
jQuery Mobile and JSON

Learn how to create mobile web applications powered by jQuery Mobile

Skill Level: Intermediate

[Frank Ableson](#)

Entrepreneur

Navitend

01 Mar 2011

jQuery powers many of the sites on the Internet today, providing dynamic user experience in the browser and helping to make traditional desktop applications increasingly rare. Now that browsers found on the major mobile platforms have caught up to desktop browser functionality, the jQuery team has introduced jQuery Mobile, or JQM. The mission of JQM is to bring a universal experience to all major mobile browsers, enabling rich content across the Internet, regardless of the viewing device. This tutorial examines fundamental design concepts around jQuery Mobile in the context of a sales force automation-oriented application. You create an intuitive and visually appealing mobile web application that interacts with an Internet-hosted website to store and manage sales opportunities.

Section 1. Before you start

Frequently used acronyms

- Ajax: Asynchronous JavaScript + XML
- ASP: Active Server Pages
- CSS: Cascading Stylesheets
- DOM: Document Object Model
- FTP: File Transfer Protocol

- GPL: GNU General Public License
- HTML: HyperText Markup Language
- HTTP: Hypertext Transfer Protocol
- MIT: Massachusetts Institute of Technology
- SQL: Structured Query Language
- UI: User interface
- XML: Extensible Markup Language

To get the most from this tutorial you should be comfortable constructing web applications with HTML, JavaScript, and CSS. Additionally, the server-side code accompanying the mobile web application is written in PHP and MySQL. Familiarity with server-side programming can also help you as you follow along with this tutorial. If you are familiar with other server-side platforms such as ASP classic, ASP.Net, or Java™ Server Pages, you can find the server-side code quite easy to follow. Familiarity with jQuery is not necessary but, of course, will not hurt. In fact, this tutorial is written from the perspective of a mobile programmer looking to explore a new framework rather than from that of a jQuery professional launching into the mobile space. After completing this tutorial, you will have learned how to construct a basic jQuery Mobile application and manage data between the mobile browser and a back-end server. The application demonstrates how to perform basic record operations such as (*insert, update, delete*) on sales-related data. You also witness how universal your mobile web application is as you run it from multiple browsers. Last, you install shortcuts to your mobile web application to the home screen of both iPod and Android devices to demonstrate how to deploy a web application.

About this tutorial

This tutorial introduces the jQuery Mobile (JQM) framework for writing mobile web applications targeted at the mobile industry's leading browsers. JQM is used to provide intuitive and consistent user experience for web-based applications running on mobile devices such as iPhone, iPad, Android, WebOS, BlackBerry Version 6 (Torch, Playbook), and others. The tutorial begins with a high-level look at the JQM project and the relation of JQM to, and dependence on, HTML5. After a brief look at one of the many ways in which you can construct a JQM application and some of the ways in which JQM enhances fundamental web UI elements, the tutorial takes a look at a basic sales force automation requirement.

With the problem in hand, the tutorial maps out a plan to implement a solution for mobile device users, without the need for native mobile development. The completed application is demonstrated so that you can get a feel for where the

tutorial will take you as you follow along and build your own application, step by step. The source files are examined, function by function, as you learn to implement the solution using JQM. The tutorial concludes with a couple of tricks on making the application easy for your users to access on their mobile devices.

Prerequisites

To follow this tutorial, you need the following:

- Text editor—You can use any text editor you prefer. An editor with syntax highlighting might come in handy. Notepad++ is an open source editor that works well.
- JQuery Mobile links—You actually do not need to download anything. The necessary files are available through jQuery's Content Delivery Network (CDN).
- Web browser—This browser is used to view the jQuery Mobile help and documentation.
- WebKit (Safari) or Chrome Browser—These browsers support jQuery Mobile development on the desktop.
- Mobile device—You can use the iPod Touch, Android, BlackBerry Torch, or similarly capable mobile devices.
- PHP and MySQL hosting environment—This is used for the server side of the application

I created the code samples for this tutorial on a MacBook Pro with VMWare Fusion running Windows® 7. I use Notepad++ for editing the files because it has good syntax highlighting plus secure FTP for making interacting with server-based PHP files very straightforward. You can edit files in vi, but who wants to? For working with the MySQL database, I used the phpMyAdmin available with my hosting account. For testing on a physical device, I used a Nexus One running Android 2.2 and an iPod Touch running iOS 4.1.

See [Resources](#) for helpful links; download the [source code for the sample application](#).

Section 2. jQuery Mobile

Begin with a look at JQM and how you can use it to enhance the development experience for mobile web applications. While jQuery Core has a large installed base, JQM is in its infancy and at the time of this writing was still in alpha mode. Look at just what the JQM project hopes to accomplish.

JQM, latest addition to the family

The objective of JQM is to deliver superior JavaScript capability in a unified UI, working across the most popular smart-phone and tablet devices. Like jQuery, JQM is a freely available, open source code base that is hosted directly on the Internet. In fact, the jQuery core library itself is receiving quite a bit of attention as part of the JQM effort to unify and optimize the code base. This attention speaks volumes to just how far mobile browser technology has advanced in a very short time.

As with jQuery core, there is nothing to install on your development computer; you just include the various *.js and *.css files directly into your web pages, and the functionality of JQM is at your fingertips, so to speak.

[Listing 1](#) demonstrates how easy it is to add jQuery Mobile files to an application.

Listing 1. Adding jQuery Mobile to an application

```
<head>
<title>IBM JQuery Tutorial</title>
<link rel="stylesheet" href="http://code.jquery.com/mobile/1.0a1
/jquery.mobile-1.0a1.min.css" />
<script src="http://code.jquery.com/jquery-1.4.3.min.js"></script>
<script src="http://code.jquery.com/mobile/1.0a1/jquery.mobile-1.0a1.min.js">
</script>
<script src="http://jquery.ibm.navitend.com/utils.js"></script>
</head>
```

The code in [Listing 1](#) is actually a sneak peak at the tutorial's sample code and is revisited a little later in this tutorial. Note for now that a single stylesheet is downloaded directly from jQuery's Content Delivery Network (CDN). A CDN is used to distribute often-used files across the Internet, making download speeds as fast as possible. CDNs are often deployed across large infrastructures, which reduce the path these files need to travel to a minimum distance by placing files in strategic locations across the Internet. The jQuery CDN and others like it are often hosted on large, worldwide infrastructures such as those available from amazon.com and other Internet giants.

In addition to the CSS file, the header includes three JavaScript files. The first JavaScript file is a reference to the jQuery core library in *minified* form, followed by a minified form of the JQM library. Finally, an application-specific JavaScript file named *utils.js* is included. *Minified* means that the code is optimized for quick

downloading and parsing. Keep in mind that every page of the application needs to download these files, so keep their size to an absolute minimum (and rely upon caching!) to greatly improve application performance.

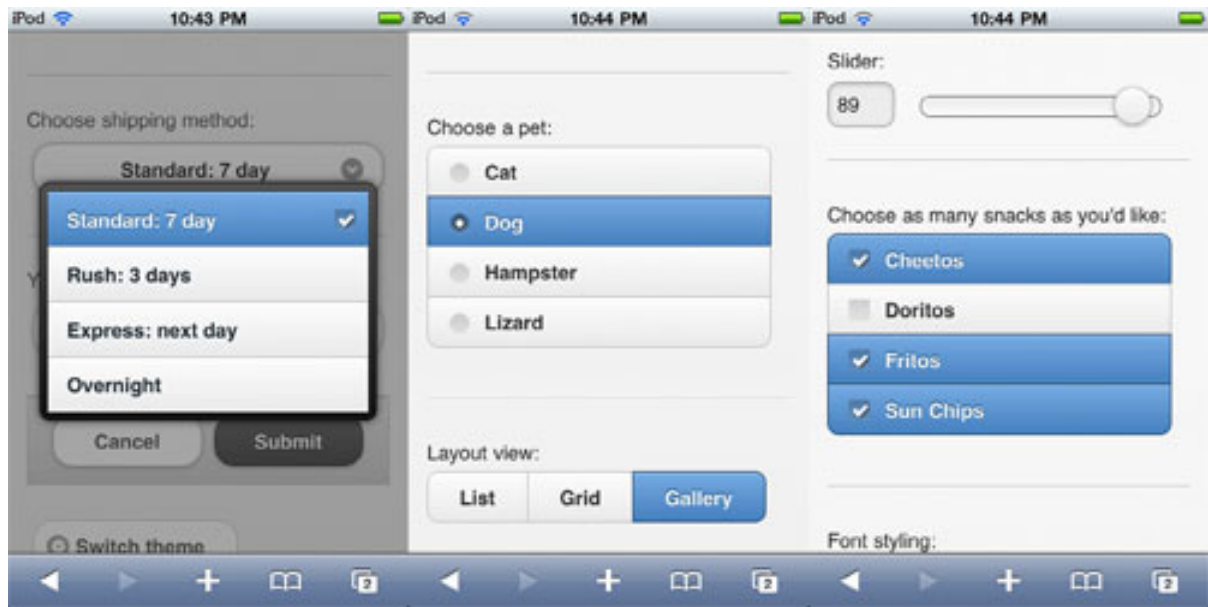
The versions in [Listing 1](#) are actually jQuery Mobile, alpha release 1. Alpha release 2 is available, but due to some buggy behavior, this tutorial relies on the first alpha release of the code. By the time you read this tutorial, a newer version of the JQM files is likely to be available for download. See [Resources](#) for links to jQuery's CDN to obtain the latest version of the library files. The JQM is dually licensed under the MIT and GPL license, which basically means that you are able to use these files in your applications provided that you keep the jQuery attribution.

Note that it is also possible to download a copy of the jQuery files and host them directly from your own web server. This approach is not a bad idea, particularly if you are releasing a commercial application that relies on this framework. Changes to these frameworks can and do occur, sometimes to the detriment of applications that rely on specific behavior. Commercial enterprises are typically more interested in predictable behavior that is easier to support as opposed to always having the latest bells and whistles.

Made for touch

JQM is a touch-optimized framework for constructing a consistent and desirable user experience for mobile browser-based applications. Much of what you already know about writing web applications still applies; however, the objective of making applications look consistent is a real key to the JQM approach. JQM has stylized the standard form elements in such a way to make them both visually attractive and easy to maneuver. Consider the images in [Figure 1](#), which show a subset of the JQM stylized form elements.

Figure 1. JQM form elements



Beyond the easy-to-touch, stylized UI elements, the real magic of JQM takes place in how it manages screen transitions. Let's take a look.

Expanding the DOM

In traditional web application construction, each screen or page is fetched from the server and the entire contents of the screen are replaced. This approach means:

1. A round trip to the server
2. Downloading of the HTML
3. Parsing the HTML
4. Rendering the HTML including the application of the cascading styles

JQM takes a different approach. One of the hallmarks of the jQuery core library is the ease with which developers can perform Ajax calls. Ajax is wrapped around the browser's asynchronous HTTP request capabilities. Ajax is often used to fetch part of a page or to perform some in-line reference data lookup when it is very costly to fetch an entire page. JQM takes Ajax to the next level by intercepting page requests and in most cases converting those requests to specialized Ajax calls. The net result is that as a visitor navigates a web application constructed with JQM, the DOM of the page is manipulated rather than each page being replaced at every turn. To provide some context for this, look at a boilerplate JQM document in [Listing 2](#).

Listing 2. jQuery Mobile page structure

```

<!DOCTYPE html>
<html>
  <head>
    <title>IBM jQuery Tutorial</title>
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.0a1
/jquery.mobile-1.0a1.min.css" />
    <script src="http://code.jquery.com/jquery-1.4.3.min.js"></script>
    <script src="http://code.jquery.com/mobile/1.0a1
/jquery.mobile-1.0a1.min.js"></script>
    <script src="http://jquery.ibm.navitend.com/utils.js"></script>
  </head>
  <div data-role="page">
    <div data-role="header">
      <h1>jQuery Tutorial</h1>
    </div>
    <div data-role="content">
      Page content goes here!
    </div>
    <div data-role="footer">
      Sample code for IBM developerWorks
    </div>
  </div>
</body>
</html>

```

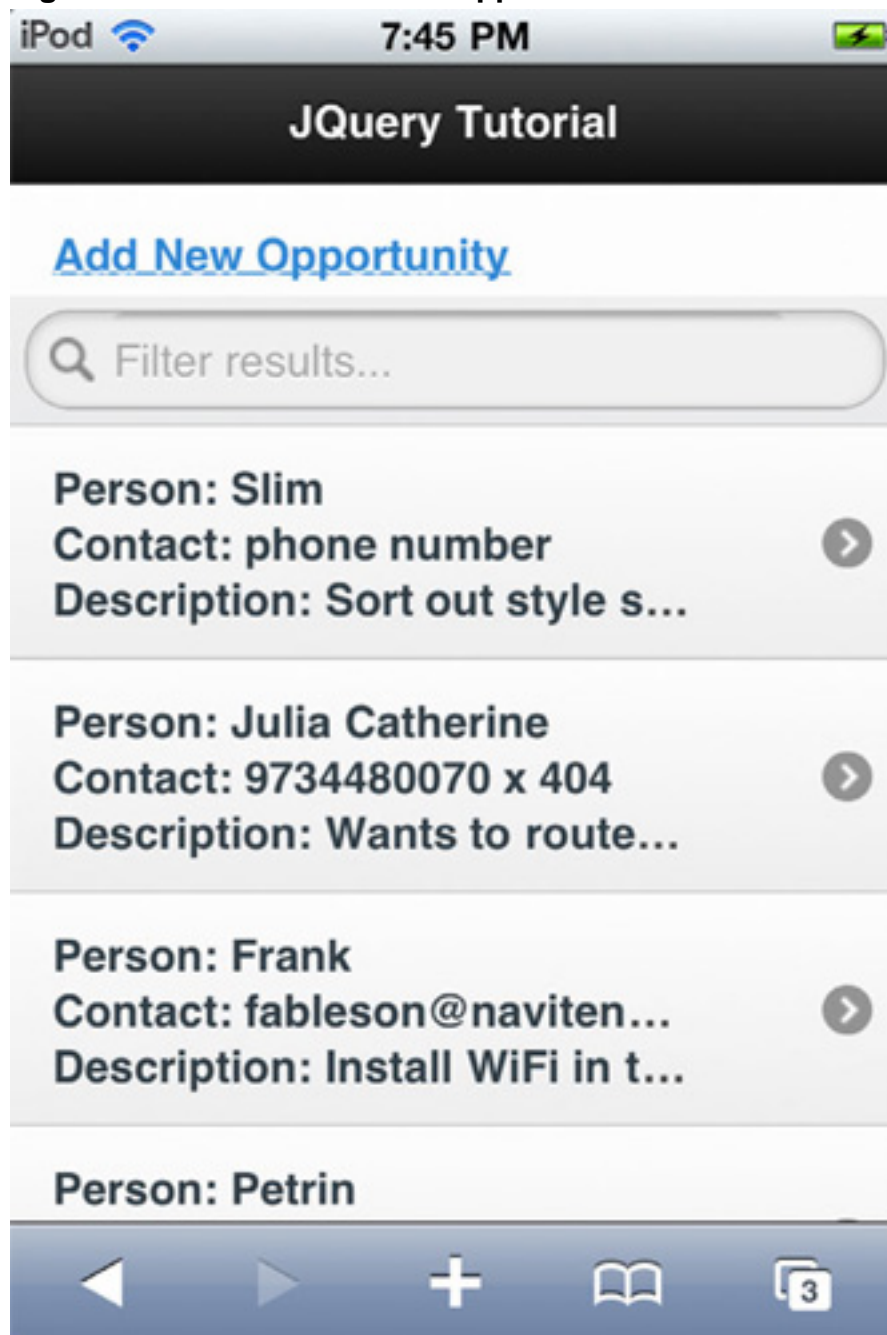
JQM makes ready use of the HTML5 `data-*` attribute. Note in Listing 2 the extensive use of the `div` tag and the `data-role` attribute. In HTML5, any attribute with a prefix of `data-` is essentially ignored by the validating parser, and the application is free to interpret those attributes at will. And that is just what JQM does; in fact, JQM relies on the `data-role` attribute in particular for stringing together its core functionality.

[Table 1](#) shows the four instances of the `data-role` attribute in [Listing 2](#).

Table 1. Four instances of the `data-role` attribute in Listing 2

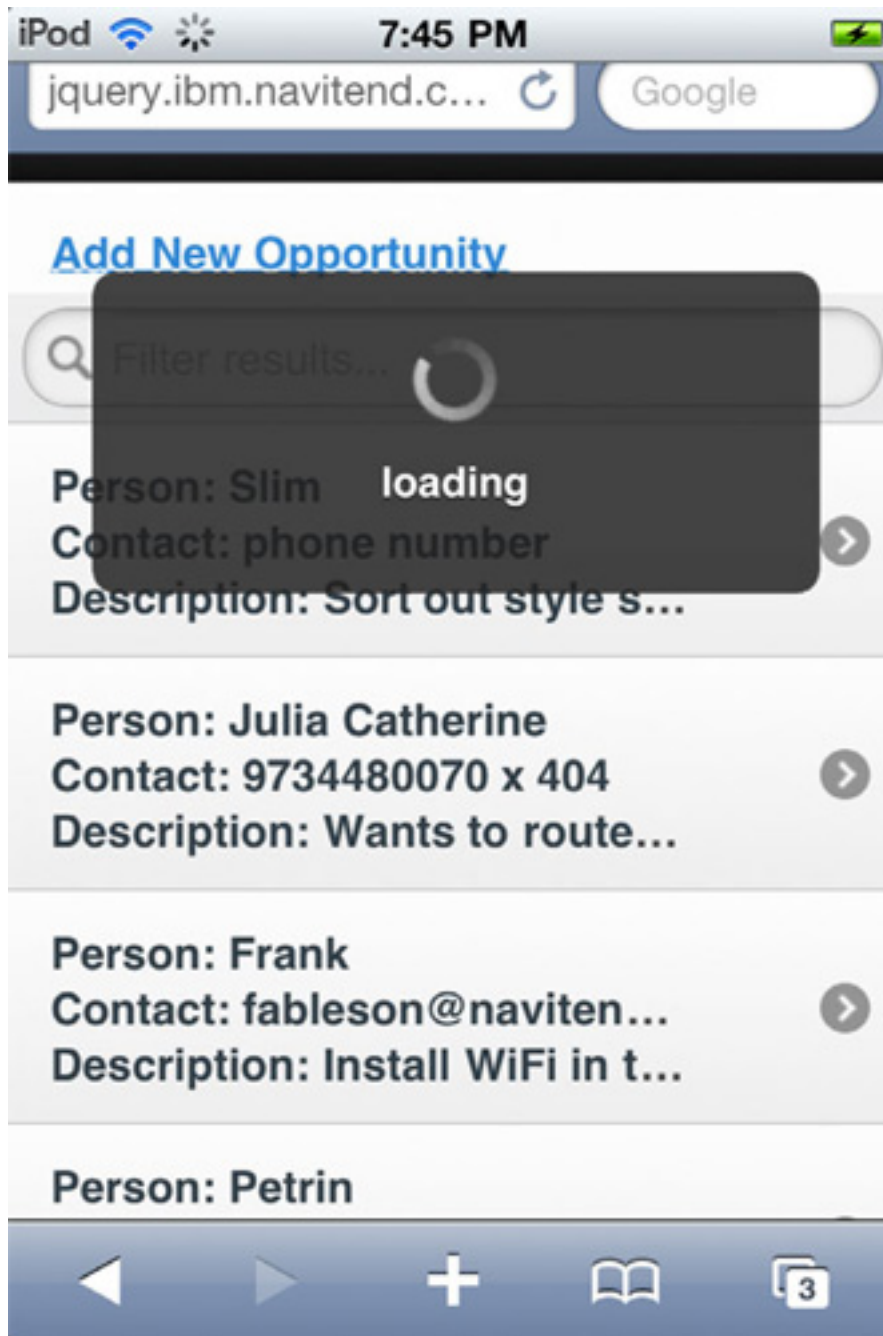
data-role attribute	Comment
page	This is the encapsulating <code>div</code> for an application's page. A single HTML file can contain one or more page-level <code>div</code> elements.
header	Header of the JQM page.
content	Content resides within the content <code>div</code> .
footer	Footer of the JQM page.

When a JQM application switches from one page to the next, the primary activity that takes place is that the `content` `div` is swapped out for the new page's content. More magic takes place than that, but in its most simplistic description, JQM swaps one set of content out for another and selectively modifies the CSS of the DOM to transition from the current page to either a new page, or equally as important, transitions to the prior page in a stack, or hash, of pages. [Figure 2](#) demonstrates the home page of the sample application for this tutorial listing multiple entries, shown running on an iPod.

Figure 2. Home screen of the application

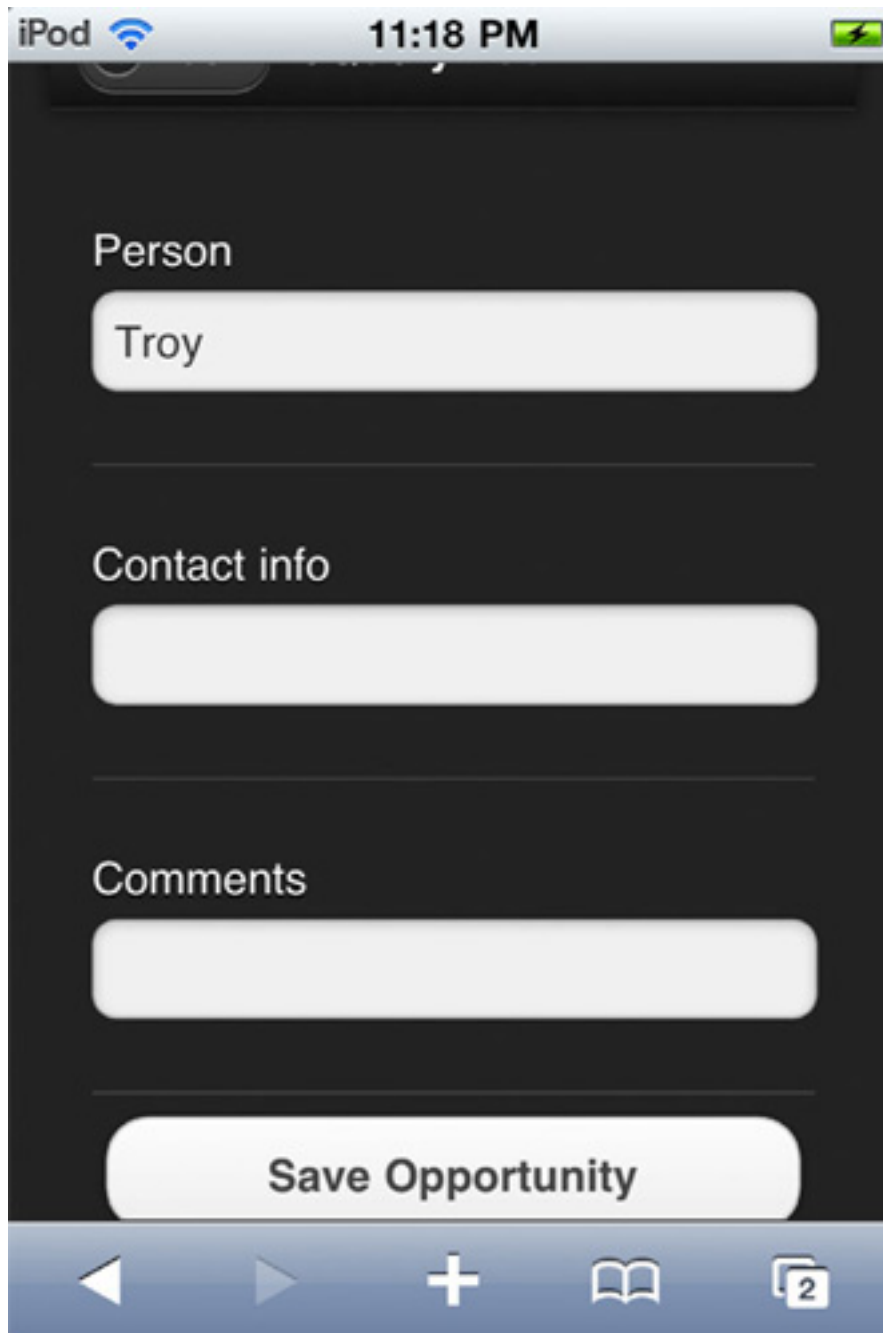
The application presents entries in a list. When an entry is selected, JQM loads a new page and in so doing, presents a loading dialog to the user, so the user can see that something is happening, as in [Figure 3](#).

Figure 3. Loading a new page



The loading status icon spins for a moment and then, when the page content is downloaded or located in the existing DOM, JQM swaps out the currently visible content and presents the new content, as in [Figure 4](#), which demonstrates a dialog window in JQM.

Figure 4. JQM dialog for adding new data



JQM is heavy on themes and the use of color swatches. Note that the color scheme of the dialog in [Figure 4](#) (dark grey with black background) is distinct from the color scheme in [Figure 3](#) (blue and grey with light background). You are encouraged to review the documentation regarding themes found on the jQuery Mobile web page, linked in [Resources](#).

When a page change occurs, you have a number of transitions to choose from, including:

- Slide
- Slide up
- Slide down
- Pop
- Fade
- Flip

When the user navigates back, the transition can reverse itself as well. I particularly like the flip transition. To select a particular transition, use the data-transition attribute as follows: `Go to some page`.

Like jQuery Core, JQM also affords the ability to bind JavaScript events including events such as:

- Device orientation changed
- Page before and after create
- Page before and after show
- Page before and after hide
- Tap
- Tap and hold
- Swipe
- SwipeLeft
- SwipeRight

To wire up one of these events, you can use code such as that in [Listing 3](#).

Listing 3. Notification before a page is shown

```
$( 'body' ).bind( 'pagebeforeshow', function( event, ui ) {  
    alert( 'beforeshow' );  
} );
```

jQuery employs the dollar (\$) symbol—anything following a \$ is jQuery syntax. As another example of wiring up events, suppose you want to do something when a list entry is swiped. [Listing 4](#) demonstrates the jQuery code to capture the `swipeleft` event on each individual list item.

Listing 4. Assigning swipe-handlers for list items

```
$( 'li' ).each(
  function (idx) {
    $(this).bind (
      'swipeleft',
      function(event,ui) {
        alert("swipe left" + $(this).attr('data-ibm-jquery-key'));
      }
    );
  }
);
```

This code can be detailed as follows:

1. Iterate over each `li` element. An `li` element is a list item in an ordered list (`ol`) or an unordered list (`ul`).
2. Each entry is wired up to track the `swipeleft` event.
3. Whenever a `swipeleft` event is captured, a function is called that displays an alert box.
4. You also can leverage the HTML5 `data*` attribute feature. Note in this example that you can gain access to an element's attributes through the `attr` method. Here you fetch the data associated with a custom attribute named: `data-ibm-jquery-key`.

The merits of mixing data and UI elements is debatable; however, the ability to store data directly in a UI element, along with an easily bound function, provides for some creative coding opportunities.

This discussion really only scratches the surface of the capabilities in jQuery Mobile, but it has to suffice for now. Now consider the motivation behind this tutorial's sample application.

Sales force automation (SFA)

While there is no shortage of entertainment applications available in the numerous App Stores, many business applications fall into the category of *sales force automation*. These applications are designed to aid in the process of managing the sales activities of an individual or a team of sales persons. Regardless of the industry, success in sales falls to a simple mantra: "Win the right to proceed to the next step." If you have a decent sales process, you eventually close the sale.

The application created in this tutorial is aimed at the front end of the sales process, sometimes called the *opportunity*. The application you construct in this tutorial allows

the user to manage new opportunities directly from his or her mobile phone. Why? Simple. When business people are out of the office, interacting at conferences or networking events, they often run into people who might have an interest in their products and services. The goal of this application is to give the user a simple means of capturing three important pieces of information:

- **Person**—The name of the counterparty they wish to follow up with
- **Contact info**—A phone number or email address
- **Comments**—A brief note about the opportunity

For example, here is a potential entry:

- Person: Joe Bloggs
- Contact info: jbloggs@someorganizationabc.com
- Comments: Joe needs help with identity management.

Short, sweet, and to the point. When I get back to my office, I can send Joe an email to learn more about his specific pain points or present some information around my company's products and services that is relevant to someone needing assistance with identity management.

This application is, by no means, a complete sales force automation application; however, it is enough of a usage scenario to begin employing some JQM functionality and, if used faithfully, will likely improve the results of any on-the-road salesperson. And, if your sales team has a variety of devices including iPhone, Android, BlackBerry Torch, and WebOS, this application can run on any and all of those devices. Let's see how it is constructed.

Section 3. The application architecture

This section explores the architecture of the tutorial's sample application, progressing through each of the major steps in its construction. You can either follow along, step by step, to reconstruct the application yourself, or you can download the complete project from [Resources](#).

The application architecture

This application, named simply jQuery tutorial, is implemented across two primary

platforms:

- Mobile code, leveraging JQM
- Server-side code, providing data persistence in a MySQL database

All the server-side interaction takes place within PHP files with data managed in a single table named opportunities within a MySQL database.

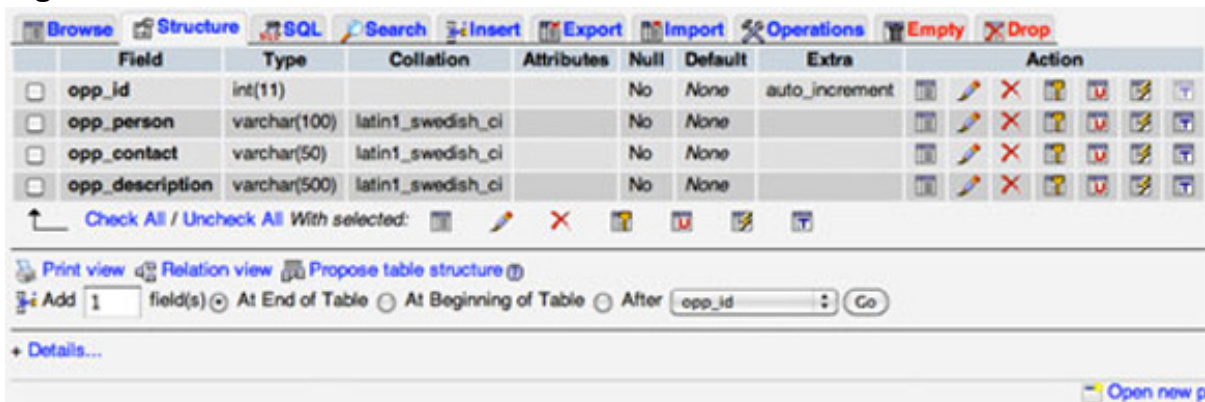
The database structure is very simple, just four fields (see [Table 2](#)).

Table 2. The four fields of the database structure

Column name	Comment
opp_id	Numeric identifier for an entry. This identifier is auto-incremented by the database.
opp_person	Name of person or company.
opp_contact	Contact info for person.
opp_description	Brief description of the opportunity.

[Figure 5](#) shows the structure of the file in phpMyAdmin. Column headings include Field, Type, Collation, Attributes, Null, Default, Extra, Action. (View a [larger version](#) of [Figure 5](#).)

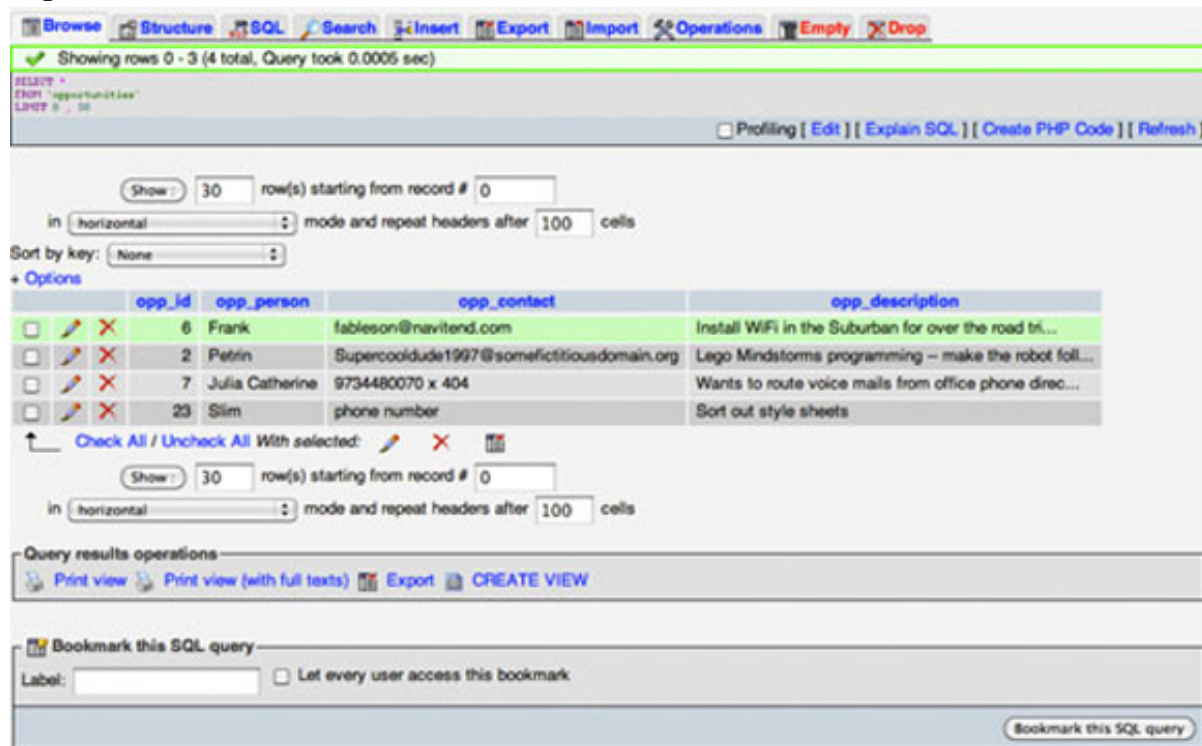
Figure 5. Database table structure



You can see the simple structure and database types. Most of these entries are the defaults. In a production application, you would likely put more thought into your data dictionary.

To get the application off the ground, I primed the database with some data using the insert feature of phpMyAdmin. [Figure 6](#) shows a snapshot of the data that correlates to the device screen images presented in this tutorial. Column headings include opp_id, opp_person, opp_contact, opp_description. (View a [larger version](#) of [Figure 6](#).)

Figure 6. Some initial data records



As you test the application, jumping over to browse the table directly is a helpful sanity check. Take an inventory of the source files used by the application, shown in [Table 3](#).

Table 3. The required application source files

File	Comment
header.php	Includes the head portion of the HTML document, including the required script tags to include the JQM files. The contents of this file was shown earlier as Listing 1 .
footer.php	Includes any HTML footer information. For many applications, this information includes the Google Analytics JavaScript glue to help gather usage statistics on the application.
index.php	Home page of the application user interface, acting as part controller, part view from a quasi-MVC paradigm, the part of the controller in a loosely configured MVC design.
utils.php	All the data access routines are placed within this file.
db.php	Database credentials are stored in this file.
utils.js	A couple of form-level validation scripts are contained in this file.

The best way to see how these files work together is to take a step-by-step walk through each of them, which is covered next.

Constructing the application

The key to most applications is to get the data model right. This application's data model is very straightforward as presented in the prior section. In the code walk-through you can start from the beginning with the database definitions in [Listing 5](#).

Listing 5. opportunities SQL script

```
CREATE TABLE IF NOT EXISTS `opportunities` (  
  `opp_id` int(11) NOT NULL AUTO_INCREMENT,  
  `opp_person` varchar(100) NOT NULL,  
  `opp_contact` varchar(50) NOT NULL,  
  `opp_description` varchar(500) NOT NULL,  
  UNIQUE KEY `opp_id` (`opp_id`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=25 ;
```

If you want to modify the application by including additional fields, you need to add them to the database manually through a tool such as phpMyAdmin, shown previously in [Figure 5](#), or by extending the SQL script. After the database table is in place, it is time to connect to the database. You accomplish that with some mysql functions, shown in db.php in [Listing 6](#).

Listing 6. db.php

```
<?  
$mysql_db = "databasename";  
$mysql_user = "username";  
$mysql_pass = "password";  
$mysql_link = mysql_connect("localhost", $mysql_user, $mysql_pass);  
mysql_select_db($mysql_db, $mysql_link);  

```

Obviously, you need to substitute the database name, username, and password for your own environment. If you don't have access to a MySQL- and PHP-enabled hosting account and want to follow along, now is a good time to get your environment configured. See [Resources](#) for some available options.

With the database in place, it is time to switch to a top-down approach by looking at the primary source file for this application, index.php, shown in [Listing 7](#).

Listing 7. Main UI source file

```
<?  
require('db.php');
```

```
require('utils.php');
require('header.php');
?>
  <div data-role="page">
    <div data-role="header">
      <h1>jQuery Tutorial</h1>
    </div>
    <div data-role="content">

<?
$action = $_REQUEST['action'];
if ($action == 'addnew') {
    showOneOpp(-1);
} else if ($action == 'upsert') {
    if ($_REQUEST['id'] == '-1') {
        addOpp($_REQUEST['person'], $_REQUEST['contact'], $_REQUEST['description']);
    } else {
        updateOpp($_REQUEST['id'], $_REQUEST['person'], $_REQUEST['contact'],
$_REQUEST['description']);
    }
    showOpps();
} else if ($action == 'delete') {
    killOpp($_REQUEST['id']);
    showOpps();
} else if ($action == 'details') {
    showOneOpp($_REQUEST['id']);
} else {
    showOpps();
}
?>
  </div>
  <div data-role="footer">
    Sample code for IBM developerWorks
  </div>
</div>
<? require('footer.php'); ?>
</body>
</html>
```

This PHP file is the entry point for all interactions on the server. Depending on the presence and value of a parameter named `action`, the script performs different functionality. Before diving into the specific operations, take note of the jQuery Mobile structure found in this document including the multiple `div` elements, each with the respective `data-role` for page, header, content, and footer.

This application architecture is rather simplistic—each time the page is loaded, the content is replaced with newly generated content in the `content` `div`. To some degree, that might be cheating and returns to some older web application habits. Perhaps, but the goal of this tutorial is to present some basic functionality of JQM in a useful real-world scenario, and keeping this simple structure helps you to accomplish all those objectives.

To understand what is happening, take a look in more detail at `index.php`, from the top to bottom.

1. The `db.php` file is included—This file gives you access to the database.

2. The `utils.php` file is included—This file provides all the data management functions specific to this application.
3. The `header.php` file is included—This file incorporates the jQuery Core and jQuery Mobile JavaScript files, the jQuery Mobile CSS file along with an application-specific JavaScript file named `utils.js`.
4. The JQM div elements are defined, and the header includes a single `h1` tag with accompanying heading text.
5. The `$action` variable is set up by extracting it from the `$_REQUEST` built-in array. The `$_REQUEST` variable coalesces the `$_GET` and the `$_POST` data, simplifying the processing of a variety of requests sent to this page.
6. The `$action` variable is evaluated with the following options, each one calling one or more functions implemented in `utils.php`:
 - a. `addnew`—Display an empty form for adding a new entry. When you are at the trade show and meet a new prospect, this is the feature for you.
 - b. `upsert`—If the record is new, you want to insert it into the table. If the record already exists, you need to update its columns. If the `id` field is equal to `-1`, that tells you that you have a brand new record and must perform the insert. Any other value is interpreted as a valid record, or opportunity, identifier.
 - c. `delete`—The user has requested to delete this record.
 - d. `details`—The user has selected an entry and wants to view the details of the opportunity.
 - e. In the event that the `$action` variable is empty, you simply display the list of opportunities. This is the default scenario when the page first loads.
7. The page wraps up by including `footer.php`. In this application, the footer includes some Google analytics code to help track the usage of the application.

With this page behind you, it is time to look at the functionality contained in `utils.php`. In particular, you want to observe the `showOpps` function as it generates the list of opportunities for the user interface and with it a new JQM piece of functionality, the listview, introduced in [Listing 8](#).

Listing 8. The listViewshowOpps generates a listview

```
function showOpps()
{
  global $mysql_link;

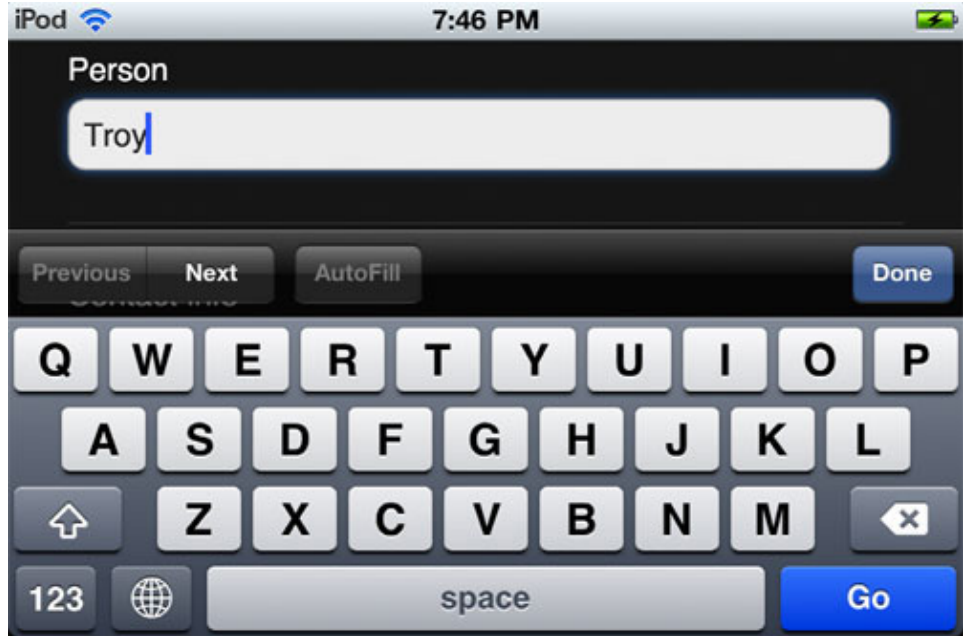
  $COL_OPPID= 0;
  $COL_PERSON= 1;
  $COL_CONTACT= 2;
  $COL_DESCRIPTION= 3;
  $sql = "select * from opportunities order by opp_id desc";
  $result = mysql_query($sql,$mysql_link);

  if(mysql_num_rows($result))
  {
    print("<a data-rel=\"dialog\" data-transition=\"pop\"
href=\"index.php?action=addnew\">Add New Opportunity</a>
<br/><br/>");
    print("<ul data-role=\"listview\" data-filter=\"true\">");
    while($row = mysql_fetch_row($result)) {
      print("<li data-ibm-jquery-contact=\"\".$row[$COL_CONTACT].\">");
      print("<a data-rel=\"dialog\" data-transition=\"pop\"
href=\"index.php?action=details&id=\".$row[$COL_OPPID].\">");
      print("Person:&nbsp;\".$row[$COL_PERSON]. "<br/>");
      print("Contact:&nbsp;\".$row[$COL_CONTACT]. "<br/>");
      print("Description:&nbsp;\".$row[$COL_DESCRIPTION]);
      print("</a>");

      print("</li>\n");
    }
    print("</ul>");
  }
}
```

The showOpps function jumps out to the database, fetches all the rows, showing the newest record first and organizing the data into a listview. Note the JQM features in this code listing.

1. Prior to the fetching of the rows, an anchor tag is generated for Add New Opportunity. Two JQM-specific features are included here:
 - a. The data-rel="dialog" tells JQM how the new window should appear when it is shown. It gets the color scheme of a dialog box.
 - b. The data-transition="pop" tells JQM to make the box pop up as it comes into view. When the box is cleared, it performs a reverse transition where the dialog box shrinks down until it is no longer visible. As you work through the application, try changing the value to flip, fade, or one of the other available transitions. [Figure 7](#) shows the dialog screen for adding a new entry. This time, it is in portrait mode on an iPod.
- Figure 7. Adding a new opportunity**



- When the list is created, it is created as an unordered list or `ul` element. The `data-role` for this element is a listview. This is an important attribute for JQM as list management is a large topic of interest. Also, note the `data-filter="true"` attribute. That simple attribute provides arguably the most valuable function in the entire application—look-ahead searching on the main screen, as in [Figure 8](#).

Figure 8. Filtered results



When the user types in a phrase, each entry in the list is checked to see if the phrase is found; if not, the entry is dropped from the list, leaving only the matching entries. In [Figure 8](#) the word "Lego" was found in only one entry. As an interesting side note: The image in [Figure 8](#) is taken from WebKit (Safari) running on my MacBook. Both Safari and Chrome are good desktop browsers to test mobile-targeted web applications.

- When the user selects an entry in the link they are actually activating an

anchor with a `data-rel` of `dialog` and a `data-transition` of `pop`. The result is the details associated with a particular entry shown in a dialog view, as in [Figure 9](#), also taken from the desktop. The entry shows the Person, Contact info, and Comments fields plus the Save Opportunity button.

Figure 9. Details of an opportunity record



Notice the similar look and feel regardless of the device from which the screen image is captured. This tutorial runs equally well on an Android device, an iPod, and Safari (WebKit nightly build). You are seeing some of the design goals of jQuery Mobile coming to fruition.

4. A final item to observe in this code listing is the attribute listed with the list item. In this case, each list item includes a custom attribute named `data-ibm-jquery-contact` with a value taken from the `opp_contact` field in the database. This item is in place for future functionality for adding the ability to call or email when the user performs a taphold operation on an entry in the list.

After an existing opportunity record is displayed in the dialog, as in [Figure 9](#), the user has a couple of options available. The code for this page is provided in the `showOneOpp` function, found in `utils.php` and presented in [Listing 9](#).

Listing 9. showOneOpp

```
function showOneOpp($id)
{
    global $mysql_link;
```

```

$COL_OPPID= 0;
$COL_PERSON= 1;
$COL_CONTACT= 2;
$COL_DESCRIPTION= 3;

$person = "";
$contact = "";
$description = "";

    if ($id != -1) {
        $sql ="select * from opportunities where opp_id = " . $id;
        $result = mysql_query($sql,$mysql_link);

        if(mysql_num_rows($result)) {
            $row = mysql_fetch_row($result);
            $person = $row[$COL_PERSON];
            $contact = $row[$COL_CONTACT];
            $description = $row[$COL_DESCRIPTION];
        }
        print("<a rel=\"external\" href=\"javascript:deleteEntry($id)
\>Delete this entry</a>");
    }

    print("<form method=\"post\" rel=\"external\" action=\"index.php\"
onsubmit=\"return checkForm();\>");
    print("<input type=\"hidden\" name=\"action\" value=\"upsert\"/>");
    print("<input type=\"hidden\" name=\"id\" value=\"\$id\"/>");
    print("<fieldset>");

    print("<div data-role=\"fieldcontain\">");
    print("<label for=\"person\">Person</label>");
    print("<input type=\"text\" name=\"person\" maxlength=\"100\"
id=\"person\" value=\"\$person\" />");
    print("</div>");

    print("<div data-role=\"fieldcontain\">");
    print("<label for=\"contact\">Contact info</label>");
    print("<input type=\"text\" name=\"contact\" maxlength=\"100\"
id=\"contact\" value=\"\$contact\" />");
    print("</div>");

    print("<div data-role=\"fieldcontain\">");
    print("<label for=\"description\">Comments</label>");
    print("<input type=\"text\" name=\"description\" maxlength=\"100\"
id=\"description\" value=\"\$description\" />");
    print("</div>");

    print("<fieldset>");
    print("<button type=\"submit\" value=\"Save\">Save Opportunity
</button>");

    print("</form>\n");
}

```

The showOneOpp code is a rather brute-force way of populating the screen where you write all of the form elements. Some items to note on this screen include:

1. If the \$id is -1, you prepare the screen for adding a new opportunity record; otherwise, you load the existing opportunity record and initialize some page -level variables, namely \$person, \$contact, and \$description.

2. If you have an existing record, you show a link to allow the user to delete the opportunity.
3. The `data-role="fieldcontain"` surrounding each field helps JQM display the fields by grouping the label and related input HTML element together, separated by thin lines.
4. When the user populates the fields and selects the **Save Opportunity** button, the fields are checked to make sure that they are all populated and if so, the record is saved. [Figure 10](#) shows an alert warning the user to populate all fields.

Figure 10. Form level validation



5. The other option on this screen is to delete an existing entry. In this case, another JavaScript function prompts the user to confirm that they really want to remove the chosen opportunity record, as shown in [Figure 11](#).

Figure 11. Confirming deletion of an opportunity record



Selecting **OK** at this deletion prompt sends the application back to `index.php` with an action of `delete`.

The JavaScript routines for these form-level validations are contained in the `utils.js` file, which is loaded by the `header.php` include file. [Listing 10](#) presents `utils.js`.

Listing 10. Utils.js form-level validation

```
function checkForm() {
    try {
        if ($.trim($('#person').val()) == "" ||
            $.trim($('#contact').val()) == "" ||
            $.trim($('#description').val()) == "") {
            alert("Please enter all fields");
            return false;
        }
    } catch (e) {
        alert(e);
        return false;
    }
    return true;
}

function deleteEntry(id) {
    try {
        var confirmString = "Delete this entry. Are you sure?\n" + $.trim($('#person')
            .val()) + "\n" + $.trim($('#contact').val()) + "\n" + $.trim($('#description').val());
        if (window.confirm(confirmString)) {
            window.location="index.php?action=delete&id=" + id;
        }
    } catch (e) {
        alert(e);
        return false;
    }
    return true;
}
```

```
}
```

These functions are rather straightforward JavaScript, aided by some jQuery selector magic.

You've seen most of the application's functionality. Now look at a couple more functions implemented in `utils.php`. They are shown in [Listing 11](#).

Listing 11. More data management routines

```
function addOpp($person,$contact,$description)
{
    global $mysql_link;

    $sql = "insert opportunities(opp_id,opp_person,opp_contact,opp_description) values
(NULL, '$person', '$contact', '$description')";
    $result = mysql_query($sql,$mysql_link);
    if ($result == 1) {
        return "SUCCESS";
    } else {
        return "FAILED";
    }
}

function updateOpp($id,$person,$contact,$description)
{
    global $mysql_link;

    $sql = "update opportunities set opp_person='". $person. "',opp_contact=
'". $contact. "',opp_description='". $description. "' where opp_id= ".$id;
    $result = mysql_query($sql,$mysql_link);
    if ($result == 1) {
        return "SUCCESS";
    } else {
        return "FAILED";
    }
}

function killOpp($id)
{
    global $mysql_link;

    $sql = "delete from opportunities where opp_id = $id";
    $result = mysql_query($sql,$mysql_link);
}
}
```

Looking at these routines, you can see that they implement straightforward PHP/MySQL data access functionality for inserting, updating, and deleting opportunity records.

Though that rounds out the code for this tutorial's sample application built with jQuery Mobile, it is arguably more of a beginning than an ending as there is much more to be learned about jQuery Mobile. The JQM project is due to make its 1.0

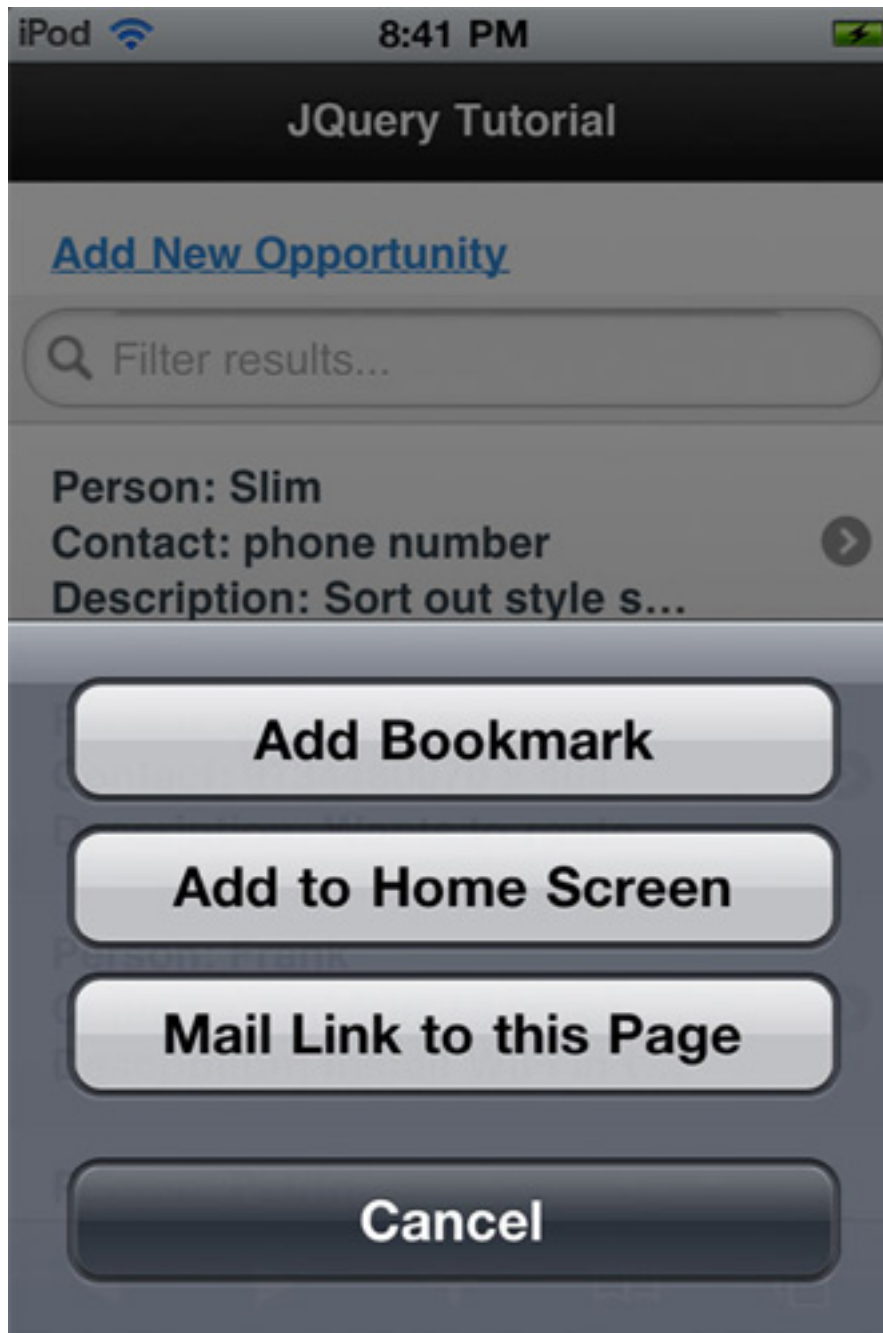
release sometime in early 2011. Over time, look for it to be incorporated into frameworks such as PhoneGap and possibly even alternative development environments such as Appcelerator's Titanium.

To conclude this tutorial, take a look at installing shortcuts to this newly created web application, powered by JQM.

Installing the application

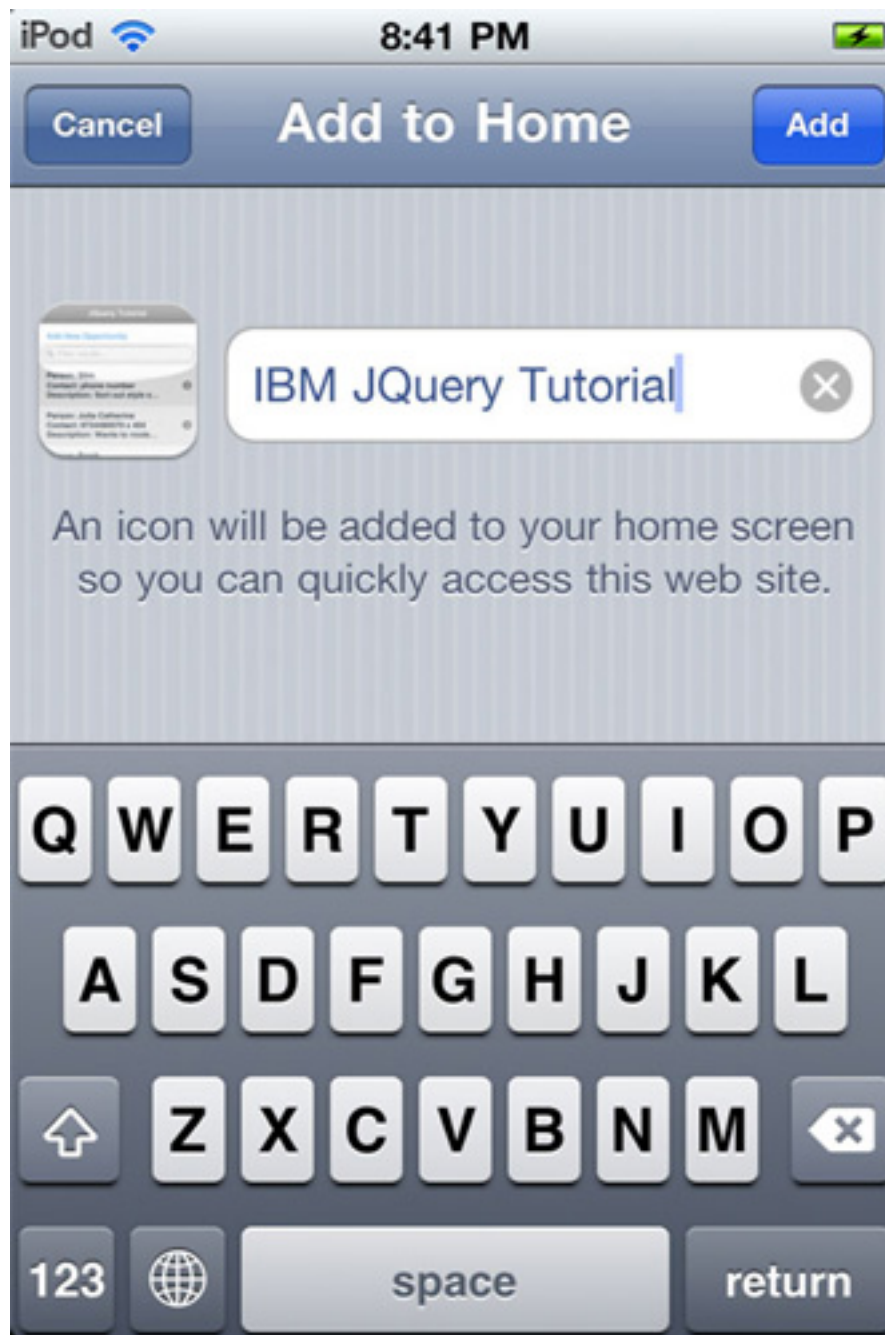
This application is not a native application, so it is not downloadable from a traditional App Store; however, you can create a shortcut to it on your device's home screen. [Figure 12](#) demonstrates creating a short cut on an iPod device.

Figure 12. Select the plus sign (+) at the bottom of your screen



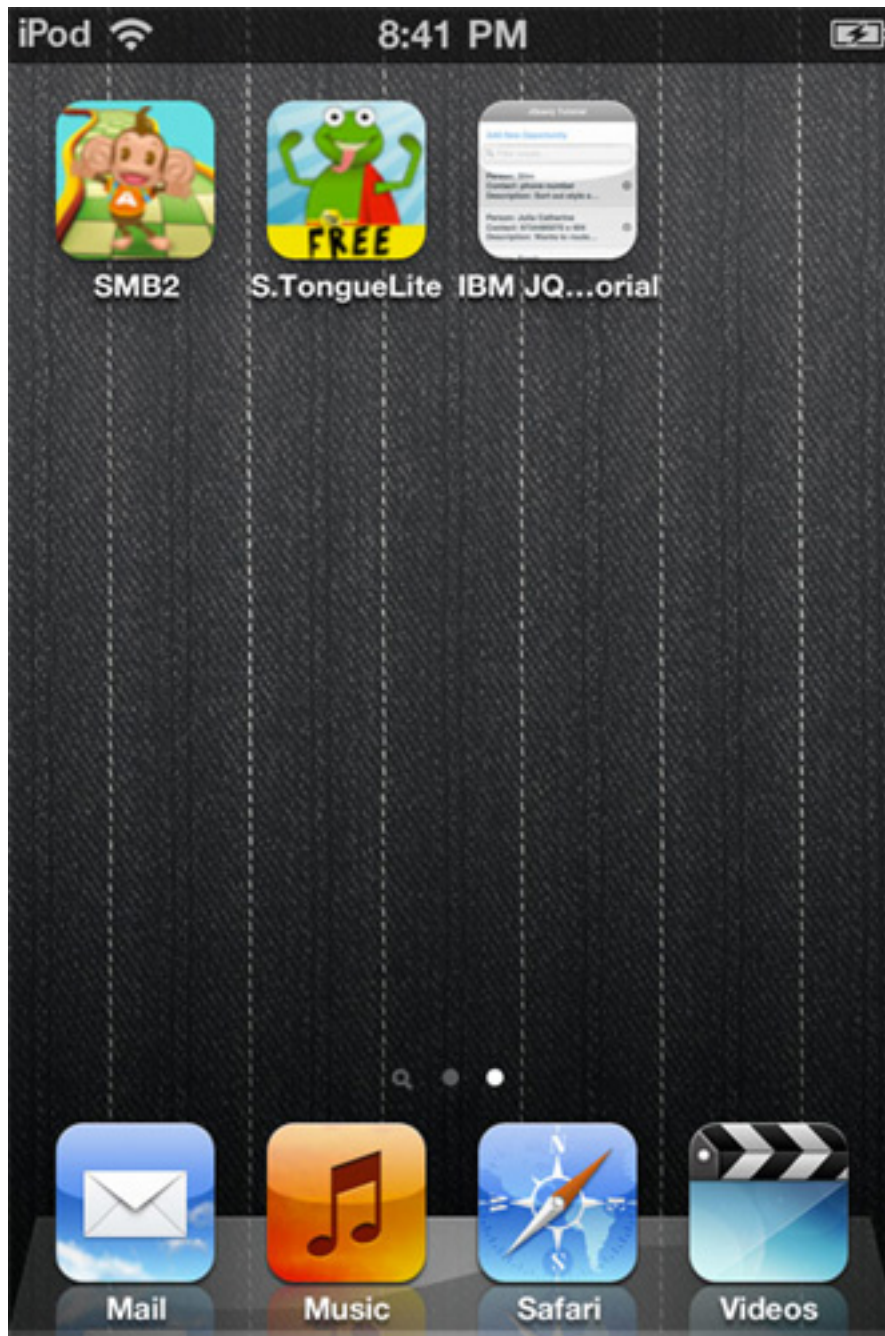
The iPod platform allows you to add a new bookmark, add a link to the home screen, or email the link to a friend. Choose the **Add to Home Screen** option. Next, give the link a name, as in [Figure 13](#).

Figure 13. Choose Add to Home Screen



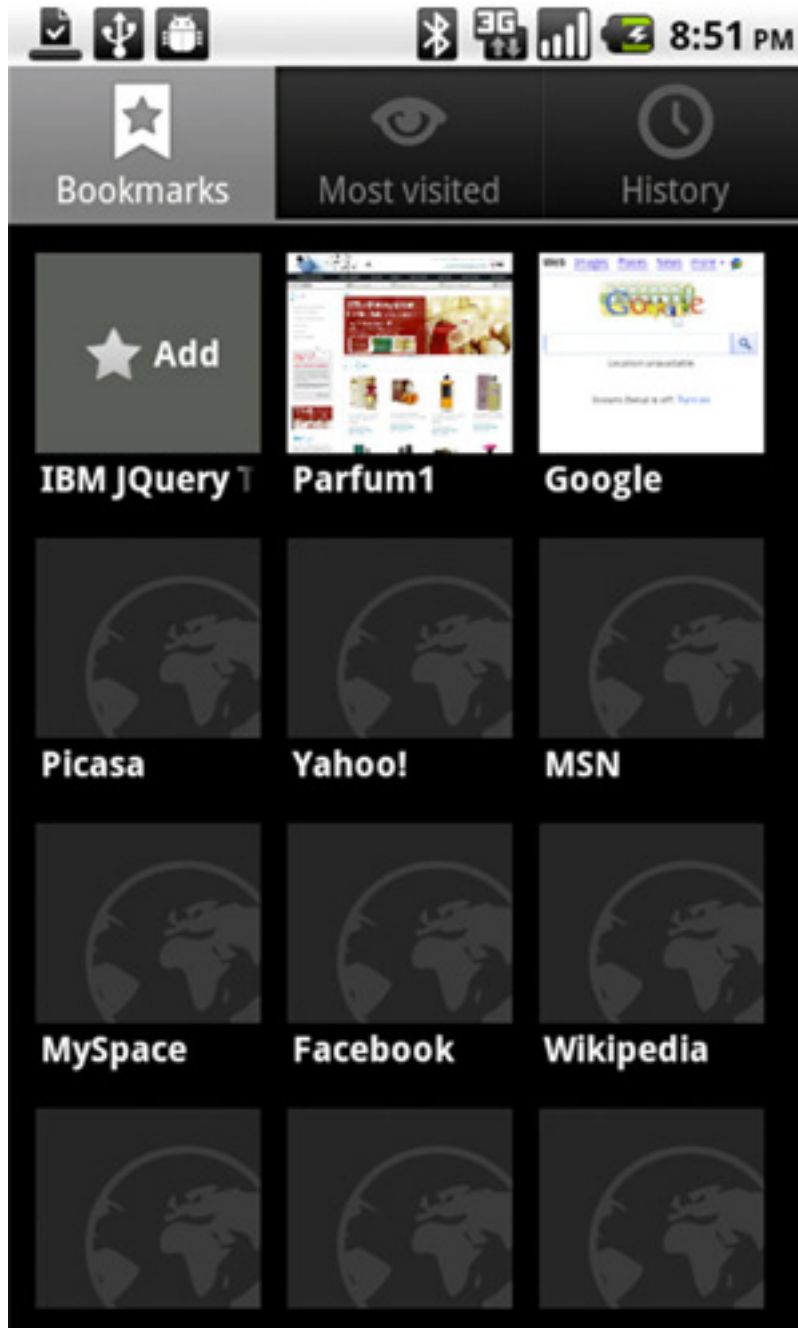
After you provide a name for your shortcut, the shortcut is dropped to your home screen, right next to your favorite games, as in [Figure 14](#).

Figure 14. Shortcut on home screen on iPod platform



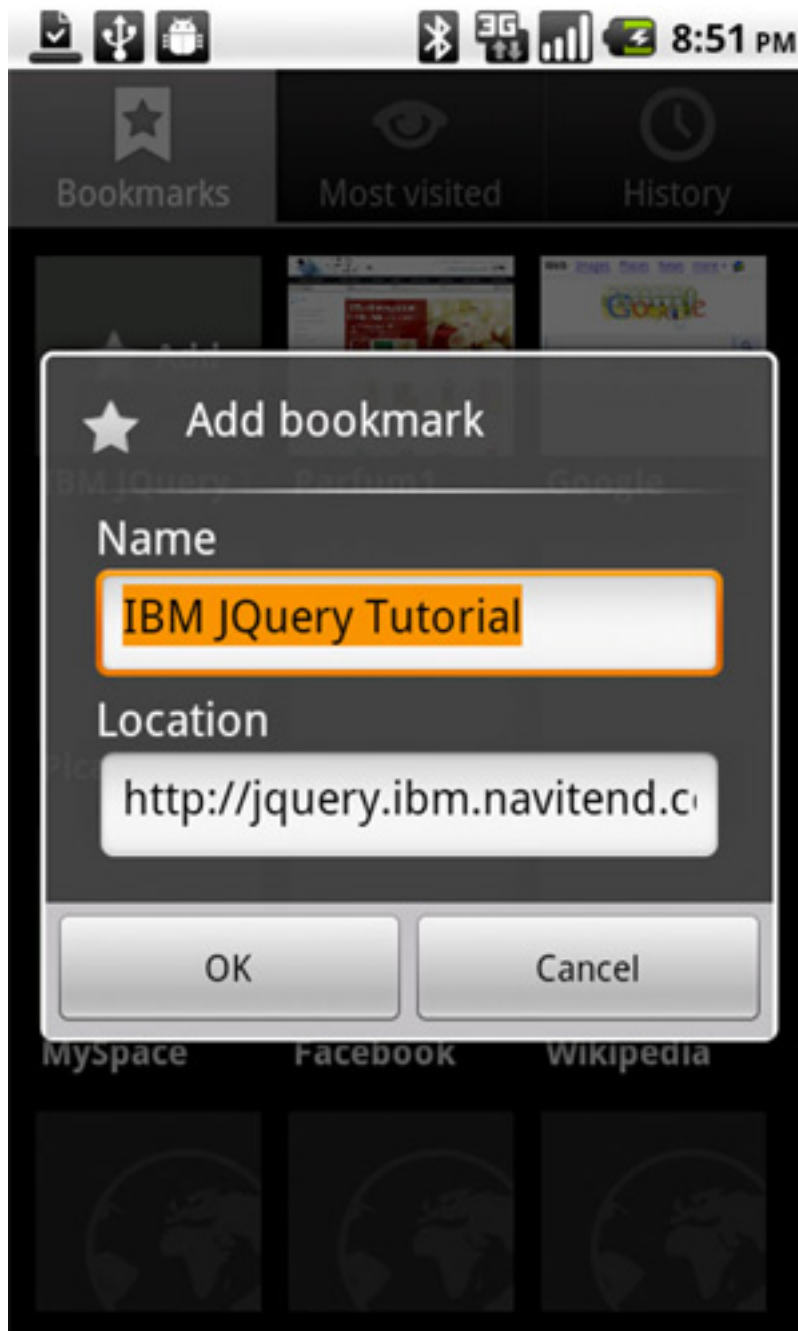
The process to add the shortcut on the Android platform takes a couple more steps, starting with the creation of a new bookmark, as in [Figure 15](#).

Figure 15. Create a new bookmark on Android platform



After the bookmark is selected, you must give it a name, as depicted in [Figure 16](#).

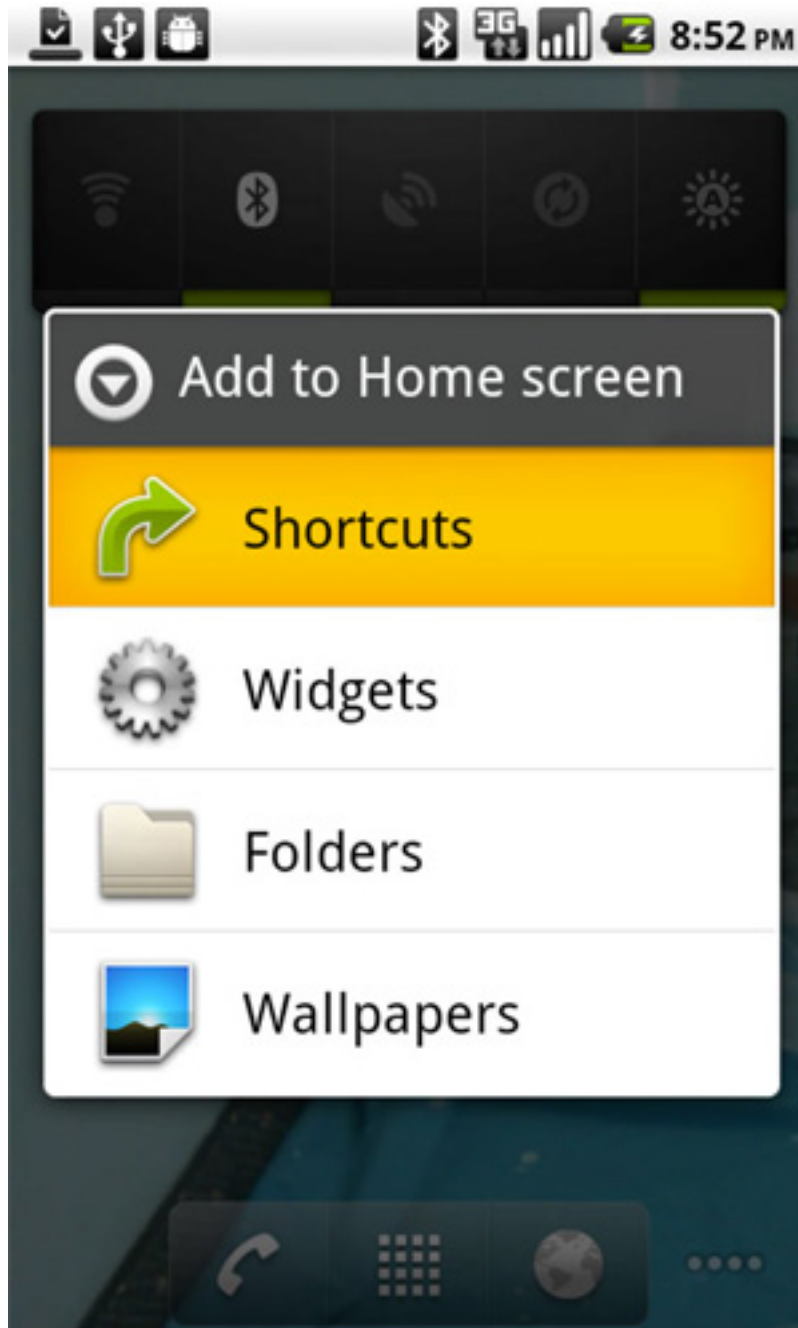
Figure 16. Available bookmarks



Now your bookmark has a name and is available for subsequent use.

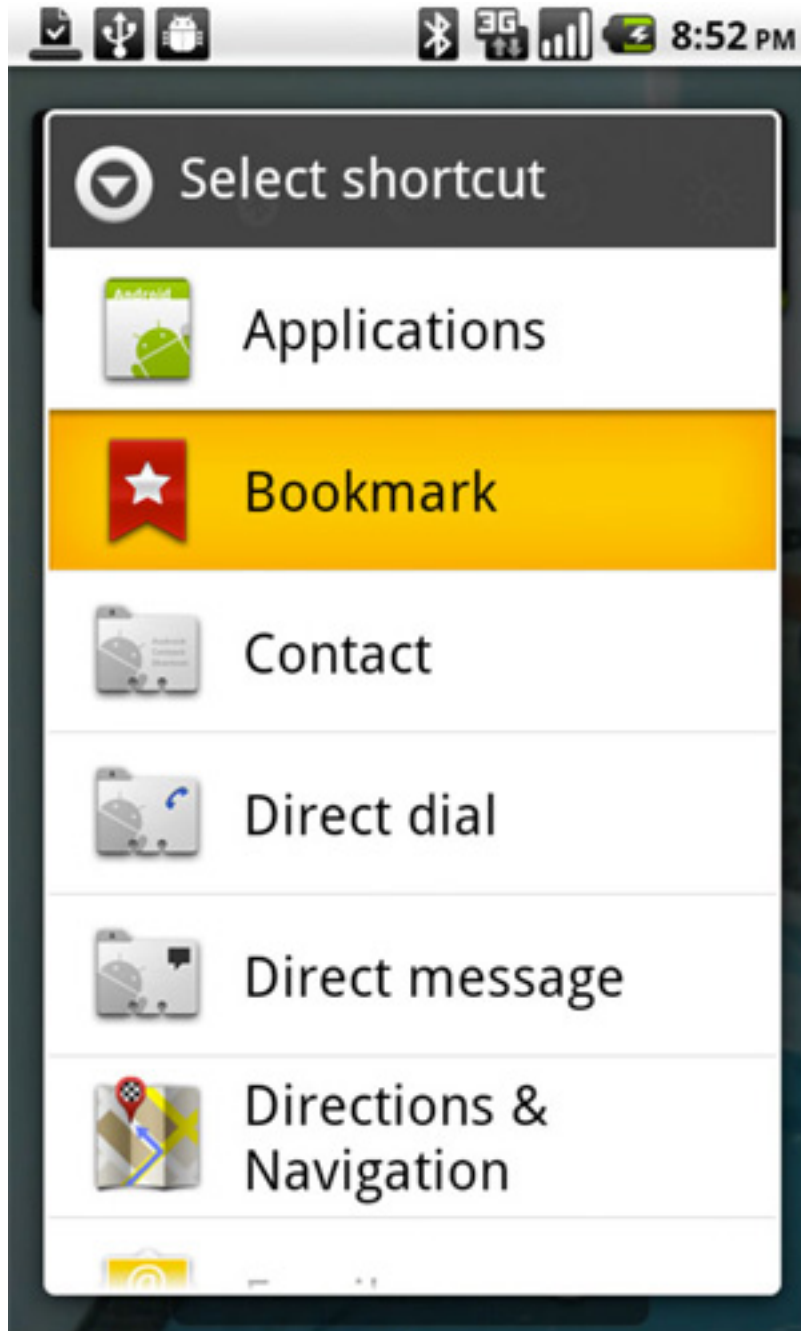
The next step is to go to the home screen, press and hold on a blank area of the screen. This brings up the option for adding items to your home screen. Select the **Shortcuts** option, as in [Figure 17](#).

Figure 17. Add a shortcut



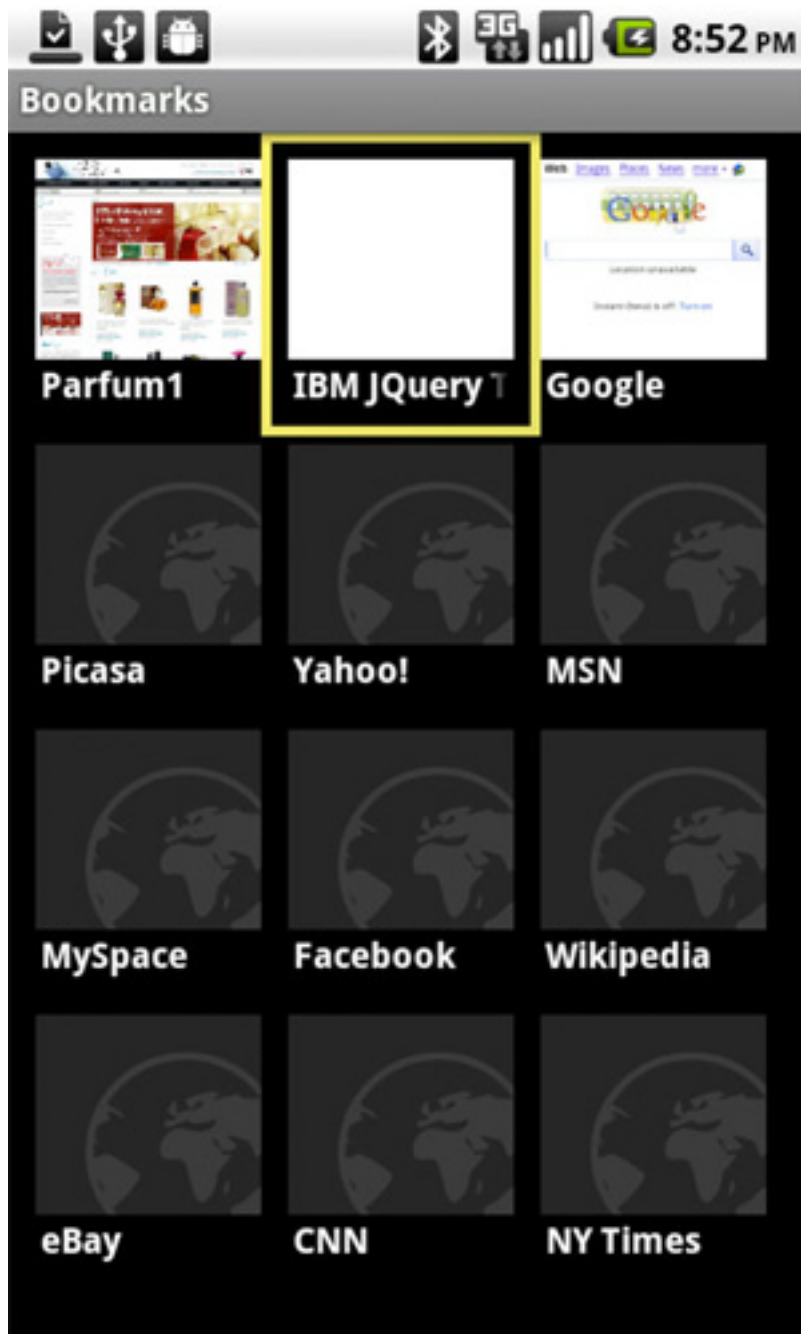
Next, choose **Bookmark** to view existing bookmarks, as in [Figure 18](#).

Figure 18. Choose to view existing bookmarks



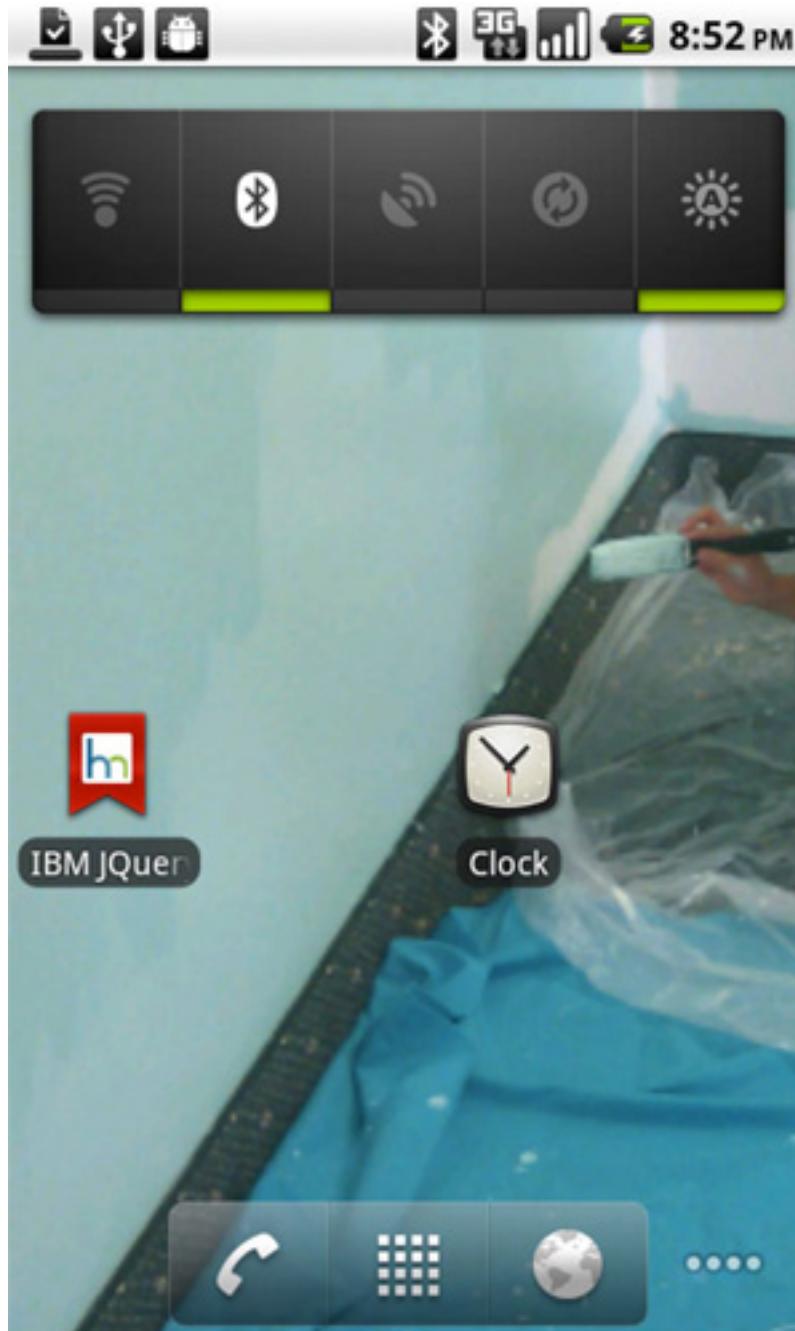
Next, you are presented with the available bookmarks. Choose the appropriate entry for your newly minted jQuery Mobile application, as in [Figure 19](#).

Figure 19. Choose your web application



Finally your shortcut is now on the desktop, as in [Figure 20](#).

Figure 20. Shortcut on the Android home screen



Section 4. Summary

And you're done. If you followed along, you learned about the basic structure of jQuery Mobile and how it relates to the jQuery Core project. You constructed a sales

opportunity tracking application that runs on multiple devices with cross-platform compatibility, and you are equipped to install the application to the home screen of your favorite smart phone.

Downloads

Description	Name	Size	Download method
Tutorial source code	jquery.tutorial.source.zip	3KB	HTTP

[Information about download methods](#)

Resources

Learn

- [jQuery Mobile](#): Visit the home page for a unified user interface system across all popular mobile device platforms, built on the rock-solid jQuery and jQuery UI foundation.
- [jQuery Mobile: Demos and Documentation](#): Access articles, APIs, and demo code for this touch-optimized web framework for smartphones and tablets.
- [jQuery.org](#): Visit the home page of the jQuery open source team.
- [Working with XML on Android](#) (Michael Galpin, developerWorks, June 2009): Learn about the different options for working with XML on Android and how to use them to build your own Android applications.
- [Android in Action, 2nd edition](#) (W. Frank Ableson, Robi Sen, and Chris King; Manning Publications, January 2011): Cover all aspects of Android development including important architectural concepts and implementation strategies through useful and intriguing examples.
- [Mobile Design and Development](#) (Brian Fling, O'Reilly Media, August 2009): Explore practical guidelines, standards, techniques, and best practices for building mobile products.
- [Android SDK documentation](#): Get details on the Android APIs.
- [The Open Handset Alliance](#): Learn more about Android's sponsor, group of 80 technology and mobile companies who have come together to accelerate innovation and improve the mobile experience.
- [Introduction to Android development](#) (Frank Ableson, developerWorks, May 2009): Begin to develop Android apps in this introduction to the Android platform and learn how to code a basic Android app.
- [More articles by this author](#) (Frank Ableson, developerWorks, April 2002-current): Read articles about Android, XML, Eclipse, GPS, BlackBerry applications, and other technologies.
- [XML area on developerWorks](#): Get the resources you need to advance your skills in the XML arena.
- [developerWorks Open source zone](#): Find extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products, as well as our most popular [articles and tutorials](#).
- [IBM developerWorks Industries](#): See this site for all the latest industry-specific technical resources for developers.
- [My developerWorks](#): Personalize your developerWorks experience.

- [IBM XML certification](#): Find out how you can become an IBM-Certified Developer in XML and related technologies.
- [XML technical library](#): See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks. Also, read more [XML tips](#).
- [developerWorks technical events and webcasts](#): Stay current with technology in these sessions.
- [developerWorks on Twitter](#): Join today to follow developerWorks tweets.
- [developerWorks podcasts](#): Listen to interesting interviews and discussions for software developers.
- [developerWorks on-demand demos](#): Watch demos ranging from product installation and setup for beginners to advanced functionality for experienced developers.

Get products and technologies

- The [jQuery Mobile CDN](#): Get jQuery Mobile quickly with already minified and compressed versions of jQuery Mobile.
- [MAMP: Mac - Apache - MySQL - PHP](#): Get and install a Mac-based Apache, MySQL, & PHP environment local server environment.
- [XAMPP](#): Get a very easy-to-install Apache Distribution for Linux®, Solaris, Windows, and Mac OS X. The package includes the Apache web server, MySQL, PHP, Perl, a FTP server and phpMyAdmin.
- [The official Android developers' site](#): Download the Android SDK, access the API reference, and get the latest news on Android. Version 1.5 and later will work. This tutorial uses Android SDK version 8, which supports the Android release labeled 2.2 (Froyo).
- The [Android Open Source Project](#): Get the source code for Android, an open-source software stack for mobile devices.
- [Notepad ++](#): Try a free, source code editor and Notepad replacement that supports several languages. Running in the Microsoft® Windows environment, its use is governed by GPL License.
- [IBM product evaluation versions](#): Download or [explore the online trials in the IBM SOA Sandbox](#) and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- The [jQuery Mobile forum](#): Get all of your jQuery Mobile questions answered.

- [XML zone discussion forums](#): Participate in any of several XML-related discussions.
- The [developerWorks community](#): Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.

About the author

Frank Ableson

W. Frank Ableson is an entrepreneur living in northern New Jersey with his wife Nikki and their children. His professional interests include mobile software and embedded design. He is the author of *Unlocking Android* (Manning Publications, 2009) and *Android in Action* (Manning Publications, 2011) and he is the mobile editor for [Linux Magazine](#).

Trademarks

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. See the current list of [IBM trademarks](#).

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.