



# BuzzTouch URL Scheme with Xcode 5

# **BuzzTouch URL Scheme with Xcode 5**

## **1 1.0 Make a URL Scheme in Xcode**

1.1 1.1 Create a URL Scheme 4

## **2 2.0 Code to put in the App Delegate files**

2.1 2.1 Code for the App Delegate's methods file (.m) 7

## **3 3.0 Testing the App**

3.1 3.1 Use Safari to test the URL Scheme 13

3.2 3.2 Create HTML screen with embedded link 16

3.3 3.3 Run the HTML test 19

3.4 3.4 Create a test PDF File and Screen 23

3.5 3.5 Run the PDF test 28

# 1.0 Make a URL Scheme in Xcode

## 1.1 Create a URL Scheme

The scheme name (or protocol) of a URL is the first part of a URL - e.g. **schemename://**.

iOS supports these **built-in** URL schemes:

- **http, https, ftp** (launches Safari)
- **mailto** Email (launches the Mail app)
- **tel** Telephone Numbers (launches the phone app)
- **sms** Text Messages (launches the SMS app)

A few automatic redirects are pre-programmed into iOS:

- Web links that point to <http://maps.google.com> are redirected to the Maps app.
- Web links that point to <http://www.youtube.com> are redirected to the YouTube app.
- iTunes store links (<https://itunes.apple.com/us/app..>) are sent to the iTunes (or App store) app

iOS apps can also specify their own custom URL scheme (for example, `myapp://doStuff`). When might you want to use a custom URL scheme for your app?

- To transfer data from lite to paid versions of your app
- To allow other apps (or even web pages) to call your app (and send data to it)
- To handle callbacks for custom authentication (such as OAuth) and third party API's

Defining your app's custom URL scheme is all done in the Info.plist file.

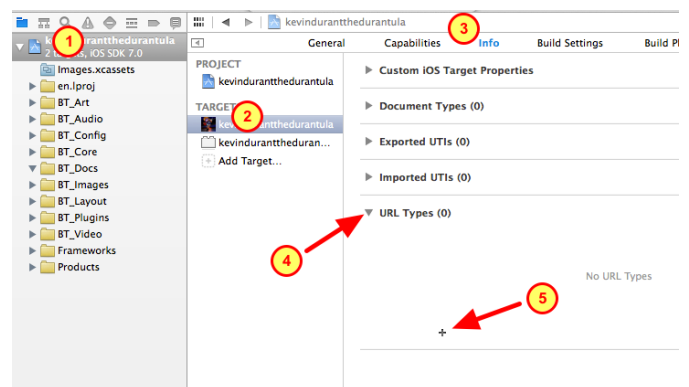
Most of this content in this introductory section comes from the terrific iDev101.com web site on [Custom URL Schemes:](#)

[http://www.idev101.com/code/Objective-C/custom\\_url\\_schemes.html](http://www.idev101.com/code/Objective-C/custom_url_schemes.html)

### 1.1.1 Create an empty URL Type

To change the plist file to be recognized as an Info.plist property list type:

1. Open up the **project** by clicking on the blue icon in the left column
2. Click on the **Target** for the app
3. Click on the **"Info"** tab
4. Expand the **URL Types** section by clicking on it's triangle



5. Create a new **URL Type** by clicking on the plus sign

## 1.1.2 Specify Parameters for the URL Type

Create the definition of your new URL Type for the app

1. Specify the **identifier**, usually this is the Bundle ID in the format of `com.CompanyName.AppName`

For example, `com.bluegridapps.durantula` is shown in the illustration

2. Enter the **URL Scheme**, this is in the format of **scheme://ScreenNickname**

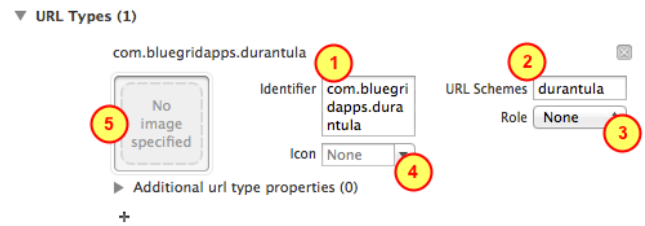
For example, `durantula://ScreenNickname` is the URL Scheme shown in the illustration

The Scheme should be unique such that no other app will use it

3. The Document **Role** is set to None

4. The **Icon** is set to None

5. No **image** for the icon is specified



## **2.0 Code to put in the App Delegate files**

## 2.1 Code for the App Delegate's methods file (.m)

This section tells you what code should be inserted into the App Delegate's methods file (.m)

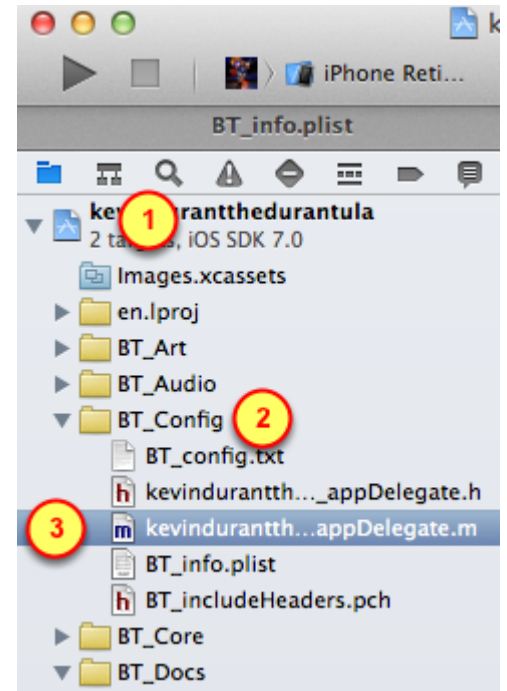
After doing a copy-paste of the code into the .m file, then each code section is explained.

### 2.2.1 Open the App Delegate's methods file (.m)

Find and edit the App Delegate's methods file (.m)

1. Open the **Project**
2. Open the **BT\_Config** group folder
3. Click on the **App Delegate's methods file (.m)**

It is in the format of *yourAppName\_appDelegate.m*



### 2.2.2 Code for the handleOpenURL method

Copy-paste this code into the App Delegate's Methods file (.m)

No changes are necessary, just do a copy-paste of the code into the methods file (.m).

Location-wise, paste the code just before the end of the file, right before the **@end** declaration.

For an explanation of the code, proceed to the next set of steps. Each code section will be explained, but no action is necessary because you've already pasted the code from this step into your file.

----- COPY BELOW THIS LINE -----

```
// Handle taps on Custom URL Scheme for this unique app
// Custom URL = myCoolApp://myCoolScreenNickname
// Scheme = myCoolApp, it is registered in the BT_Info.plist file
// Host = myCoolScreenNickname, is the Nickname of the next screen to
```

```

be displayed
//   XXX = replace that with yourAppName, such as myCoolApp
//
// Get more info on Custom URL Schemes from this excellent web site
//   http://www.idev101.com/code/Objective-C/custom\_url\_schemes.html

-(BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url {

    XXX_appDelegate *appDelegate = (XXX_appDelegate *) [[UIApplication sharedApplication]
delegate];

    [BT_debugger showIt:self theMessage:@"Tapped on a custom URL"];

    // Check if we got a screen nickname from the URL to process

    NSString *loadScreenNickname = [url host];

    BT_item *screenObjectToLoad = nil;

    if ( [loadScreenNickname length] > 1 ) {
        [BT_debugger showIt:self theMessage:[NSString stringWithFormat:@"Nickname of screen
to load: %@", loadScreenNickname]];

        screenObjectToLoad = [appDelegate.rootApp
getScreenDataByNickname:loadScreenNickname];
    } else {
        [BT_debugger showIt:self theMessage:@"ScreenNickname to load is blank"];

        return NO;
    }

    // Check if the Screen was found within the App's JSON data

    if ( screenObjectToLoad != nil ) {
        NSLog(@"loading screen:%@", screenObjectToLoad.itemId);

        [self.getViewControllor handleTapToLoadScreen:screenObjectToLoad
theMenuItemData:nil];
    } else {
        [BT_debugger showIt:self theMessage:[NSString stringWithFormat:@"URL Scheme
error: %@", url]];
    }
}

```



```

}

return YES;
}

```

### 2.2.2.1 Register a Handler for custom URL

Just before the dealloc method, the Handler for the Custom URL Scheme is created.

Any time the Custom URL is tapped on the device (within any app), then this App is launched.

Then the Handler code is executed.

```

//Handler for the URL Scheme for PDFs
// Get more info on Custom URL Schemes from this excellent web site
// http://www.idev101.com/code/Objective-C/custom_url_schemes.html
-(BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url {
    XXX_appDelegate *appDelegate = (XXX_appDelegate *) [[UIApplication sharedApplication]delegate];
    [BT_debugger showIt:self theMessage:@"Tapped on a custom URL"];
}

```

### 2.2.2.2 Use the Screen Nickname to get Information for the next Screen

This section grabs the Screen Nickname from the Host field of the URL:

Then using that Screen Nickname, it gets the actual Screen object.

```

// Check if we got a screen nickname from the URL to process
NSString *loadScreenNickname = [url host];
BT_item *screenObjectToLoad = nil;
if ( [loadScreenNickname length] > 1 ) {
    [BT_debugger showIt:self theMessage:[NSString stringWithFormat:@"Nickname of screen to load: %@", loadScreenNickname]];
    screenObjectToLoad = [appDelegate.rootApp getScreenDataByNickname:loadScreenNickname];
} else {
    [BT_debugger showIt:self theMessage:@"ScreenNickname to load is blank!"];
    return NO;
}

```

### 2.2.2.3 Load the next screen to be displayed

The software does three primary things in this section:

1. Check if the Screen Nickname led us to an actual screen that has data in the BT\_config.txt file (the JSON data)
2. Load the next screen using it's object (information)
3. Otherwise, report an error that the Screen Nickname did not have the associated screen object (information) within the JSON data.

```

// Check if the Screen was found within the App's JSON data
if ( screenObjectToLoad != nil ) {
    NSLog(@"loading screen:%@", screenObjectToLoad.itemId);
    [self.getViewController handleTapToLoadScreen:screenObjectToLoad theMenuItemData:nil];
} else {
    [BT_debugger showIt:self theMessage:[NSString stringWithFormat:@"URL Scheme error: %@", url]];
}
return YES;

```

Using the given nickname for the Screen, there was not a match in the JSON data for that nickname.

### 2.2.3 Code for the `getViewController`

In the App Delegate's Methods file (.m), find the method `getViewController`

Comment out that method by prefixing it with `/*` and suffixing it with `*/`

When done, it should look like this:

```
/*
//getViewController...
-(BT_viewController *)getViewController{
    return (BT_viewController *)[self navigationController] topViewController;
}
*/
```

Right under that old `getViewController` method, copy-paste the next block of code right below it.

----- COPY-PASTE BELOW THIS LINE -----

```
// getViewController...
-(BT_viewController *)getViewController{

    BT_viewController *theViewController;

    int selectedTab = 0;

    // IF have a Tabbar THEN get the active tab's viewController object
    // ELSE get the current screen's viewController object

    if ( [self.rootApp.tabs count] > 0 ) {
        selectedTab = [self.rootApp.rootTabBarController selectedIndex];

        theViewController = (BT_viewController
*)[[self.rootApp.rootTabBarController.viewControllers objectAtIndex:selectedTab]
```

```
visibleViewController];

    }else{
        theViewController = (BT_viewController *)[self.rootApp.rootNavController
visibleViewController];
    }

    return theViewController;
}
```

# 3.0 Testing the App

### 3.1 Use Safari to test the URL Scheme

Since the new URL Scheme is an Internet object, lets use an Internet tool to test it.

Safari in the Emulator or Safari on the Mobile Device will be the testing tool.

Be sure to load the app into the Emulator or onto the Mobile Device.

In Safari's URL bar, type in the URL Scheme for the app.

For example, **durantula://More** or **durantula://LA%20Arena**

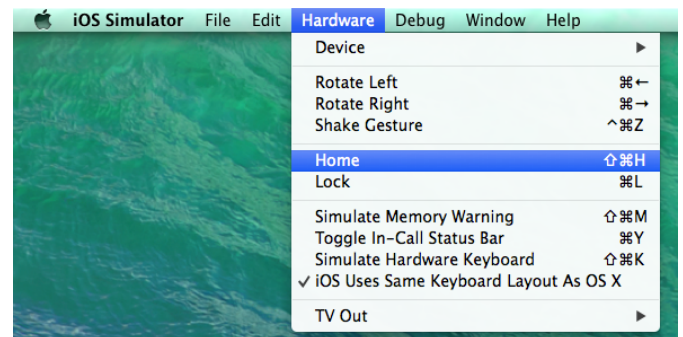
#### 3.1.1 Run the app in the Emulator

1. After the URL Scheme has been configured within Xcode and the two methods have been put into the App Delegate's method file, then the App can be run on the Emulator.
2. Test the various screens of the app to make sure it is functional.
3. Pick a screen to be displayed via the Custom URL.
4. Look at the **BT\_config.txt** file to get the correct Screen Nickname by looking at the **itemNickname** property.
5. For example, the Custom URLs are: **durantula://More** or **durantula://LA%20Arena**

( Notice that **%20** is used wherever a **space** exists in the itemNickname )

#### 3.1.2 Switch to Safari

1. Using the **Hardware** menu of the iOS Simulator, select the **Home** menu item to close the App
2. Click on Safari to run it



### 3.1.3 Test the Custom URL in Safari

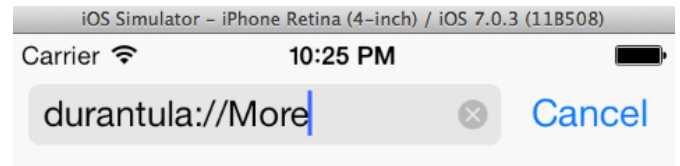
1. In Safari's URL bar, type in the custom URL of the type: **scheme://itemNickname**

**scheme** = value specified in the URL Schemes box when configuring within Xcode

**itemNickname** = value of property within the BT\_config.txt JSON data file, this screen will be displayed by the app

Example: **durantula://More** or **durantula://LA%20Arena**

2. Click on the Go button (or press the return key on the keyboard)

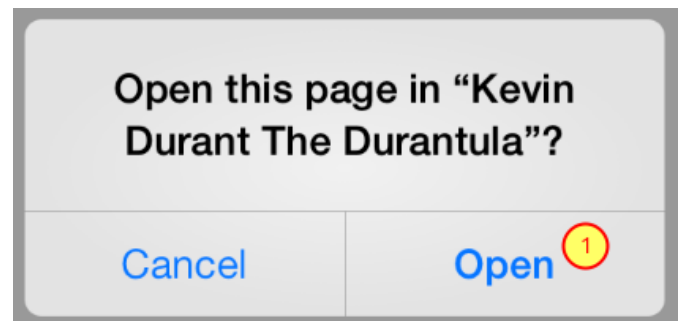


### 3.1.4 Verify the app was launched and correct screen is displayed

1. Accept confirmation that you want to launch the app for that custom URL

2. Verify the app with the Custom URL was launched

3. Verify the specified screen is displayed



### 3.1.5 Error - Wrong screen is displayed

If the correct screen is NOT displayed, then check for these common errors:

1. Uppercase and lowercase letters are correct in Safari's URL bar
2. Each space between words of the nickname is replaced with a %20 designation



## 3.2 Create HTML screen with embedded link

---

After configuring the App for a Custom URL and establishing code to handle taps on the Custom URL, we are ready to do some testing!

There are two ways a Custom URL can be used:

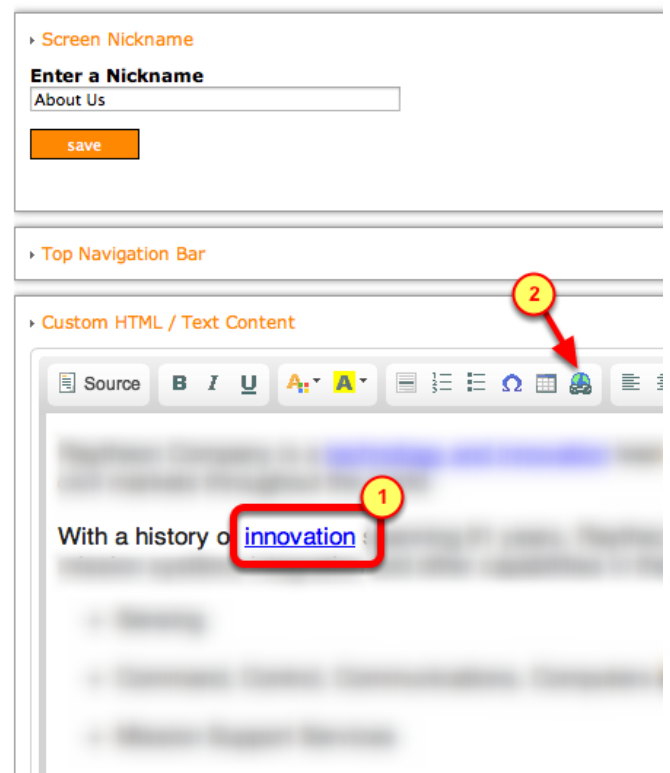
1. As an link in HTML
2. As a link in PDF

For the first test, we will make an HTML screen that contains a link to another screen within the app.

### 3.1.1 Make an About Us screen

Hyperlink words to point to another screen within this app.

1. Select and highlight the words
2. Click on the **Link Tool** to specify the custom URL





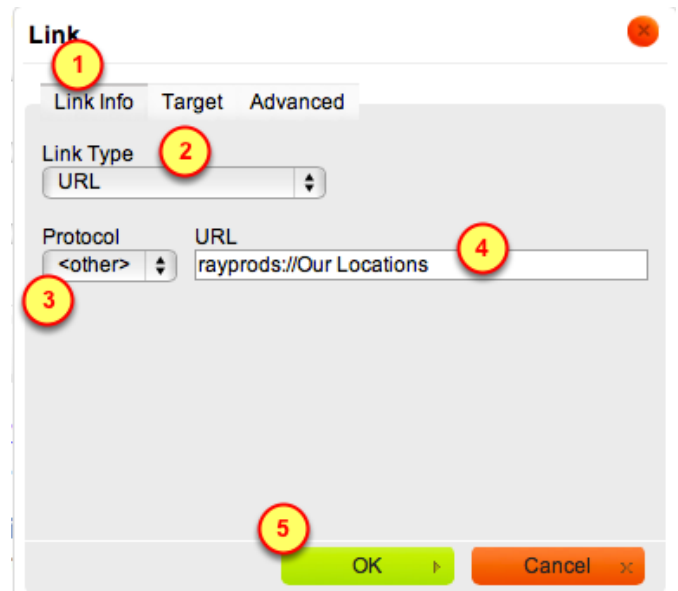
### 3.1.2 Add a custom URL link into HTML

For this HTML link, specify this is a custom URL

1. Be on the **Link Info** tab
2. The **Link Type** should be of type **URL**
3. The **Protocol** should be **<other>**
4. The **URL** value should be the **Custom URL**

\* Spaces are allowed in the custom URL, make an exact match for the Screen Nickname)

5. Click on the **OK** button to save the entry for the custom URL



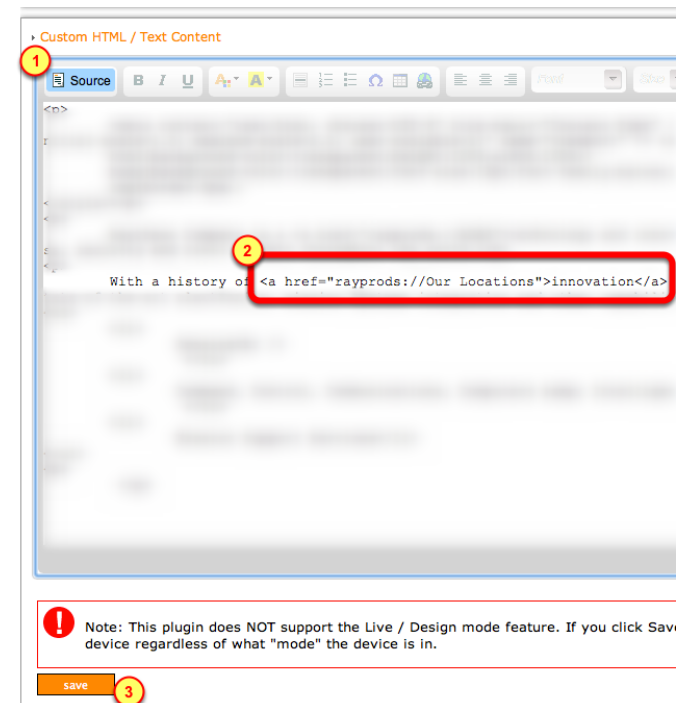
### 3.1.3 Inspect the custom link

View the source code to validate the HTML is correct

1. Click on the **Source** button to see the actual HTML
2. Verify the Custom URL looks like this:

```
<a href="rayprods://Our Locations">innovation</a>
```

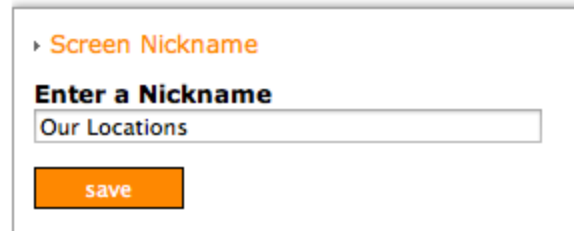
3. Click on the **save** button



### 3.1.4 Make a Map screen for Our Locations

Make a Map Screen.

1. Give it a Screen Nickname of "Our Locations"



▸ Screen Nickname

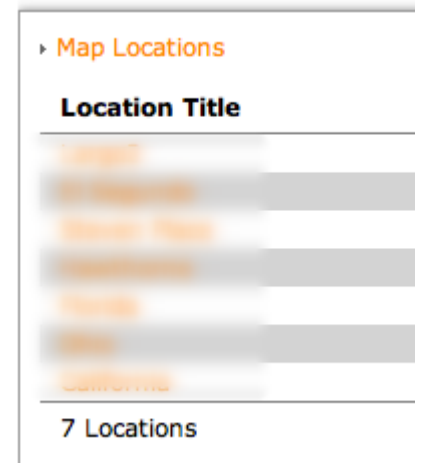
**Enter a Nickname**

save

### 3.1.5 Add map locations for Our Locations screen

Within the Map screen, add a few locations.

The quantity of locations is not important, two is enough.



▸ Map Locations

**Location Title**

- [blurred]
- [blurred]
- [blurred]
- [blurred]
- [blurred]
- [blurred]
- [blurred]

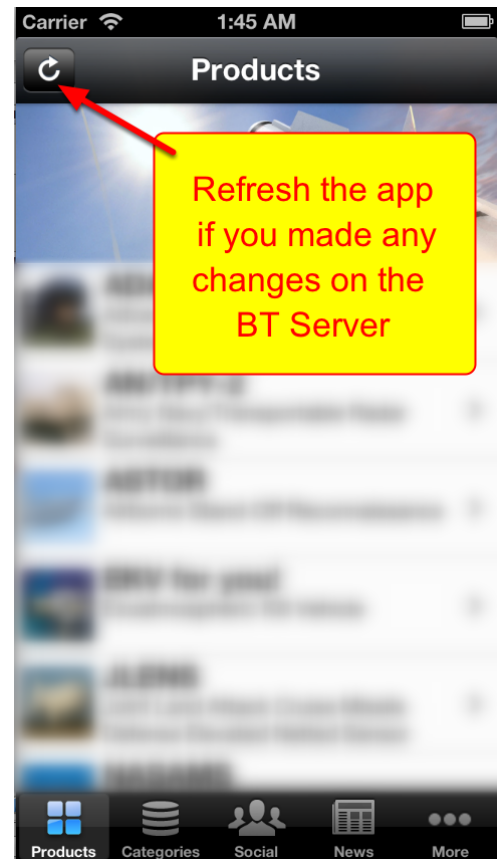
7 Locations

### 3.3 Run the HTML test

---

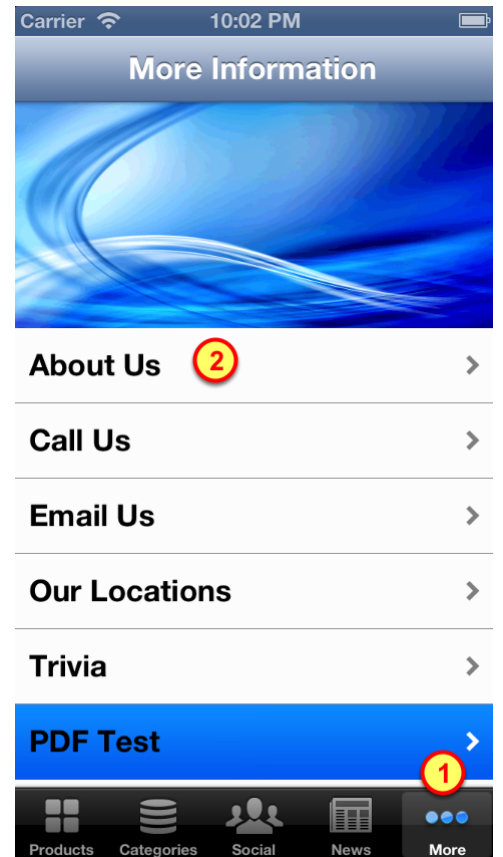
#### 3.2.1 Run the App in Simulator

If any changes were made on the BuzzTouch server, be sure to get the latest configuration and data into the app.



### 3.2.2 Menu to the HTML screen

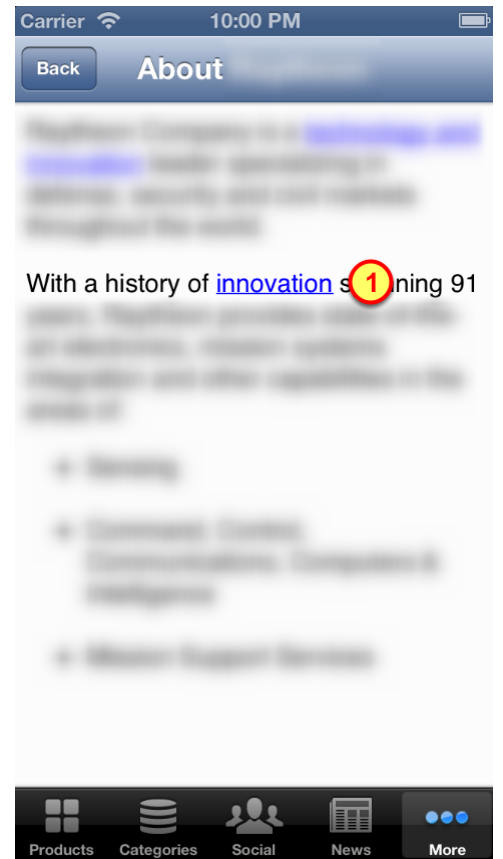
1. Navigate to the menu items table (can be anywhere within the app)
2. Tap on the menu item to show the HTML content



### 3.2.3 HTML Screen with link to another screen

Locate the Link on the HTML screen

1. Tap on the the Link to view the map of our locations

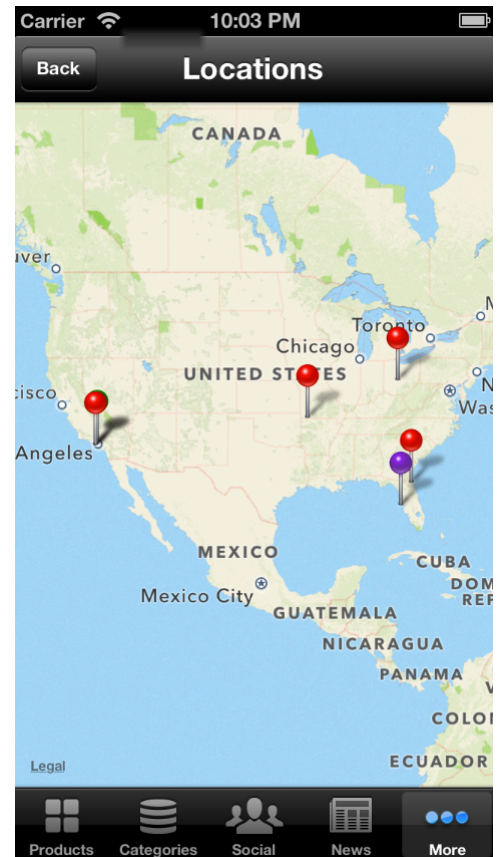


### 3.2.4 Screen is displayed by using it's nickname

This screen has a nickname of "Our Locations".

A tap on the link on the HTML screen made a request for the screen with a nickname of "Our Locations".

After fetching that screen's object from the JSON data, the actual "Our Locations" screen is then displayed.



### 3.4 Create a test PDF File and Screen

After configuring the App for a Custom URL and establishing code to handle taps on the Custom URL, we are ready to do some testing!

There are two ways a Custom URL can be used:

1. As an link in HTML
2. As a link in PDF

For the second test, we will make a PDF file that contains a link to another screen within the app.

The PDF file will be uploaded to the BuzzTouch server. Then we shall attach that PDF file to a menu item, which then can be shown on a screen.

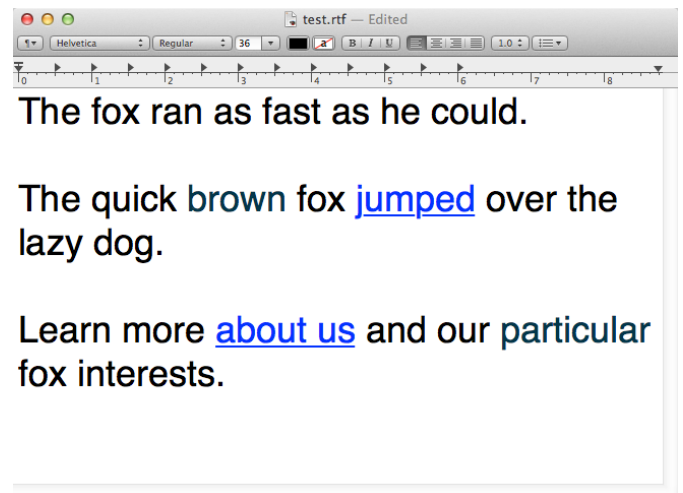
#### 3.3.1 Create a text file

Using the TextEdit application on the Mac, create a test document with sample text:

The fox ran as fast as he could.

The quick brown fox jumped over the lazy dog.

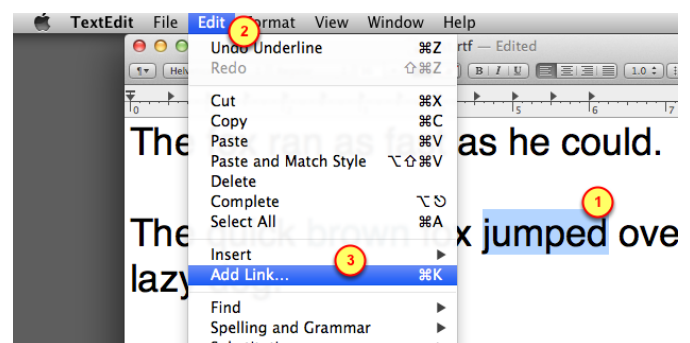
Learn more about us and our particular fox interests.



#### 3.3.2 Create custom links in the text document

Lets hyperlink one of the words for the Custom URL link:

1. Select and **highlight** a word to be hyperlinked
2. Click on the **Edit** menu



3. Select the **Add Link** menu item

### 3.3.3 Specify the custom URL

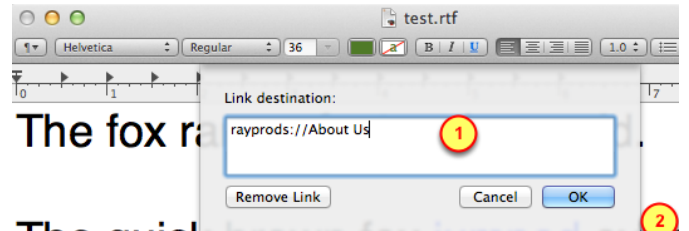
Insert the custom URL in the dialog box:

1. Specify the link for the **custom URL**.

Remember that it must have two components:

- **Scheme:** rayprods
- **Screen Nickname:** About Us (it is okay to have spaces in the nickname)

2. Click on **OK** to complete this step



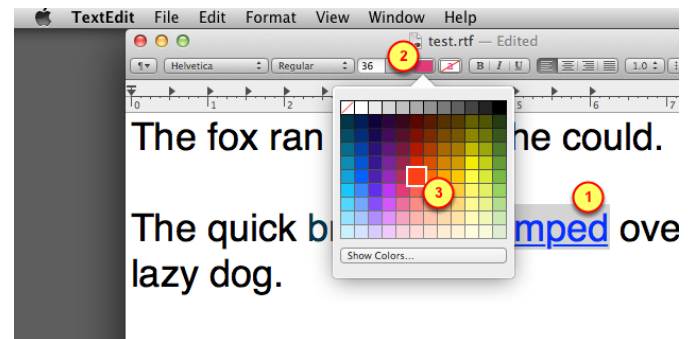
The fox ran  
The quick brown fox jumped over  
lazy dog.

Learn more about us and our par  
fox interests.

### 3.3.4 Color the link to look like a link

Even though the link looks like a link within the text editor, it won't look like it as a PDF. Lets color that link by changing it's color.

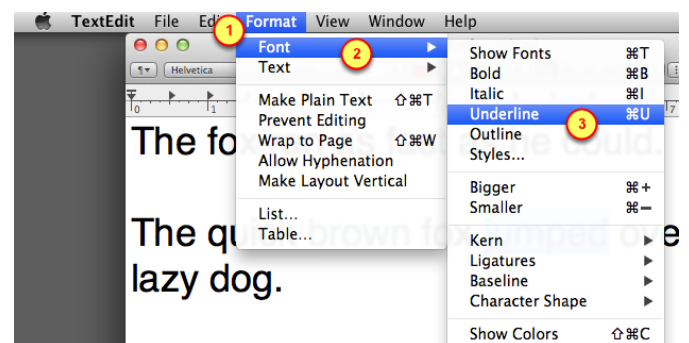
1. Select and **highlight** the word
2. Using the **color box**, select a **color** for the word



### 3.3.5 Underline the link to look like a link

Lets underline the link explicitly, even though it already looks like an underlined link:

0. Select and **highlight** the word
1. Click on the **Format** menu
2. Click on the **Font** menu
3. Select the **Underline** menu item

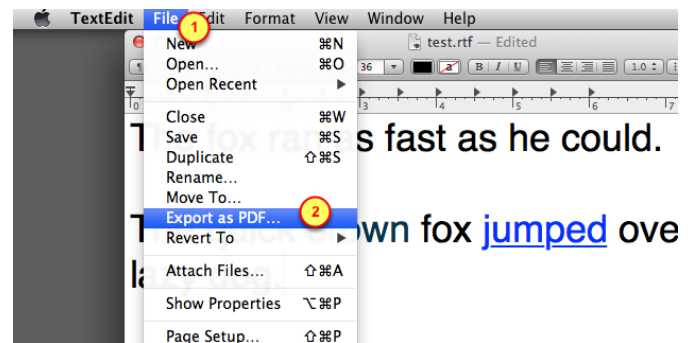




### 3.3.6 Print a PDF version of the text file

Generate a PDF version of the test file

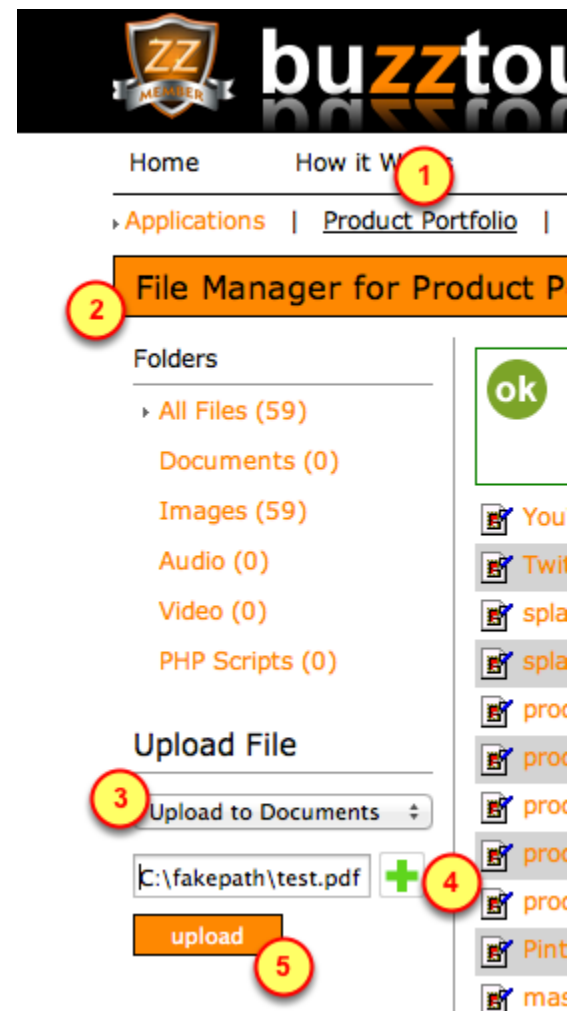
1. Click on the **File** menu
2. Select the **Export as PDF** menu item



### 3.3.7 Upload the PDF file to BuzzTouch server

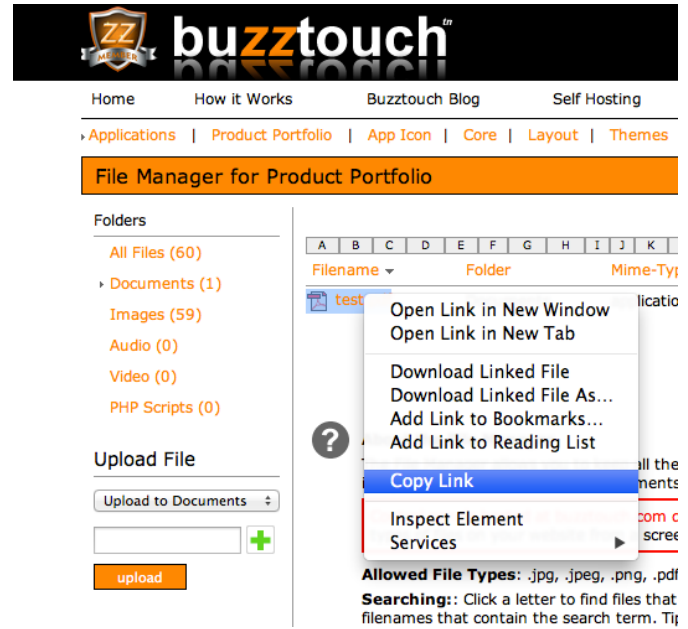
Upload the test PDF file into the BuzzTouch server for your App

1. Select the **Application** being tested
2. Select the **Files** section of the Application
3. Select **Documents** as the file type
4. **Browse** to the PDF file
5. Click on the **upload** button



### 3.3.8 Copy link for PDF file

Get the link to the PDF file, it will be used as a sample screen.



### 3.3.9 Add a BuzzTouch menu item pointing to PDF file

Lets get that PDF file into the App by connecting it to a BuzzTouch Menu by adding a new row:

1. Specify the **tile** of the row to show a User
2. Give a **nickname** for the screen that will be displaying the contents of that PDF file
3. The **Screen Type** should be "PDF Doc"
4. Click on the **add** button to create a new row that points to that PDF file

Selecting this menu item...	Loads this screen /
About Us	About
Call Us	Call Us
Email Us	Email Us
Our Locations	Locations
Trivia	Trivia

5 Menu Items

### 3.3.10 Select the PDF Screen for configuration

Now that we've created a screen to display a PDF, we must connect that screen and the PDF file.

1. Click on the screen for the **PDF File**

Selecting this menu item...	Loads this screen ,
About Us	About
Call Us	Call Us
Email Us	Email Us
Our Locations	Locations
Trivia	Trivia
PDF Test	PDF File

### 3.3.11 Insert link to PDF file

Lets connect the screen that is showing the PDF contents to the actual PDF file

1. Click on **Screen Nickname** to verify we are configuring the correct screen (the one for the PDF File)
2. Click on **Document Location** to fill-in the URL box
3. Paste in the link to the PDF File into the **URL box**
4. Click on the **save** button

• Screen Nickname

Enter a Nickname  
PDF File

save

• Top Navigation Bar

• Document Location

File Name in Project → Select

OR an internet address (enter one or the other, not both)

Load from this URL instead → Select

https://www.buzztouch.com/documents/test.pdf

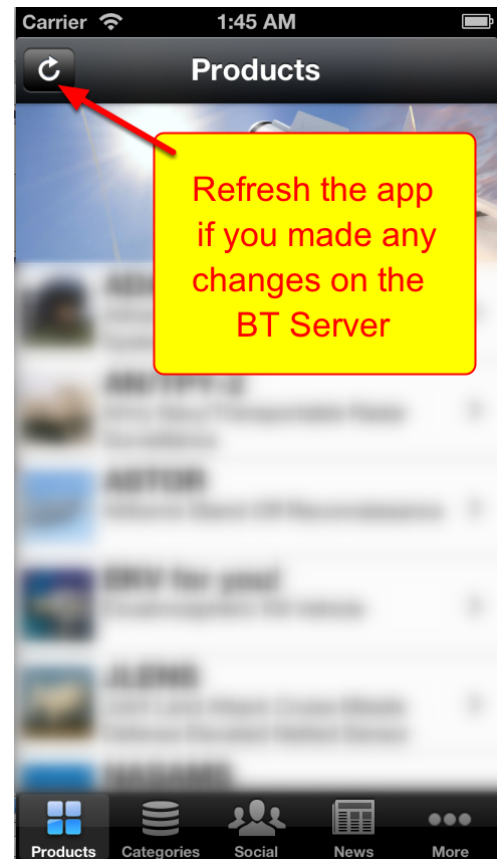
save

## 3.5 Run the PDF test

---

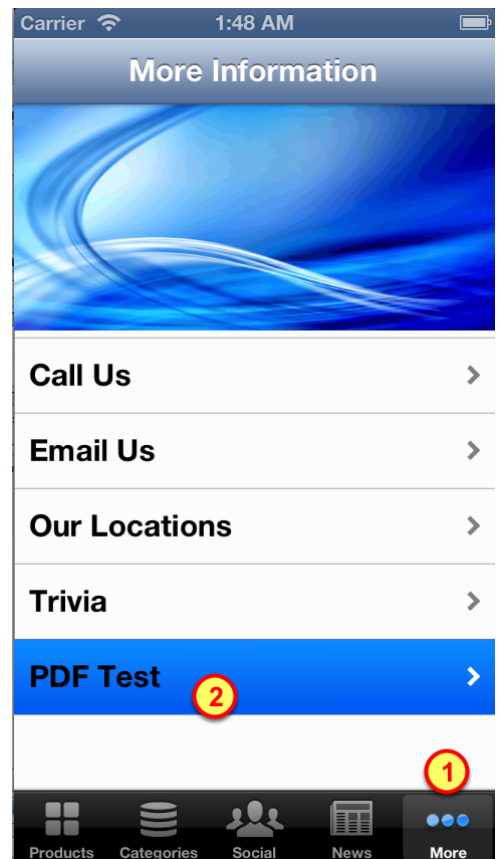
### 3.4.1 Run the App in Simulator

If any changes were made on the BuzzTouch server, be sure to get the latest configuration and data into the app.



### 3.4.2 Select the PDF Test menu item

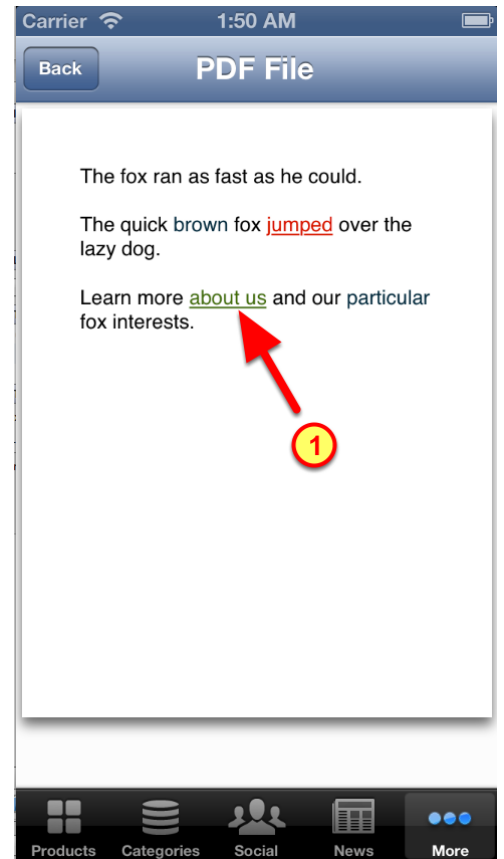
1. Switch over to the screen that shows the PDF menu item (could be located anywhere in your sample app)
2. Tap on the menu item to view the PDF file



### 3.4.3 Show the contents of the PDF file

View the contents of the PDF file. Tap on links that will take you to another screen within the App:

1. Tap on **about us** to read the screen displaying information **about the providers** of this app



### 3.4.4 Screen is displayed by using it's nickname

This screen has a nickname of "About Us".

A tap on the link in the PDF makes a request for the screen with a nickname of "About Us".

After fetching that screen's object from the JSON data, the actual screen is then displayed.

