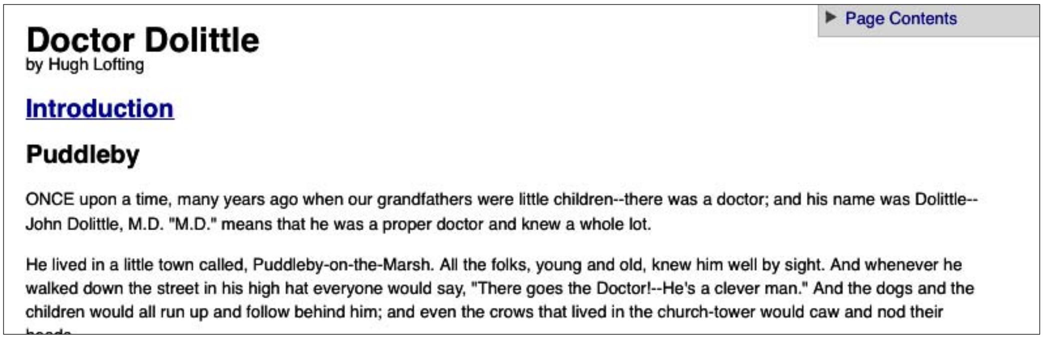
1
Anatomy of a jQuery Script

A typical jQuery script uses a wide assortment of the methods that the library offers. Selectors, DOM manipulation, event handling, and so forth come into play as required by the task at hand. In order to make the best use of jQuery, it's good to keep in mind the breadth of capabilities it provides.

This book will itemize every method and function found in the jQuery library. As there are so many to sort through, it will be useful to know what the basic categories of methods are and how they come to play within a jQuery script. Here we will see a fully functioning script, and examine how the different aspects of jQuery are utilized in each part of it.

A dynamic table of contents

As an example of jQuery in action, we'll build a small script that dynamically extracts the headings from an HTML document and assembles them into a table of contents for the page. Our table of contents will be nestled on the top-right corner of the page as shown in the following screenshot:



Doctor Dolittle
by Hugh Lofting

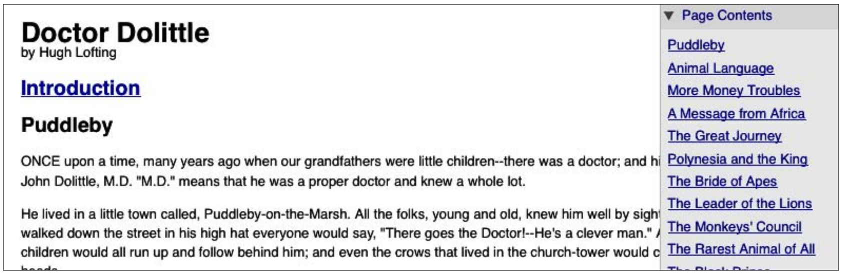
Introduction

Puddleby

ONCE upon a time, many years ago when our grandfathers were little children--there was a doctor; and his name was Dolittle--John Dolittle, M.D. "M.D." means that he was a proper doctor and knew a whole lot.

He lived in a little town called, Puddleby-on-the-Marsh. All the folks, young and old, knew him well by sight. And whenever he walked down the street in his high hat everyone would say, "There goes the Doctor!--He's a clever man." And the dogs and the children would all run up and follow behind him; and even the crows that lived in the church-tower would caw and nod their

We'll have it collapsed initially as shown, but a click will expand it to full height.



Doctor Dolittle
by Hugh Lofting

Introduction

Puddleby

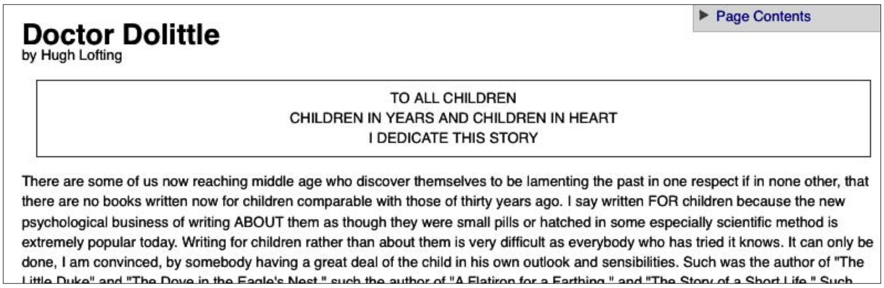
ONCE upon a time, many years ago when our grandfathers were little children--there was a doctor; and hi
John Dolittle, M.D. "M.D." means that he was a proper doctor and knew a whole lot.

He lived in a little town called, Puddleby-on-the-Marsh. All the folks, young and old, knew him well by sigh
walked down the street in his high hat everyone would say, "There goes the Doctor!--He's a clever man."
children would all run up and follow behind him; and even the crows that lived in the church-tower would c

▼ Page Contents

- Puddleby
- Animal Language
- More Money Troubles
- A Message from Africa
- The Great Journey
- Polynesia and the King
- The Bride of Apes
- The Leader of the Lions
- The Monkeys' Council
- The Rarest Animal of All

At the same time, we'll add a feature to the main body text. The introduction of the text on the page will not be loaded initially, but when the user clicks on **Introduction**, the intro text will be inserted in place from another file.



▶ Page Contents

Doctor Dolittle
by Hugh Lofting

TO ALL CHILDREN
CHILDREN IN YEARS AND CHILDREN IN HEART
I DEDICATE THIS STORY

There are some of us now reaching middle age who discover themselves to be lamenting the past in one respect if in none other, that there are no books written now for children comparable with those of thirty years ago. I say written FOR children because the new psychological business of writing ABOUT them as though they were small pills or hatched in some especially scientific method is extremely popular today. Writing for children rather than about them is very difficult as everybody who has tried it knows. It can only be done, I am convinced, by somebody having a great deal of the child in his own outlook and sensibilities. Such was the author of "The

Before we reveal the script that performs these tasks, we should walk through the environment in which the script resides.

Obtaining jQuery

The official jQuery web site (`http://jquery.com/`) is always the most up-to-date resource for code and news related to the library. To get started, we need a copy of jQuery, which can be downloaded right from the front page of the site. Several versions of jQuery may be available at any given moment; the most appropriate for us will be the latest uncompressed version. As of the writing of this book, the latest version of jQuery is 1.4.

No installation is required. To use jQuery, we just need to place it on our site in a web-accessible location. As JavaScript is an interpreted language, there is no compilation or build phase to worry about. Whenever we need a page to have jQuery available, we will simply refer to the file's location from the HTML document with a `<script>` tag as follows:

```
<script src="jquery.js" type="text/javascript"></script>
```

Setting up the HTML document

There are three pieces to most examples of jQuery usage—the HTML document itself, CSS files to style it, and JavaScript files to act on it. For this example, we'll use a page containing the text of a book:

```
<!DOCTYPE html>
<html lang="en">
  <head>
     <meta http-equiv="Content-Type" content="text/html; charset=utf-
      8">
     <title>Doctor Dolittle</title>
    <link rel="stylesheet" href="dolittle.css" type="text/css"
                                          media="screen" />
     <script src="jquery.js" type="text/javascript"></script>
     <script src="dolittle.js" type="text/javascript"></script>
  </head>
  <body>
    <div class="container">
      <h1>Doctor Dolittle</h1>
      <div class="author">by Hugh Lofting</div>
      <div id="introduction">
        <h2><a href="introduction.html">Introduction</a></h2>
      </div>
      <div class="content">
        <h2>Puddleby</h2>
        <p>ONCE upon a time, many years ago when our
           grandfathers were little children--there was a
           doctor; and his name was Dolittle-- John Dolittle,
           M.D.  &quot;M.D.&quot; means that he was a proper
           doctor and knew a whole lot. </p>

         <!-- More text follows... -->
      </div>
    </div>
  </body>
</html>
```

The actual layout of files on the server does not matter. References from one file to another just need to be adjusted to match the organization we choose. In most examples in this book, we will use relative paths to reference files (`../images/foo.png`) rather than root-relative path (`/images/foo.png`). This will allow the code to run locally without the need for a web server.

Immediately following the standard `<head>` elements, the stylesheet is loaded. Here are the portions of the stylesheet that affect our dynamic elements:

```
/** =page contents
************************************************************/
#page-contents {
  position: absolute;
  text-align: left;
  top: 0;
  right: 0;
  width: 15em;
  border: 1px solid #ccc;
  border-top-width: 0;
  background-color: #e3e3e3;
}
#page-contents a {
  display: block;
  margin: .25em 0;
}
#page-contents a.toggler {
  padding-left: 20px;
  background: url(arrow-right.gif) no-repeat 0 0;
  text-decoration: none;
}
#page-contents a.arrow-down {
  background-image: url(arrow-down.gif);
}
#page-contents div {
  padding: .25em .5em .5em;
  display: none;
  background-color: #efefef;
}

/** =introduction
************************************************************/

.dedication {
  margin: 1em;
  text-align: center;
  border: 1px solid #555;
  padding: .5em;
}
```

After the stylesheet is referenced, the JavaScript files are included. It is important that the script tag for the jQuery library be placed *before* the tag for our custom scripts; otherwise, the jQuery framework will not be available when our code attempts to reference it.

To enable faster rendering of visual elements on the page, some developers prefer to include JavaScript files at the end of the document just before the closing `</body>` tag, so that the JavaScript file is not requested until the majority of the document has been loaded. For more information about this perceived performance boost, see `http://developer.yahoo.com/performance/rules.html#js_bottom`.

Writing the jQuery code

Our custom code will go in the second, currently empty, JavaScript file that we included from the HTML using `<script src="dolittle.js" type="text/javascript"></script>`. Despite how much it accomplishes, the script is fairly short.

```
jQuery.fn.toggleNext = function() {
  this.toggleClass('arrow-down')
    .next().slideToggle('fast');
  return this;
};

$(document).ready(function() {
  $('<div id="page-contents"></div>')
    .prepend('<a class="toggler" href="#">Page Contents</a>')
    .append('<div></div>')
    .prependTo('body');

  $('.content h2').each(function(index) {
    var $chapterTitle = $(this);
    var chapterId = 'chapter-' + (index + 1);
    $chapterTitle.attr('id', chapterId);
    $('<a></a>').text($chapterTitle.text())
      .attr({
        'title': 'Jump to ' + $chapterTitle.text(),
        'href': '#' + chapterId
      })
      .appendTo('#page-contents div');
  });
```

```
    $('#page-contents > a.toggler').click(function() {
      $(this).toggleNext();
      return false;
    });

    $('#introduction > h2 a').click(function() {
      $('#introduction').load(this.href);
      return false;
    });
  });
```

We now have a dynamic table of contents that brings users to the relevant portion of the text, and an introduction that is loaded on demand.

Script dissection

This script has been chosen specifically because it illustrates the widespread capabilities of the jQuery library. Now that we've seen the code as a whole, we can identify the categories of methods used therein.

[We will not discuss the operation of this script in much detail here, but a similar script is presented as a tutorial on the Learning jQuery blog: `http://www.learningjquery.com/2007/06/automatic-page-contents`.]

Selector expressions

Before we can act on an HTML document, we need to locate the relevant portions. In our script, we sometimes use a simple approach to find an element as follows:

```
$('#introduction')
```

This expression creates a new jQuery object that references the element with the ID `introduction`. On the other hand, sometimes we require a more intricate selector.

```
$('#introduction > h2 a')
```

Here we produce a jQuery object referring to potentially many elements. With this expression, elements are included if they are anchor tags that are descendants of `<h2>` elements, which are themselves children of an element with the ID `introduction`.

These **selector expressions** can be as simple or as complex as we need. Chapter 2, *Selector Expressions*, will enumerate all of the selectors available to us and how they can be combined.

DOM traversal methods

Sometimes we have a jQuery object that references a set of **Document Object Model (DOM)** elements already, but we need to perform an action on a different, related set of elements. In these cases, **DOM traversal** methods are useful. We can see this in part of our script:

```
this.toggleClass('arrow-down')
  .next()
  .slideToggle('fast');
```

Because of the context of this piece of code, the keyword `this` refers to a jQuery object (it often refers to a DOM element, instead). In our case, this jQuery object is in turn pointing to the `toggler` link of the table of contents. The `.toggleClass()` method call manipulates this element. However, the subsequent `.next()` operation changes the element we are working with, so that the following `.slideToggle()` call acts on the `<div>` containing the table of contents rather than its clicked link. The methods that allow us to freely move about the DOM tree like this are listed in Chapter 3, *DOM Traversal Methods*.

DOM manipulation methods

Finding elements is not enough; we want to be able to change them as well. Such changes can be as straightforward as changing a single attribute.

```
$chapterTitle.attr('id', chapterId);
```

Here we modify the ID of the matched element on the fly.

Sometimes the changes are further-reaching:

```
$('<div id="page-contents"></div>')
  .prepend('<a class="toggler" href="#">Page Contents</a>')
  .append('<div></div>')
  .prependTo('body');
```

This part of the script illustrates that the **DOM manipulation** methods can not only alter elements in place but also remove, shuffle, and insert them. These lines add a new link at the beginning of `<div id="page-contents">`, insert another `<div>` container at the end of it, and place the whole thing at the beginning of the document body. Chapter 4, *DOM Manipulation Methods*, will detail these and many more ways to modify the DOM tree.

Event methods

Even when we can modify the page at will, our pages will sit in place, unresponsive. We need **event methods** to react to user input, making our changes at the appropriate time.

```
$('#introduction > h2 a').click(function() {
  $('#introduction').load(this.href);
  return false;
});
```

In this snippet we register a handler that will execute each time the selected link is clicked. The click event is one of the most common ones observed, but there are many others; the jQuery methods that interact with them are discussed in Chapter 5, *Event Methods*.

Chapter 5 also discusses a very special event method, `.ready()`.

```
$(document).ready(function() {
  // ...
});
```

This method allows us to register behavior that will occur immediately when the structure of the DOM is available to our code, even before the images have loaded.

Effect methods

The event methods allow us to react to user input; the **effect methods** let us do so with style. Instead of immediately hiding and showing elements, we can do so with an animation.

```
this.toggleClass('arrow-down')
  .next()
  .slideToggle('fast');
```

This method performs a fast-sliding transition on the element, alternately hiding and showing it with each invocation. The built-in effect methods are described in Chapter 6, *Effect Methods*, as is the way to create new ones.

AJAX methods

Many modern web sites employ techniques to load content when requested without a page refresh; jQuery allows us to accomplish this with ease. The **AJAX methods** initiate these content requests and allow us to monitor their progress.

```
$('#introduction > h2 a').click(function() {
  $('#introduction').load(this.href);
  return false;
});
```

Here the `.load()` method allows us to get another HTML document from the server and insert it in the current document, all with one line of code. This and more sophisticated mechanisms of retrieving information from the server are listed in Chapter 7, *AJAX Methods*.

Miscellaneous methods

Some methods are harder to classify than others. The jQuery library incorporates several **miscellaneous methods** that serve as shorthand for common JavaScript idioms. Even basic tasks like iteration are simplified by jQuery.

```
$('#content h2').each(function(index) {
  // ...
});
```

The `.each()` method seen here steps through the matched elements in turn, performing the enclosed code on all of them. In this case, the method helps us to collect all of the headings on the page so that we can assemble a complete table of contents. More helper functions like this can be found in Chapter 8, *Miscellaneous Methods*.

A number of additional pieces of information are provided by jQuery as properties of its objects. These global and object **properties** are itemized in Chapter 9, *jQuery Properties*.

Plug-in API

We need not confine ourselves to built-in functionality, either. The **plug-in API** that is part of jQuery allows us to augment the capabilities already present with new ones that suit our needs. Even in the small script we've written here, we've found use for a plug-in.

```
jQuery.fn.toggleNext = function() {
  this.toggleClass('arrow-down')
    .next().slideToggle('fast');
  return this;
};
```

This code defines a new `.toggleNext()` jQuery method that slides the following element open and shut. We can now call our new method later when needed.

```
$('#page-contents > a.toggler).click(function() {
  $(this).toggleNext();
  return false;
});
```

Whenever a code could be reused outside the current script, it might do well as a plug-in. Chapter 10, *Plug-in API,* will cover the plug-in API used to build these extensions.

Summary

We've now seen a complete, functional jQuery-powered script. This example, though small, brings a significant amount of interactivity and usability to the page. The script has illustrated the major types of tools offered by jQuery, as well. We've observed how the script finds items in the DOM and changes them as necessary. We've witnessed response to user action, and animation to give feedback to the user after the action. We've even seen how to pull information from the server without a page refresh, and how to teach jQuery brand new tricks in the form of plug-ins.

In the following chapters, we'll be stepping through each function, method, and selector expression in the jQuery library. Each method will be introduced with a summary of its syntax and a list of its parameters and return value. Then we will offer a description, which will provide examples where applicable. For further reading about any method, consult the online resources listed in Appendix A, *Online Resources*. We'll also examine jQuery's plug-in architecture and discuss both how to use plug-ins and how to write our own.