Regular Expressions for Technical Writers

TECHNICAL COMMUNICATION SUMMIT'17

STC's 64th Annual Conference

7-10 May 2017 • Washington, DC

STC Summit 2017 - Washington DC

Scott Prentice, Leximation, Inc.

Introduction

- Scott Prentice, President of Leximation, Inc.
- Specializing in FrameMaker plugin development as well as structured FrameMaker conversions, consulting, and development. FrameMaker user/developer since 1991.
- Developed DITA-FMx, a FrameMaker plugin for efficient DITA authoring and publishing.
- Consulting for custom Help systems, creative/functional web applications, and EPUB solutions.

Disclaimer

- This information is not exhaustive or complete
- Will discuss regular expression features that may be most useful to technical writers
- Designed for beginners or infrequent users
- (However, some advanced topics are discussed)

Regular expression?

- Regular expression, AKA “regex”
- Text string describing a search pattern
- Way beyond wildcards
- May also define a replacement string
- Replacement may contain content extracted from match
- Like a mini programming language

Where can you use a regex?

- Many authoring tools provide regex support
- Most “serious” text editors
- Scripting languages like Perl, PHP, JavaScript, Python, Ruby
- Unix utilities like grep, sed, and awk
- Compiled programming languages like Java, C#, VB.NET
- Anything with a “regex engine”!

Benefits

- Powerful searching
- Complex string replacements and intelligent modifications
- Powerful syntax in very few characters
- Text format conversions (this is huge)
    - HTML or XML to CSV (or the other way around)
    - HTML or XML cleanup

Problems?

- Can appear very complex and overwhelming
- Regex syntax varies based on the “engine” and implementation
- Watch out for “greedy” matches
- Typically no “one right way” to do the same thing
- Some people say you shouldn’t parse XML with a regex; as long as you understand the limitations it’s fine

Regex basics

- Literal characters – `z`, `zorch`, `FOO`, `F00`
- Metacharacters – `\s`, `\S`, `\w`, `\W`, `\d`, `\D`
- Anchors/boundaries – `^`, `$`, `\b`, `\B`
- Quantifiers – `*`, `+`, `?`, `{2}`, `{3,5}`, `{3,}`
- Grouping – `.`, `(…)`, `(…|…)`, `[…]`, `[…-…]`, `[^…]`

Basic regex examples

- Find the word .. "cat" (lowercase) – `\bcat\b`
- .. "cat" or "dog" (lowercase) – `\b(cat|dog)\b`
- .. "Cat" or "cat" – `\b[Cc]at\b`
- .. "cat" followed by numbers – `\bcat[0-9]+\b`
- .. that contains "cat" – `\Bcat\B`
- .. that starts with "cat" or "Cat" – `\b[Cc]at\B`

Modifiers

- Common modifiers (options) in many tools
  - g - global replace
  - i - case insensitive match
  - m - multiline mode (treats each line separately)
  - s - single-line mode ("dot matches all", includes `\r\n`)
  - x - free-spacing mode (comments follow "#")
- Inline use: `(?imsx)` enables, `(?-imsx)` disables

Naturally "greedy"

- Regexes will typically match on as much as possible
- Need to add code for minimal match
- Use `?` for a minimal match - `this .*? that`
- Match any char except ">" - `[^>]+`
- Use multiline mode (if possible) `(?m)`

Captures / Backreferences

- Parenthesis define a capture group
- Matched content is passed to the numeric backreference
- Find any word followed by the same word:

  ```
  (\w+)\s+\1
  ```
- Attributes in HTML may be in single or double quotes:

  ```
  class=(["']).+?\1
  ```
- Tools use `\1` or `$1` to identify the captured string

Date regex examples

- Match date in the form of yyyy-mm-dd or yyyy/mm/dd

  `\b\d{4}[/-]\d\d?[/-]\d\d?\b`

  or ..

  `\b\d{4}([/-]\d\d?){2}\b`

- Change format of date string to mm/dd/yyyy .. match:

  `\b(\d{4})[/-](\d\d?)[/-](\d\d?)\b`

  replace:

  `$2/$3/$1`

HTML/XML regex examples

- Extract the element name to **`$1`** –

  `<([\w-]+)[^>]*>`

- Extract the @class attribute value to **`$1`** –

  `<[\w-]+[^>]*class="([^"]+)"[^>]*>`

- Extract content from the element to **`$2`** –

  `<([\w-]+)[^>]*>(.+)?</\1>`

Where to start?

- Start simple, really simple .. get used to your editor
- Match on some literal characters
- Match on string of a specific length
- Try extracting and replacing portions of strings
- Use a text editor and match on some code, HTML, CSV, or whatever you're likely to encounter

Tool-specific issues

- Adobe FrameMaker
- Adobe RoboHelp
- Microsoft Word
- MadCap Flare
- Oxygen XML
- Text editors and scripting languages

General differences

- Text/code editors are line-based
- Authoring tools are paragraph-oriented
- Default may be single-line or multiline mode
- Not all modifiers are available in all tools (try inline)
- Use `$1` or `\1` format for capture replacement match?
- Tool may or may not support backreferences

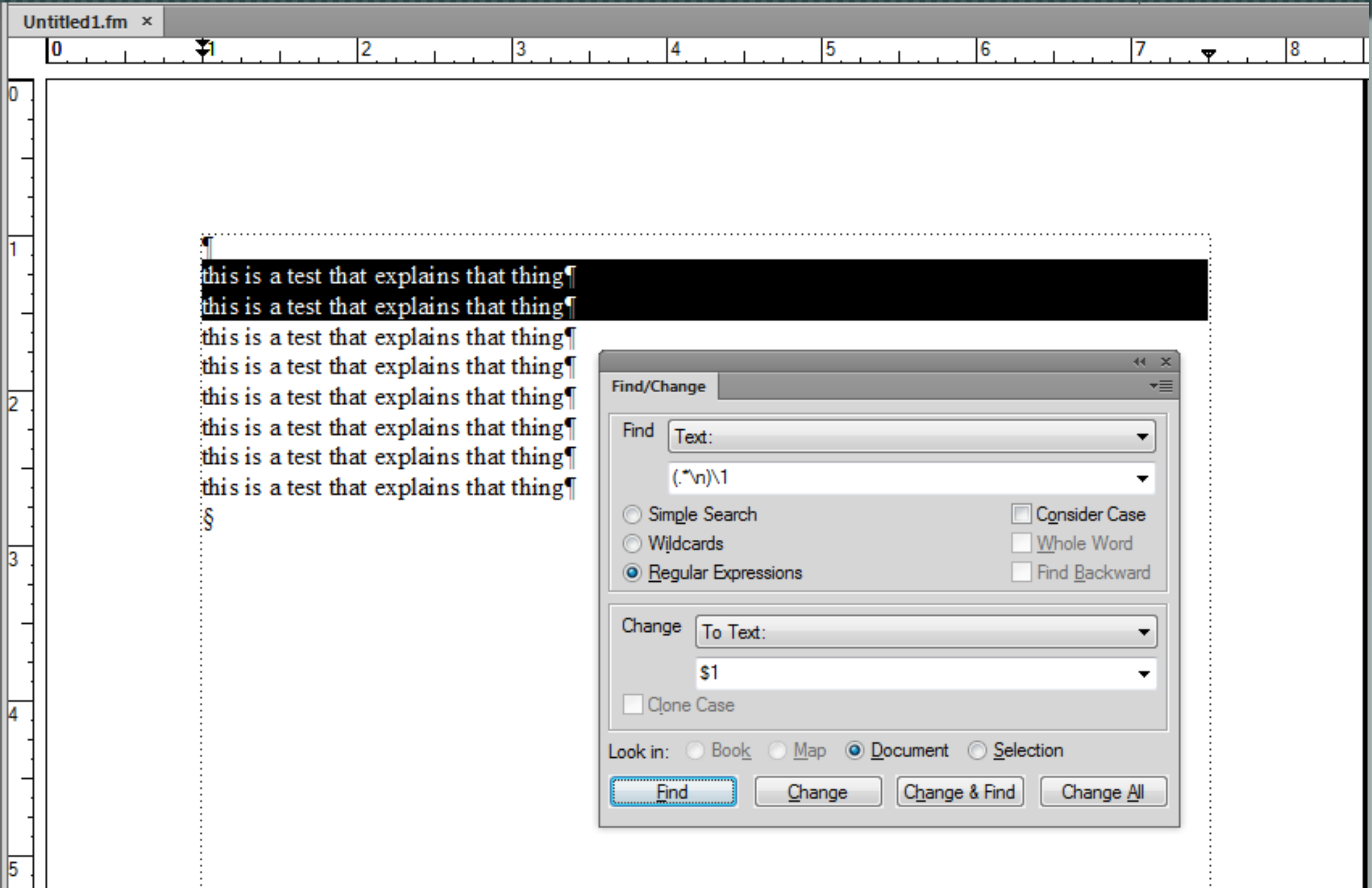
FrameMaker (unstructured)

- Enable single-line mode with inline modifier `(?s)`
- Match: `\n` for EOL, `\x09` for line break (not `\r`), `\t` or `\x08` for tab
- Replace: `\r` or `\x09` for line break, `\x08` for tab
- Use `$1` format for captured replacement value
- maker.ini setting RegularExpressionSyntax for engine

FrameMaker (structured)

- No single-line mode; inline modifiers not supported
- Each node defines a “line” (match cannot span nodes)
- Use `\n` to match EOL (but that’s all it’ll match)
- Use `$1` format for captured replacement value
- In XML View, use “Complex Expressions” option (limited features)

FrameMaker



```
¶
this is a test that explains that thing¶
this is a test that explains that thing¶
this is a test that explains that thing¶
this is a test that explains that thing¶
this is a test that explains that thing¶
this is a test that explains that thing¶
this is a test that explains that thing¶
this is a test that explains that thing¶
§
```

Find/Change

Find: Text:

```
(.*\n)\1
```

Simple Search
Wildcards
[✓ Regular Expressions]
Consider Case
Whole Word
Find Backward

Change: To Text:

```
$1
```

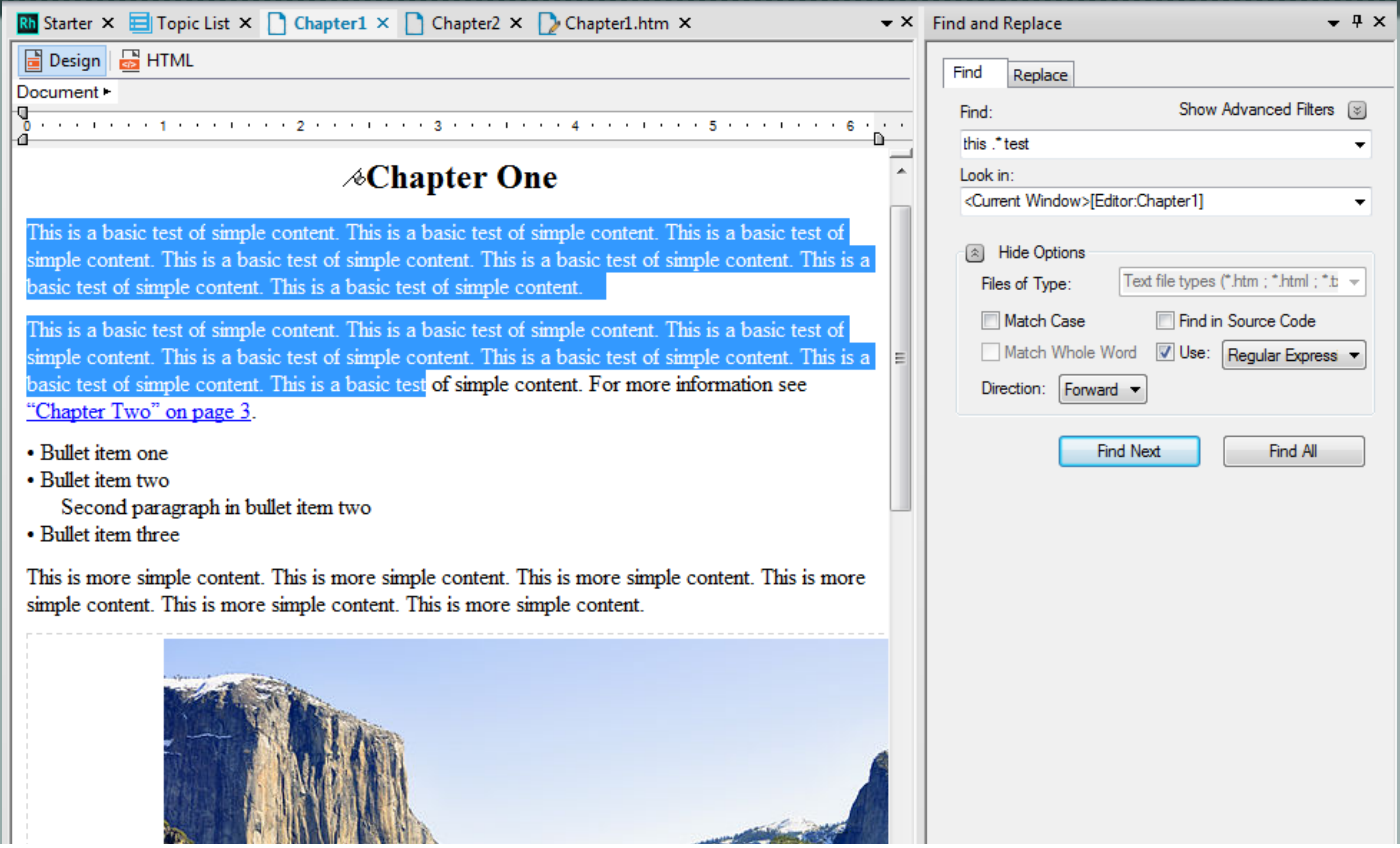
Clone Case

Look in: Book Map [✓ Document] Selection

RoboHelp

- Single-line mode is default in design view
- Multiline mode is default in source code view
- Inline modifiers not allowed, no capture group replacements
- Uses “Microsoft-style” regular expressions (??)
- Newline (`\n`) only matches in code view
- Supports find/replace in files

RoboHelp



Find: this .*test

Look in: <Current Window>[Editor:Chapter1]

Hide Options

Files of Type: Text file types (*.htm ; *.html ; *.t

Match Case [ ]

Find in Source Code [ ]

Match Whole Word [ ]

Use: [✓] Regular Express

Direction: Forward

Chapter One

This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content.

This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. This is a basic test of simple content. For more information see "Chapter Two" on page 3.

- Bullet item one
- Bullet item two

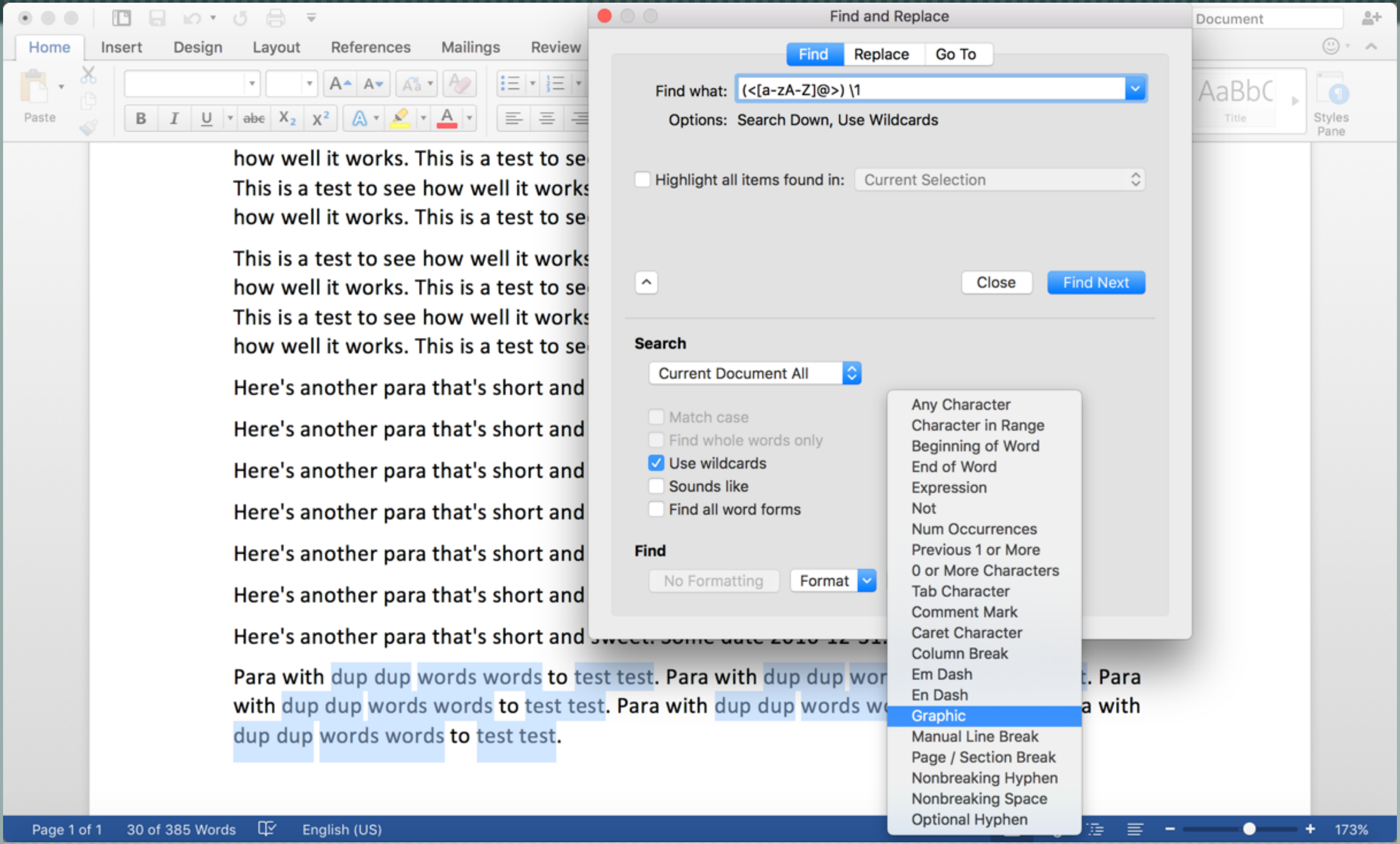
  Second paragraph in bullet item two
- Bullet item three

This is more simple content. This is more simple content. This is more simple content. This is more simple content. This is more simple content. This is more simple content.

MS Word

- Special MS hybrid regex/wildcard syntax; not “real”
- The `*` matches anything except EOL (non-greedy), and `@` after a char or char class matches one or more
- Use `^13` to find a paragraph mark and replace with `^p` (replacing with `^13` can be bad)
- Find duplicate paras – `(*^13)\1`
- Find duplicate “words” – `(<[a-zA-Z0-9]@>) \1`

MS Word



Find and Replace

Find what: (<[a-zA-Z]@>) \1

Options: Search Down, Use Wildcards

Highlight all items found in: Current Selection

Search: Current Document All

- [ ] Match case
- [ ] Find whole words only
- [✓] Use wildcards
- [ ] Sounds like
- [ ] Find all word forms

how well it works. This is a test to se
This is a test to see how well it works
how well it works. This is a test to se

This is a test to see how well it works
how well it works. This is a test to se
This is a test to see how well it works
how well it works. This is a test to se

Here's another para that's short and

Here's another para that's short and

Here's another para that's short and

Here's another para that's short and

Here's another para that's short and

Here's another para that's short and

Here's another para that's short and

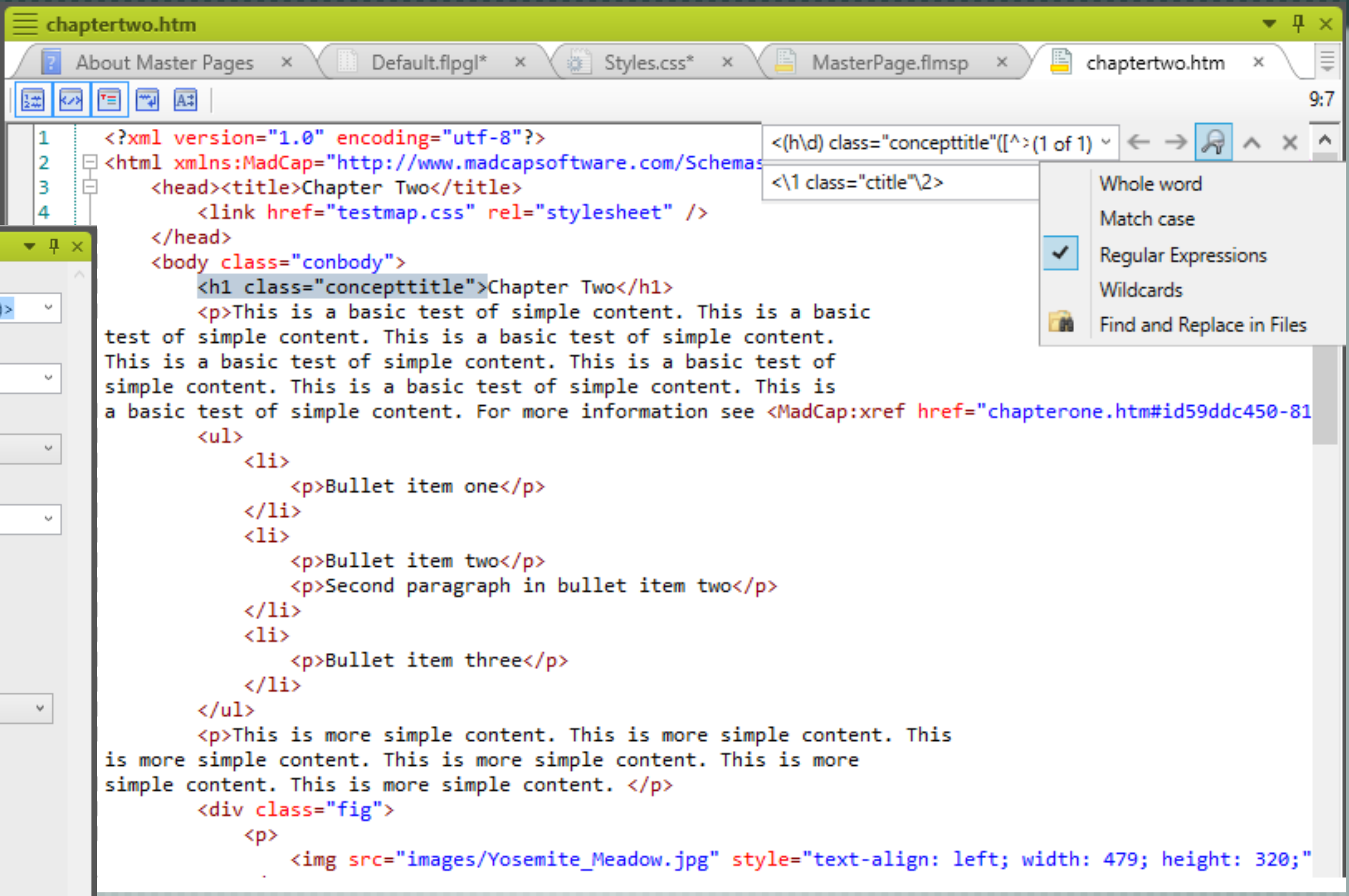
Para with dup dup words words to test test. Para with dup dup wor . Para with dup dup words words to test test. Para with dup dup words w a with dup dup words words to test test.

Page 1 of 1 30 of 385 Words English (US) 173%

Flare

- Best to use regexes in code view, seems unreliable in XML Editor view (search is done on underlying code)
- No single-line mode; inline modifiers not supported
- Use `\1` format for captured replacement value
- Supports find/replace in files

Flare



chaptertwo.htm

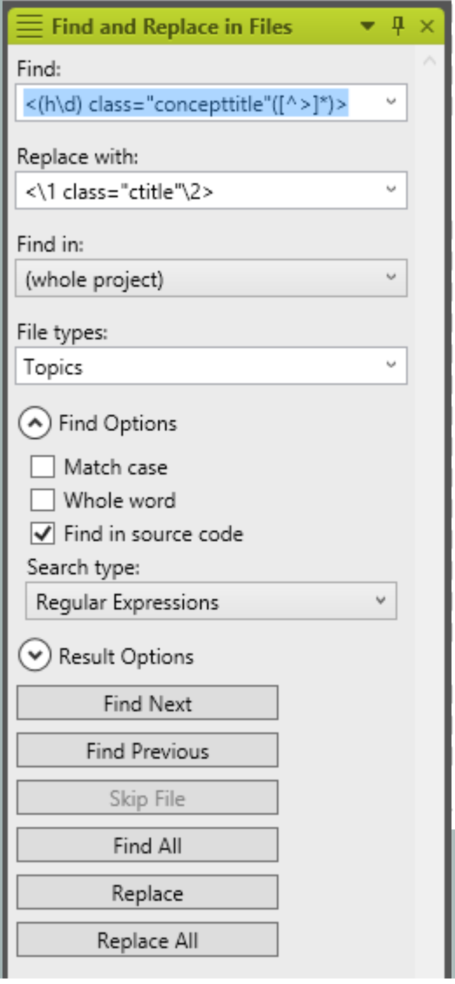
About Master Pages | Default.flpgl* | Styles.css* | MasterPage.flmsp | chaptertwo.htm

9:7

Find: `<(h\d) class="concepttitle"([^>]*)>` (1 of 1)

Replace: `<\1 class="ctitle"\2>`

```
<?xml version="1.0" encoding="utf-8"?>
<html xmlns:MadCap="http://www.madcapsoftware.com/Schemas
    <head><title>Chapter Two</title>
        <link href="testmap.css" rel="stylesheet" />
    </head>
    <body class="conbody">
        <h1 class="concepttitle">Chapter Two</h1>
        <p>This is a basic test of simple content. This is a basic
test of simple content. This is a basic test of simple content.
This is a basic test of simple content. This is a basic test of
simple content. This is a basic test of simple content. This is
a basic test of simple content. For more information see <MadCap:xref href="chapterone.htm#id59ddc450-81
        <ul>
            <li>
                <p>Bullet item one</p>
            </li>
            <li>
                <p>Bullet item two</p>
                <p>Second paragraph in bullet item two</p>
            </li>
            <li>
                <p>Bullet item three</p>
            </li>
        </ul>
        <p>This is more simple content. This is more simple content. This
is more simple content. This is more simple content. This is more
simple content. This is more simple content. </p>
        <div class="fig">
            <p>
                <img src="images/Yosemite_Meadow.jpg" style="text-align: left; width: 479; height: 320;"
```



Find and Replace in Files

Find: `<(h\d) class="concepttitle"([^>]*)>`

Replace with: `<\1 class="ctitle"\2>`

Find in: (whole project)

File types: Topics

Find Options

- [ ] Match case
- [ ] Whole word
- [x] Find in source code

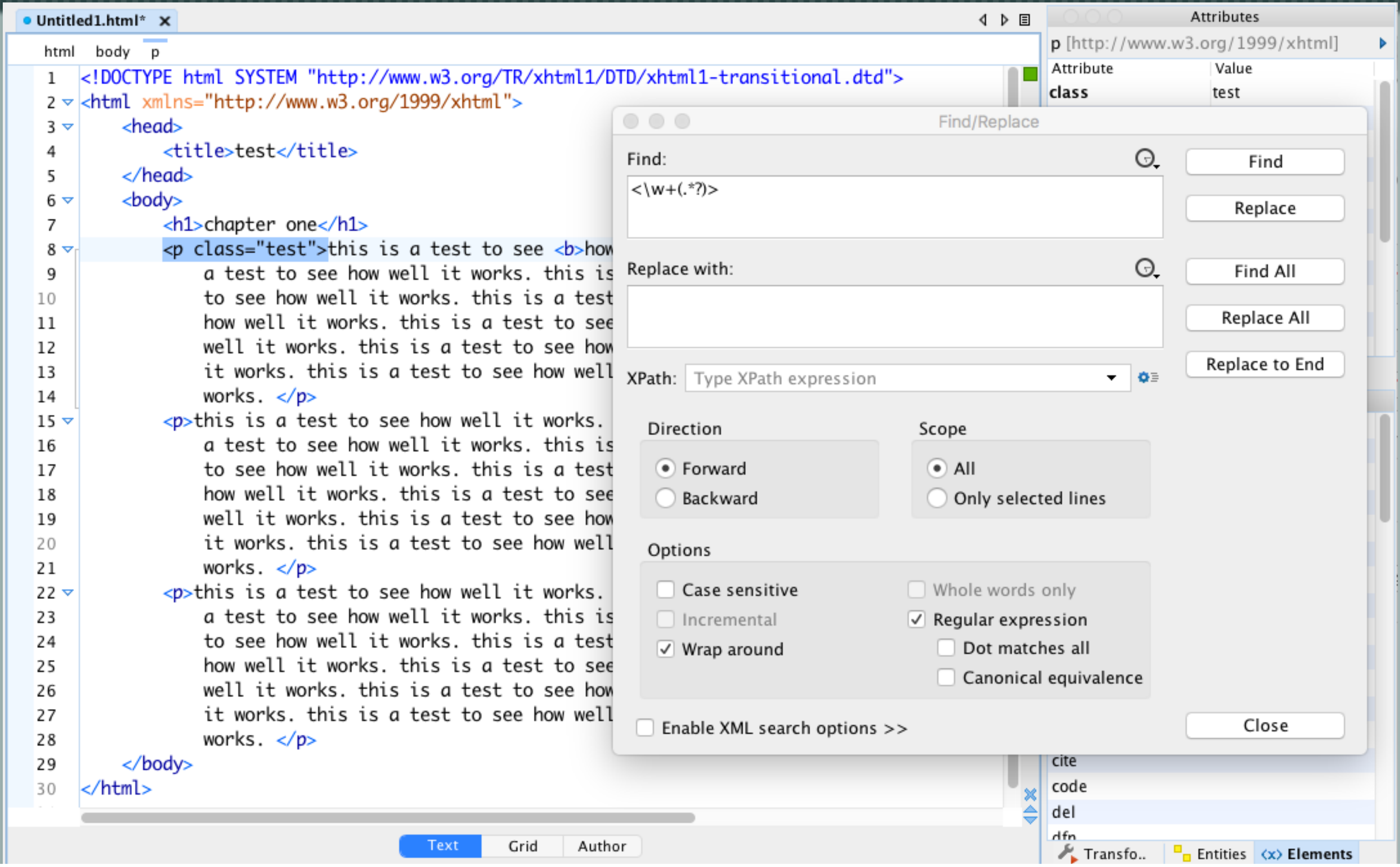
Search type: Regular Expressions

Result Options

OxygenXML

- In author view, matches are limited to “block-level” (?)
- In code view, enable single-line mode with “dot matches all” option
- Use `\1` format for captured replacement value
- Supports find/replace in files

OxygenXML



```
<!DOCTYPE html SYSTEM "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>test</title>
    </head>
    <body>
        <h1>chapter one</h1>
        <p class="test">this is a test to see <b>how
            a test to see how well it works. this is
            to see how well it works. this is a test
            how well it works. this is a test to see
            well it works. this is a test to see how
            it works. this is a test to see how well
            works. </p>
        <p>this is a test to see how well it works.
            a test to see how well it works. this is
            to see how well it works. this is a test
            how well it works. this is a test to see
            well it works. this is a test to see how
            it works. this is a test to see how well
            works. </p>
        <p>this is a test to see how well it works.
            a test to see how well it works. this is
            to see how well it works. this is a test
            how well it works. this is a test to see
            well it works. this is a test to see how
            it works. this is a test to see how well
            works. </p>
    </body>
</html>
```

p [http://www.w3.org/1999/xhtml]

Attribute	Value
class	test

Find/Replace

Find: `<\w+(.*?)>`

Replace with:

XPath: Type XPath expression

Direction: [● Forward] ○ Backward

Scope: [● All] ○ Only selected lines

Options:
- ☐ Case sensitive
- ☐ Incremental
- [✓] Wrap around
- ☐ Whole words only
- [✓] Regular expression
- ☐ Dot matches all
- ☐ Canonical equivalence

☐ Enable XML search options >>

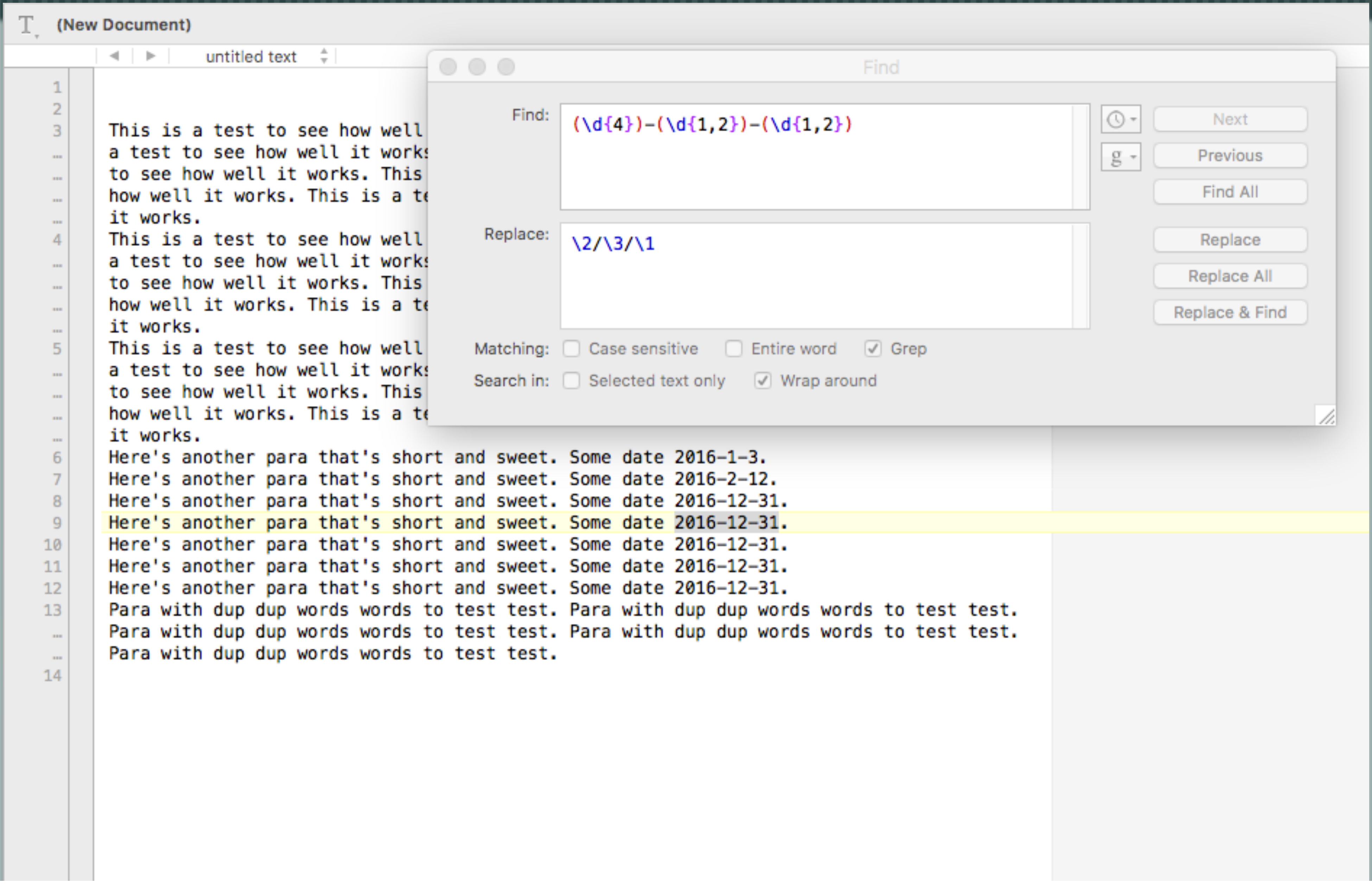
Find | Replace | Find All | Replace All | Replace to End | Close

cite
code
del
dfn

TextWrangler

- Choose "grep" option to perform regex search/replace
- Enable single-line mode with inline modifier `(?s)`
- Use `\1` format for captured replacement value
- Supports find/replace in files

TextWrangler



Find: `(\d{4})-(\d{1,2})-(\d{1,2})`

Replace: `\2/\3/\1`

Matching: [ ] Case sensitive [ ] Entire word [✓ Grep]

Search in: [ ] Selected text only [✓ Wrap around]

```
This is a test to see how well
a test to see how well it works
to see how well it works. This
how well it works. This is a te
it works.
This is a test to see how well
a test to see how well it works
to see how well it works. This
how well it works. This is a te
it works.
This is a test to see how well
a test to see how well it works
to see how well it works. This
how well it works. This is a te
it works.
Here's another para that's short and sweet. Some date 2016-1-3.
Here's another para that's short and sweet. Some date 2016-2-12.
Here's another para that's short and sweet. Some date 2016-12-31.
Here's another para that's short and sweet. Some date 2016-12-31.
Here's another para that's short and sweet. Some date 2016-12-31.
Here's another para that's short and sweet. Some date 2016-12-31.
Here's another para that's short and sweet. Some date 2016-12-31.
Para with dup dup words words to test test. Para with dup dup words words to test test.
Para with dup dup words words to test test. Para with dup dup words words to test test.
Para with dup dup words words to test test.
```

Scripting with regexes

- Many languages provide regex modules
- Perform batch processing
- Easily repeat complex processing
- Perl and JavaScript are common

JavaScript

- Processing of HTML forms or other data
- search() - returns the position of the match (-1 if none)

```
var str = "Welcome to STC Summit";
var pos = str.search(/STC/i);
```

- replace() - returns the new value

```
var ret = str.replace(/STC/ig,"The");
```

ExtendScript

- Scripting language in FrameMaker and RoboHelp
- Strip the full path and file name down to just the "name" (strips the ".fm")

```
var doc = app.ActiveDoc;
var filename = doc.Name.replace
  (/^.*?([^\\]+)\.fm$/i, "$1");
```

Perl

- Tightly integrated into language
- Great for quick batch processing scripts
- Platform independent
- Find: `if ($str =~ m/\bcat\b/i) { … }`
- Replace: `$str =~ s/\bcat\b/dog/g;`

Wrap Up

- Brief dip into regex pool
- Regexes aren’t just for geeks
- Start simple and work up as needed
- Simplify your tasks through automation
- Don’t forget the quick reference card!

Resources

- RexEgg – www.rexegg.com
- Regular-Expressions.info – www.regular-expressions.info
- Mastering Regular Expressions – O’Reilly

- Scott Prentice <scott AT leximation.com>