inspect the results. In Figure 11-10, notice that one character, the first uppercase `A`, is highlighted.
4. Click the Find Next button a few more times, each time inspecting the result.
5. Modify the pattern in the Find What text box to `A*`.
6. Click at the beginning of the file to ensure that the cursor is at the position before the first uppercase `A`. Click the Find Next button, and inspect the results.
7. Click the Find Next button twice more, inspecting the result each time.

   Notice in Figure 11-11 that each time, only the first uppercase `A` on the relevant line is highlighted.
8. Modify the pattern in the Find What text box to `A*{2,5}`.
9. Click at the beginning of the file to ensure that the cursor is at the position before the first uppercase `A`. Click the Find Next button, and inspect the results, as shown in Figure 11-12.
10. Click the Find Next button twice more, inspecting the result each time. Each time, the initial `ABC` on each line is highlighted.

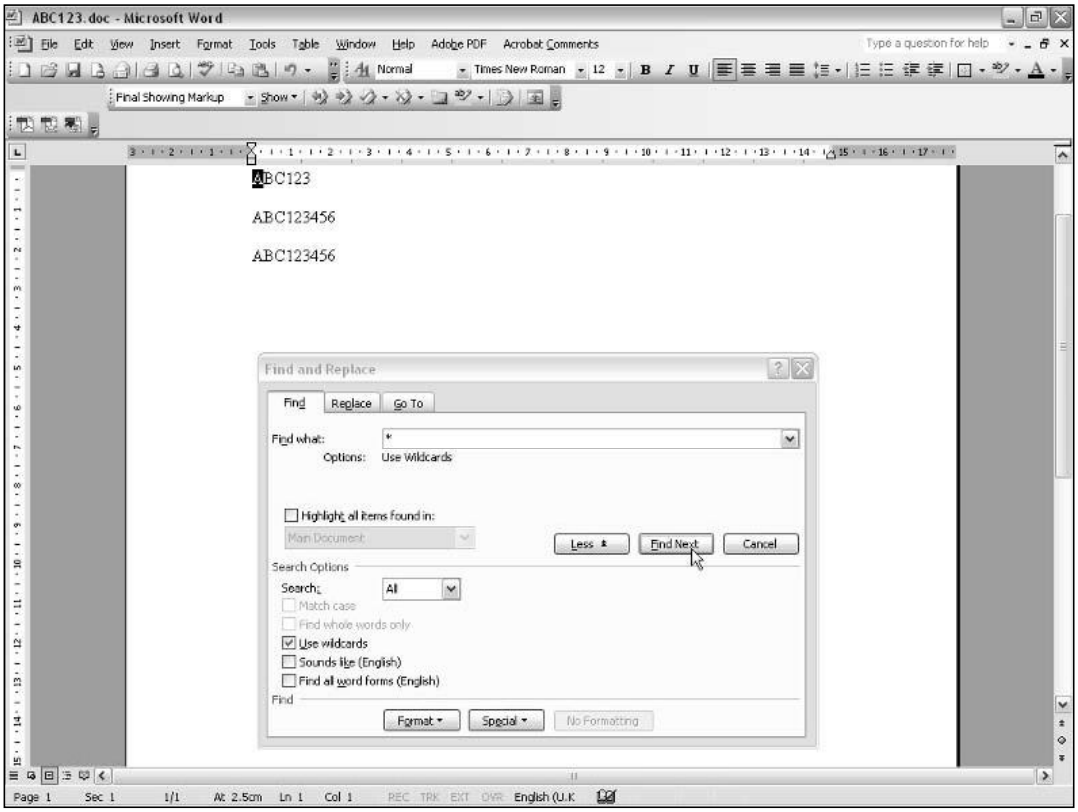

Figure 11-10

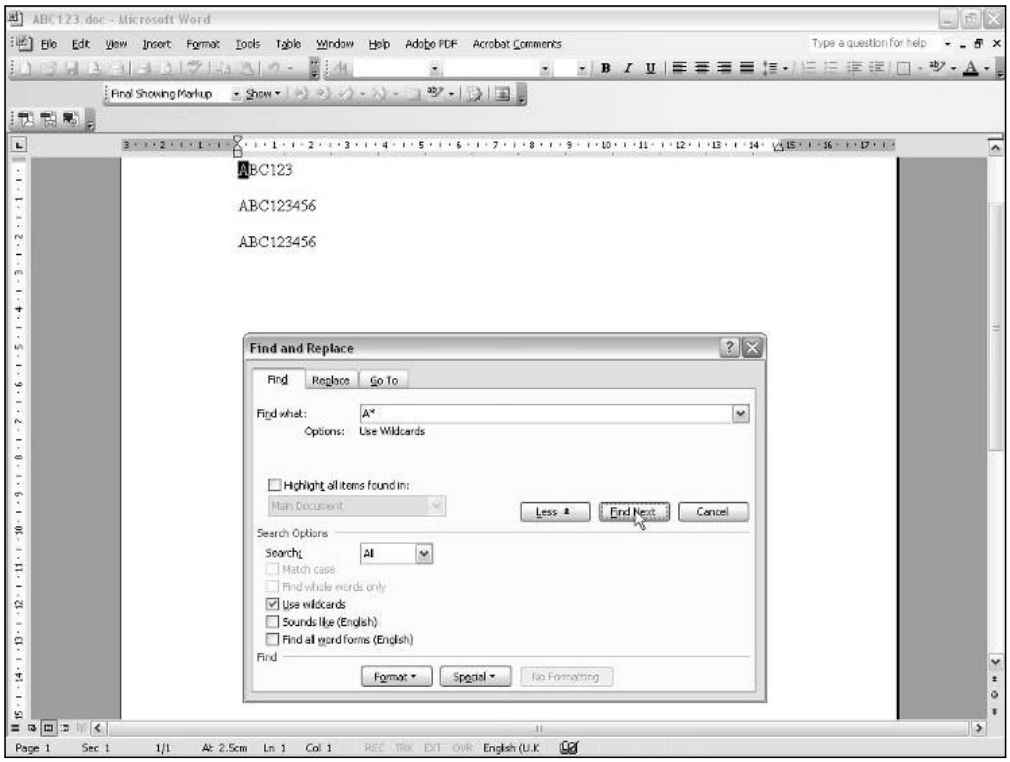

Figure 11-11

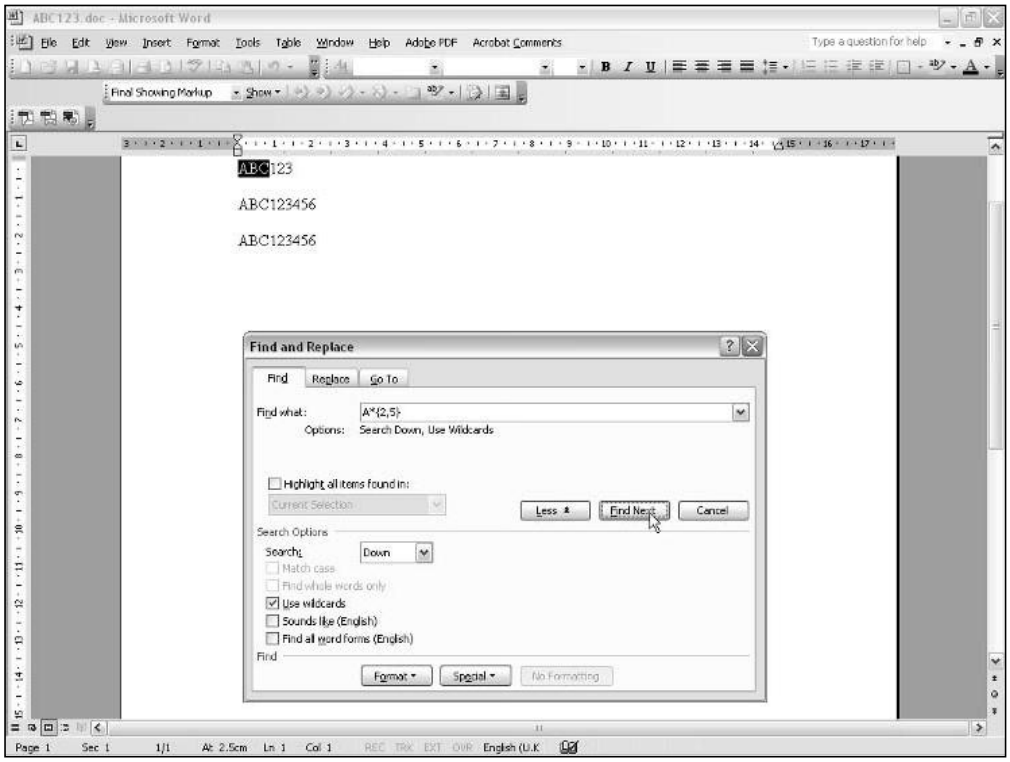

Figure 11-12

How It Works

The behavior when matching the `*` metacharacter on its own is odd. The `*` metacharacter is supposed to match zero or more characters, so if lazy matching were applied to the letter, it would match zero characters. In fact, it matches one character, as you saw in Figure 11-10.

If you use the `*` metacharacter on its own, which you won't do often, it behaves as anticipated for lazy matching. The pattern `A*` matches only the initial character `A` on each line. Assuming that the regular expression starts matching at the beginning of the first line, it matches the first character in the pattern, a literal uppercase `A`, with the first character of the line in the test text. Then it matches the minimum possible number of following characters, which is none. That is lazy matching.

Similarly, in the pattern `A*{2,5}`, only the minimum possible number of characters, `BC`, are matched.

Examples

Some further examples using wildcards in Word are provided here.

Character Class Examples, Including Ranges

Microsoft Word supports searches using character classes.

Word uses standard regular expression syntax for positive character classes. For example, to find the character sequences `gray` and `grey` in a document, `gray.doc`, you could use the following regular expression:

```
gr[ae]y
```

The content of `gray.doc` is shown here:

```
gray
grey
greying
greyed
grapple
grim
goat
filigree
great
groat
gloat
```

The character class `[ae]` specifies that the search is for a single character, which can be either `a` or `e`. There is nothing to restrict the regular expression to whole words, so you can expect the search to return the first four character sequences in `gray.doc`, because each of those words contains the literal characters `gr` followed by a character in the specified character class, followed in turn by the literal character `y`.

Whole Word Searches

It is a common requirement to find a sequence of characters that constitutes a whole word. You might, for example, search for the words (not merely the character sequences) `gray` and `grey`. The word boundary metacharacters < and > can be combined with other regular expression components.

Try It Out Whole Word Searches

The preceding test document, `Gray.doc`, will be used.

1. Open `Gray.doc` in Microsoft Word, and open the Find and Replace dialog box.
2. Check the Using Wildcards check box, and type the pattern **<gr[ae]y>** in the Find What text box.
3. Click the Find Next button three times, inspecting the text that is or is not matched each time. Figure 11-13 shows the appearance after Find Next is clicked the first time.

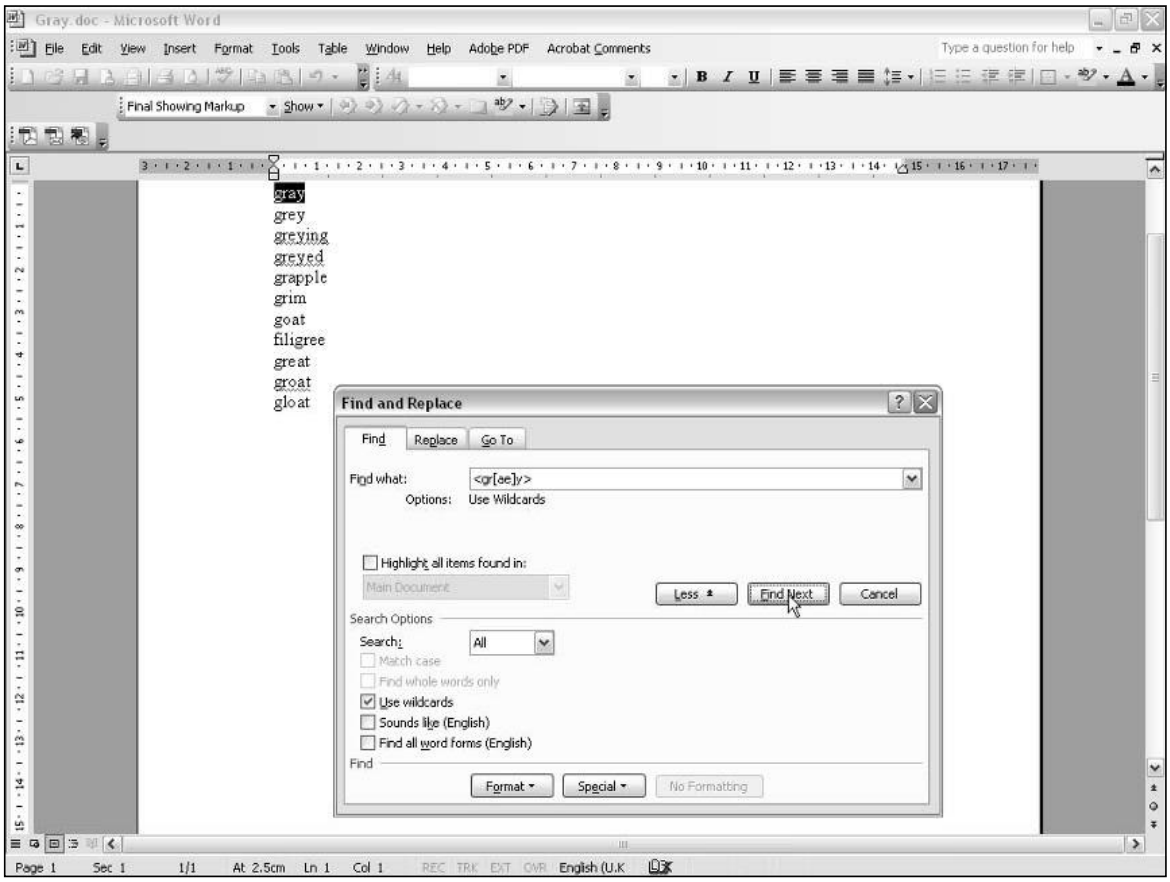

Figure 11-13

How It Works

The character sequence `gray` is matched because it is a word, rather than simply a character sequence. The `<` metacharacter matches the position before the `g` of `gray`. The literal `g` matches, as does the literal `r`. The character class `[ae]` includes a match for the `e`, which is the third character of the test text. The final `y` of the pattern matches the final `y` of the character sequence. After the `y` matches, the `>` metacharacter matches the end-of-line position.

The word `grey` matches for reasons explained in the preceding paragraph. However, the words `greying` and `greyed` do not match. The `<gr[ae]y` component of the regular expression matches, but matching fails on the `>` metacharacter. The character following the `y` in `greying` and in `greyed` is an alphabetic character, so there is no end-of-word boundary position.

Search-and-Replace Examples

The following examples illustrate how wildcards can be useful in a search-and-replace operation.

Changing Name Structure Using Back References

In Microsoft Word, you can use wildcards to reverse the structure of all the names in a document. For example, a name that originally appears as:

```
Fred Smith
```

becomes the following, after the use of wildcards, in a search-and-replace operation:

```
Smith, Fred
```

Achieving this depends on using back references.

The content of the sample document, `Names.doc`, is shown here:

```
Fred Smith
Alice Green
Barbara Kaplan
Ali Hussein
Paul Simenon
```

Notice that each name appears as a first name followed by a space character, followed by a surname. The consistency of that structure is important in the find operation.

Try It Out Changing Name Structure

1. Open `Names.doc` in Microsoft Word, and use the Ctrl+F keyboard shortcut to open the Find and Replace dialog box.
2. If the Search Options part of the Find and Replace window is not visible, click the More button to display the Search Options area. Check the Use Wildcards check box.

3. In the Find What text box, type the pattern **(<*>) (<*>)** (make sure that there is a space character between the first closing parenthesis and the second opening parenthesis).
4. In the Replace With text box, type the pattern **\2, \1** (make sure that there is a space character after the comma in the pattern).
5. Click the Find Next button. (The text `Fred Smith` should be highlighted.)
6. Click the Replace button. Inspect the result of the first replace operation. (The text `Fred Smith` should be replaced by `Smith, Fred`, as shown in Figure 11-14.)
7. Click the Replace All button, and inspect the results, as shown in Figure 11-15. All the names should be in the following form:

```
Surname, FirstName
```

How It Works

Let's look at how the pattern `(<*>) (<*>)` works. The two pairs of parentheses are there for grouping. The < metacharacter matches the position at the beginning of a sequence of alphabetic characters. The * metacharacter matches zero or more characters (equivalent to `.*` in more standard regular expression syntax). And the > metacharacter matches the position between the end of a sequence of alphabetic characters and the beginning of a sequence of nonalphabetic characters. So <*> matches a word of any length. When that is contained inside paired parentheses, grouping takes place.

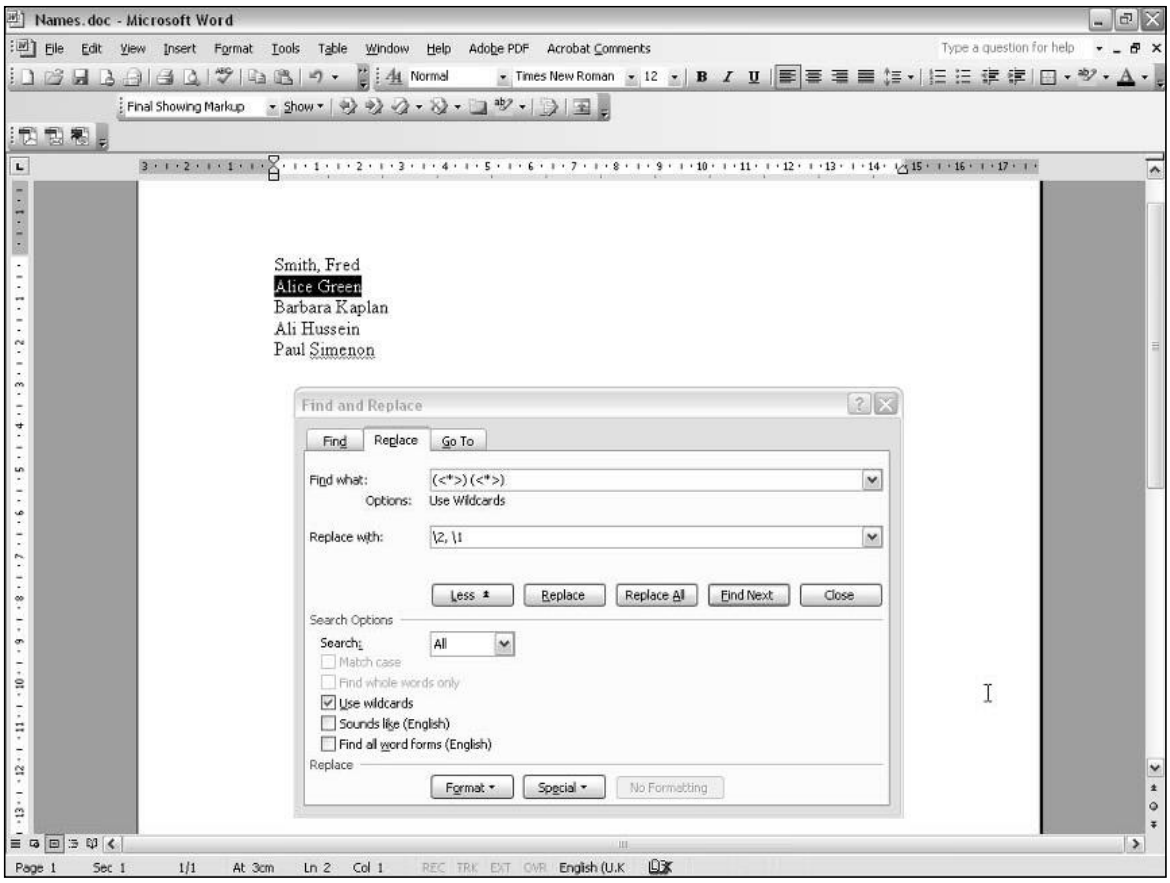

Figure 11-14

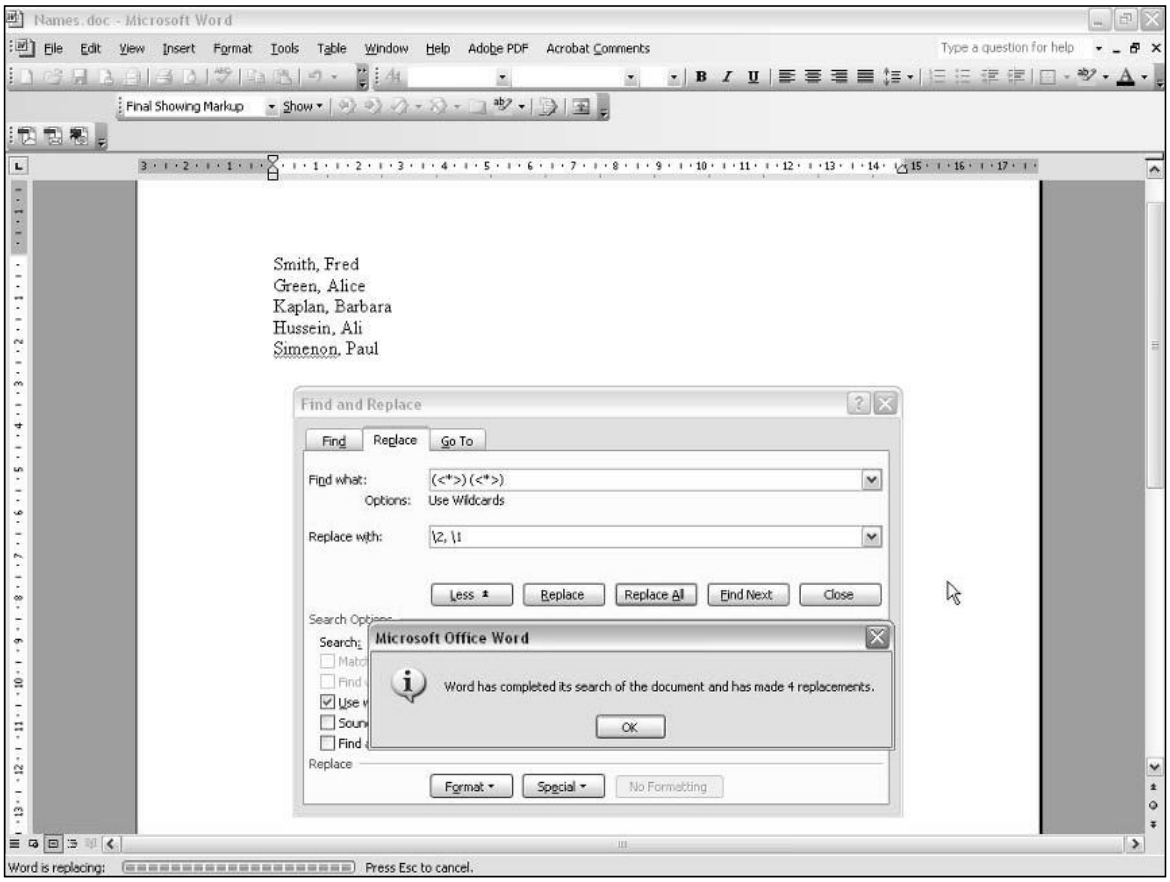

Figure 11-15

Because there is a space character between the first (<*>) pattern and the second, the pattern:

```
(<*>) (<*>)
```

matches two words separated by a single space character, which is the structure of the names in the file Names.doc. It also captures two groups, `\1` and `\2`, which are available for use in the replace operation.

The parentheses create two groups: `\1` contains the word before the space character, and `\2` contains the word that follows the space character. So the pattern:

```
\2, \1
```

replaces what had been two words separated by a space character, with the second word (indicated by `\2`), followed by a comma, followed by a space character, followed by what had been the first word (indicated by `\1`).

Manipulating Dates

Just as words such as `gray` (`grey`) and `license` (`licence`) differ in individual dialects of English, so does the structure of dates. In the United States, dates in numeric form are typically written as MM/DD/YYYY. In the United Kingdom, dates are written in the DD/MM/YYYY format. In Japan, the format YYYY/MM/DD is used. I have shown those formats using a forward slash as a separator, but other separators, such as a hyphen (dash) or a period character, are also used. In this example, I assume that the dates you want to manipulate are not written in the format 25 December 2005, 25 Dec 2005, or other similar formats.

The test document, `Dates.doc`, is shown here:

```
2005-12-25
12/11/2003
01.25.2006
03-18-2007
07-19-2004
2006/09/18
```

The dates in the test document consist of some in the International date format, YYYY-MM-DD, and some in U.S. date format, MM/DD/YYYY (with a range of separators). The task is to match U.S. date formats and replace them with the international date format equivalent. Because the international date format is used in XML, it is likely that an increasing amount of date-related information will be stored in that format.

We will assume that no dates are in U.K. date format, because the date on the second line would be ambiguous if U.K. dates were present. In U.S. date format, it would mean December 11, 2003, while in U.K. date format, it would mean 12 November 2003.

Try It Out — Conversion to International Date Format

1. Open `Dates.doc` in Microsoft Word.
2. Open the Find and Replace dialog box using Ctrl+F, and check the Use Wildcards check box.

   First, we will check whether we can create a regular expression to find the dates that are in U.S. date format.
3. Type the pattern **([0-9]{2})[./-]([0-9]{2})[./-]([0-9]{4})** in the Find What text box, click the Find Next button four times, and inspect the highlighted text.

   If all has gone well, the pattern should match the dates that are in U.S. format but not those that are already in international date format. We have assumed that all entries are valid dates. The pattern shown would also match nonsense character sequences such as `44/12/5432`.

   Figure 11-16 shows the text's appearance after Step 3, when the Find Next button has been clicked four times.
4. Switch to the Replace tab of the Find and Replace dialog box, and type the pattern **\3-\1-\2** in the Replace With text box.

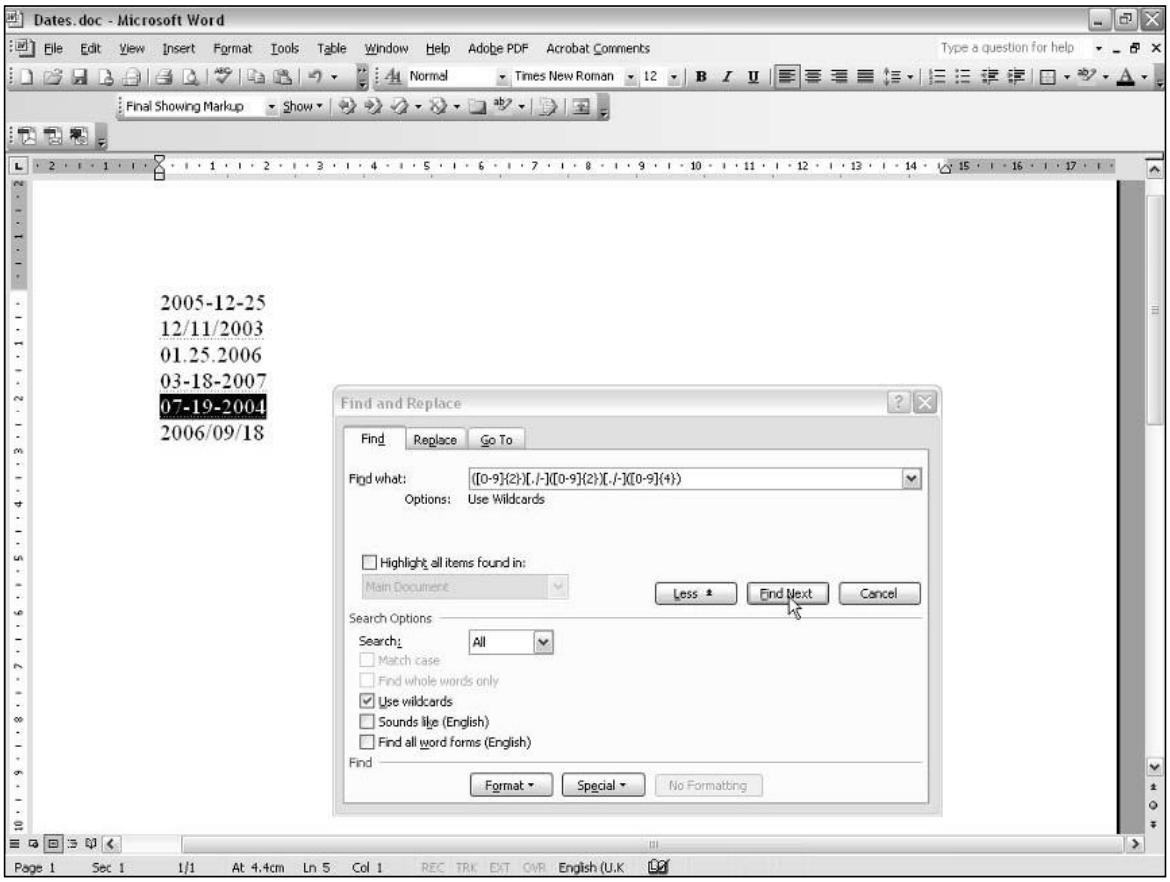

Figure 11-16

5. Click Find Next. Click Replace. Confirm that the change has produced an appropriately formatted date — that is, a date in international date format.

6. Repeat Step 5 three times to replace all occurrences of dates in U.S. date format. Figure 11-17 shows the appearance after Step 6 has been completed.

How It Works

Let's break the pattern `([0-9]{2})[./-]([0-9]{2})[./-]([0-9]{4})` into its component parts to understand what it is doing.

The `([0-9]{2})` component matches a character sequence made up of two successive numeric digits. We have to use the character class `[0-9]` to match numeric digits, because Microsoft Word wildcards don't have a metacharacter that is specific to numeric digits. The parentheses are grouping parentheses, creating the group `\1`, which contains the date's month component.

The `[./-]` component will match dates that use the hyphen, forward slash, or period character as the separator. Be careful not to write this character class as [.-/], because written that way, the hyphen creates a range that won't give you the desired results.

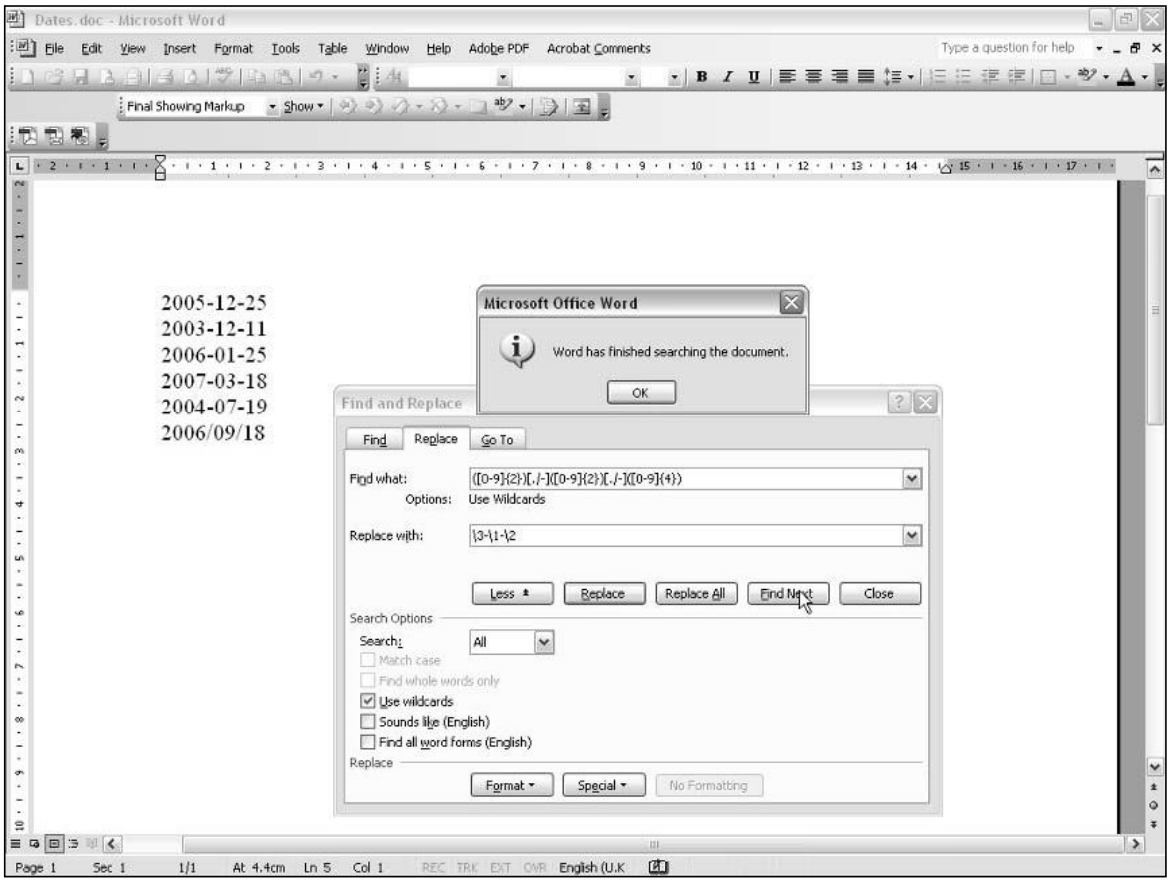

Figure 11-17

The components `([0-9]{2})[./-]` do the same as the first two components of the pattern, except that the grouping parentheses create group `\2`, which contains the date's day component.

The `([0-9]{4})` component of the pattern matches a character sequence of four numeric digits. The grouping parentheses create group `\3`, which contains the date's year component.

The replacement pattern, `\3-\1-\2`, reorders the date. The `\3` group, which contains the year component, is followed by a literal hyphen, which is in turn followed by the `\1` group, which contains the month component of the date. Another literal hyphen follows and, finally, the `\2` group containing the date's day component.

The Star Training Company Example

Microsoft Word has no lookahead functionality, so we can't use that to carry out a search and replace of the Star Training Company document in Word. However, we can make use of back references to achieve what we want, although not quite so elegantly as we might have if lookahead were available in Word.

Try It Out Star Training Company Example

1. Open `StarOriginal.doc` in Microsoft Word, and open the Find and Replace dialog box.
2. Switch to the Replace tab. Because of the absence of alternation (there is no | metacharacter available in Word), we have to carry out the replacement in two stages.
3. In the Find What text box, type the pattern **(Star)(Training)**, making sure that there is a space character inside the parentheses before the uppercase `T` of `Training`.
4. Click Find Next several times, inspecting the result each time. Confirm that each occurrence of the character sequence `Star Training` is matched.
5. In the Replace With text box, type the pattern **Moon\2**. Be careful to avoid inserting a space character between `Moon` and `\2`. The space character was captured inside the group `\2`, so you don't need to add one here.
6. Use the Find Next and Replace buttons to step through the document, replacing each occurrence of the character sequence `Star Training`. Figure 11-18 shows the screen's appearance when the first occurrence of `Star Training` has already been replaced.

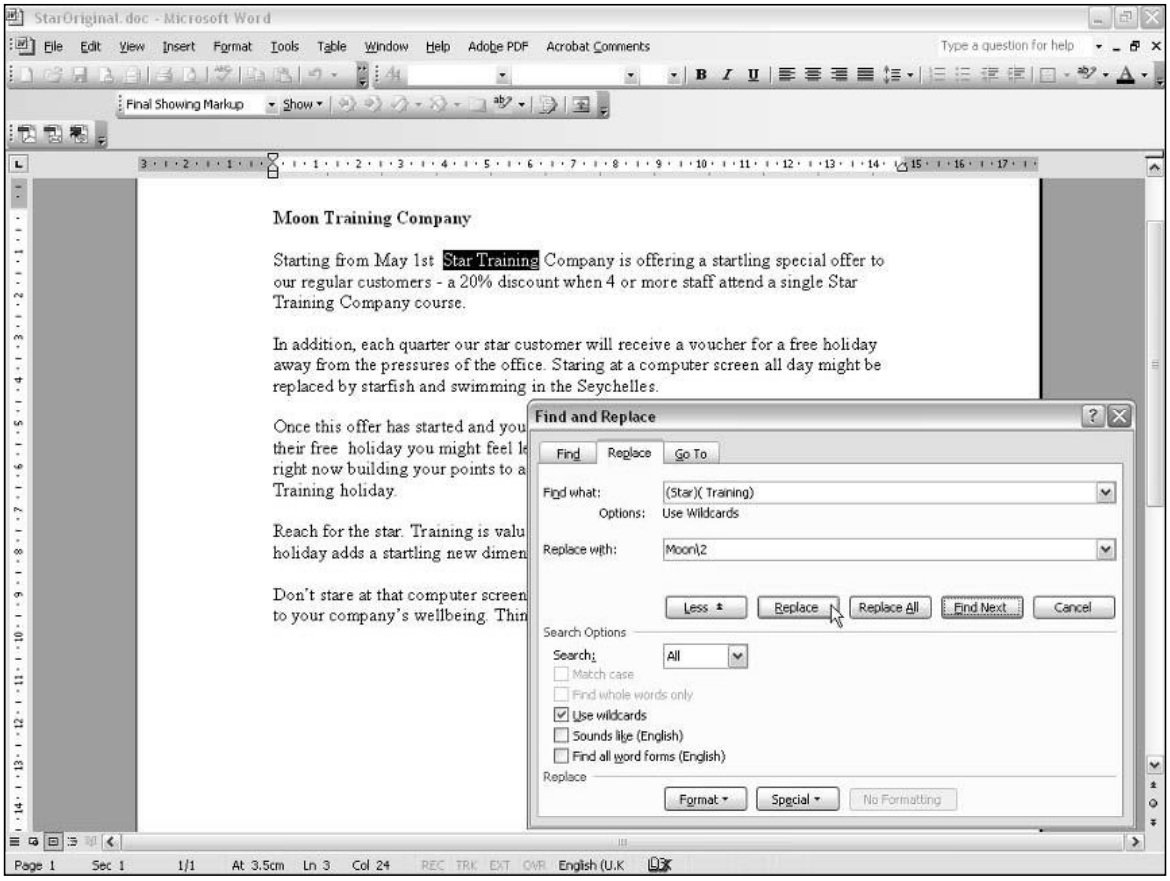

Figure 11-18

7. Edit the pattern in the Find What text box to `(Star)([.])`. This slightly unusual pattern is necessary in Word, as explained later.

8. You can leave the pattern in the Replace With text box unchanged. Use the Find Next and Replace buttons to replace the pattern `Star.` with `Moon.`. Figure 11-19 shows the appearance when the first occurrence of the character sequence `Star.` is about to be replaced.

How It Works

First, let's look at how the pattern `(Star)( Training)` matches. Two groups are created using this approach. We have created a group with `(Star)` and a second group that contains a space character followed by the character sequence `Training`. By capturing group `\2` and simply putting it in the replace string we are, in effect, creating a workaround for the lack of lookahead functionality. The character sequence `Star` is replaced with `Moon`, and we replace the content of `\2` with itself.

The pattern `(Star)([.])` creates two groups. The first contains the character sequence `Star`, and the second, a period character. In most regular expression implementations, you wouldn't have to enclose the period character inside a character class. You would likely use the `\.` metacharacter instead. However, in Word you have to enclose the period character inside a character class to avoid an error message. Occurrences of `Star` before a period character are replaced by `Moon`.

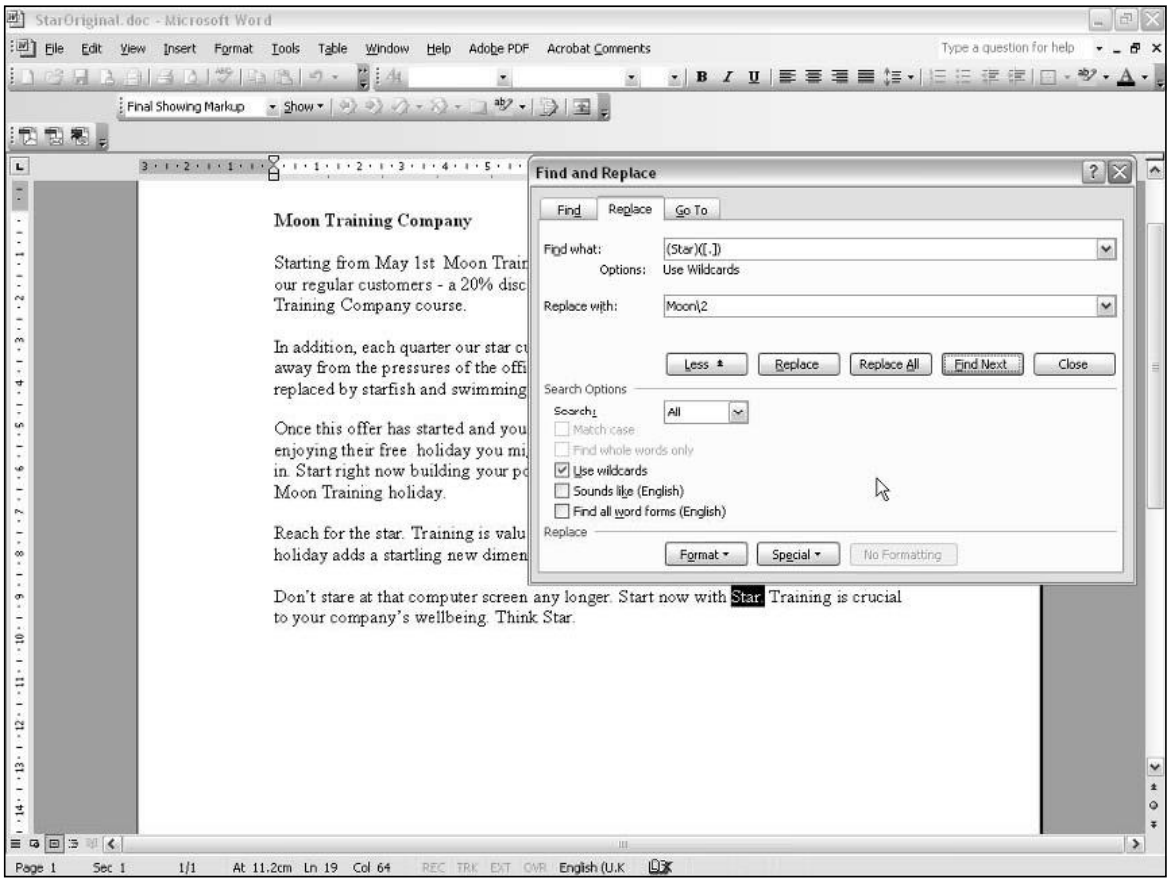

Figure 11-19

The resulting document, `StarWhichIsNowMoon.doc`, with all relevant occurrences of the character sequence `Star` replaced using the two-step replace just described, is shown here:

```
Moon Training Company

Starting from May 1st  Moon Training Company is offering a startling special offer
to our regular customers - a 20% discount when 4 or more staff attend a single Moon
Training Company course.

In addition, each quarter our star customer will receive a voucher for a free
holiday away from the pressures of the office. Staring at a computer screen all day
might be replaced by starfish and swimming in the Seychelles.

Once this offer has started and you hear about other Moon Training customers
enjoying their free  holiday you might feel left out. Don't be left on the outside
staring in. Start right now building your points to allow you to start out on your
very own Moon Training holiday.

Reach for the star. Training is valuable in its own right but the possibility of a
free holiday adds a startling new dimension to the benefits of Moon Training
training.

Don't stare at that computer screen any longer. Start now with Moon. Training is
crucial to your company's wellbeing. Think Moon.
```

We have replaced all the relevant occurrences of `Star`. It's a big improvement over the novice attempt in Chapter 1, achieved with high sensitivity and high specificity.

Regular Expressions in Visual Basic for Applications

Visual Basic for Applications (VBA) allows the programmatic use of regular expressions (wildcards) in ways similar to those available from the Microsoft Word interface. The following Word macro, `StarToMoon`, does the same:

```
Sub StarToMoon()
'
' StarToMoon Macro
' Macro recorded 19/07/2004 by Andrew Watt
'
    Selection.Find.ClearFormatting
    Selection.Find.Replacement.ClearFormatting
    With Selection.Find
        .Text = "(Star)( Training)"
        .Replacement.Text = "Moon\2"
        .Forward = True
        .Wrap = wdFindContinue
        .Format = False
        .MatchCase = False
        .MatchWholeWord = False
        .MatchAllWordForms = False
```

```
        .MatchSoundsLike = False
        .MatchWildcards = True
    End With
    Selection.Find.Execute Replace:=wdReplaceAll
    With Selection.Find
        .Text = "(Star)(.)"
        .Replacement.Text = "Moon\2"
        .Forward = True
        .Wrap = wdFindContinue
        .Format = False
        .MatchCase = False
        .MatchWholeWord = False
        .MatchAllWordForms = False
        .MatchSoundsLike = False
        .MatchWildcards = True
    End With
    Selection.Find.Execute
    With Selection
        If .Find.Forward = True Then
            .Collapse Direction:=wdCollapseStart
        Else
            .Collapse Direction:=wdCollapseEnd
        End If
        .Find.Execute Replace:=wdReplaceOne
        If .Find.Forward = True Then
            .Collapse Direction:=wdCollapseEnd
        Else
            .Collapse Direction:=wdCollapseStart
        End If
        .Find.Execute
    End With
    With Selection
        If .Find.Forward = True Then
            .Collapse Direction:=wdCollapseStart
        Else
            .Collapse Direction:=wdCollapseEnd
        End If
        .Find.Execute Replace:=wdReplaceOne
        If .Find.Forward = True Then
            .Collapse Direction:=wdCollapseEnd
        Else
            .Collapse Direction:=wdCollapseStart
        End If
        .Find.Execute
    End With
End Sub
```

It isn't the purpose of this chapter to teach how to create Word macros, but it can be useful to have find-and-replace functionality inside a Word macro.

The file `StarOriginalWithMacro.doc` contains the preceding macro, which you can execute.

The file `StarAfterMacro.doc` contains the file after replacement of the text using the macro.

To run the `StarToMoon` macro, you may need to adjust your macro security settings in Word.

Please note that the machine on which the macro was created was believed to be free of viruses. However, you may wish to check using your own antivirus software to ensure that it remains virus free.

Exercises

The following exercises allow you to test your understanding of the material introduced in this chapter:

1. Create a pattern, that, in Word, will match the character sequences `peak` and `peek`.
2. Modify the date example so that it will take dates in U.K. date format and convert them to international date format.