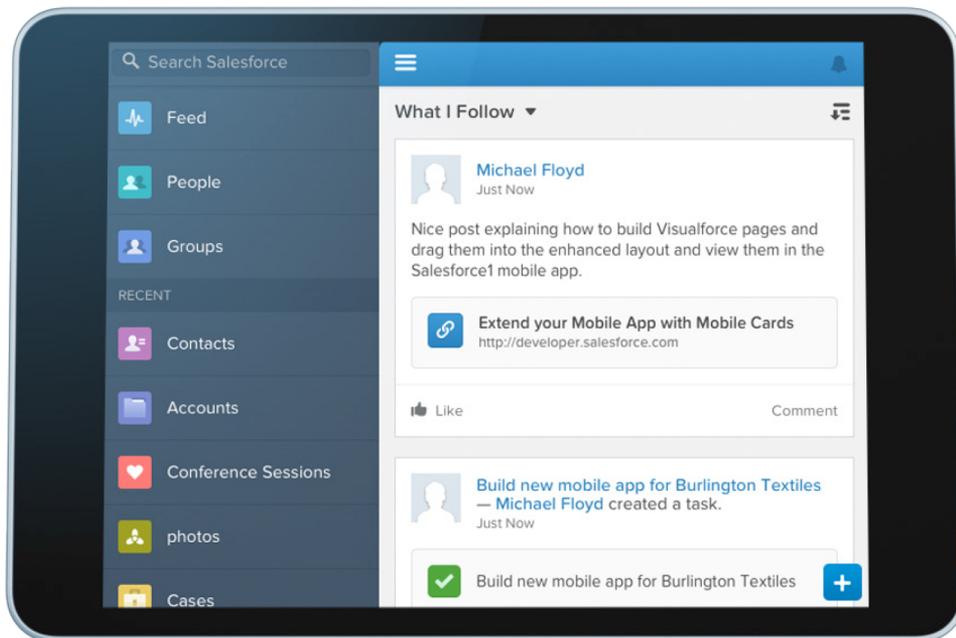


# Salesforce1 Mobile Workbook

Build Mobile-enabled Cloud Apps on Force.com



Configure & Use

Mobile Cards

Compact Layouts

Object-specific Actions

Global Actions

Visualforce & Apex



# Table of Contents

<b>About the Salesforce1 Mobile App Workbook.....</b>	<b>1</b>
<b>Tutorial 1: Initial Set Up.....</b>	<b>2</b>
Step 1: Get a New DE Org.....	2
Step 2: Create an App.....	2
Step 3: Download the Salesforce1 Mobile App.....	2
<b>Tutorial 2: Use the Mobile App.....</b>	<b>4</b>
Step 1: Create Your First Post.....	4
Step 2: Create a Task.....	5
Step 3: Use the Today app.....	5
Step 4: Navigate to a Record.....	5
Step 5: Try a Record Action.....	8
Step 6: Add a Record to Your App.....	8
Step 7: Pin Frequently Used Searches.....	9
<b>Tutorial 3: Optimize for the Mobile Display.....</b>	<b>11</b>
Step 1: Create a Page Layout for a Mobile User Profile.....	11
Step 2: Display Key Fields Using Compact Layouts.....	13
Step 3: Add Mobile Cards to the Record Related Information Page.....	13
Step 4: Enable Notifications.....	14
<b>Tutorial 4: Quickly Create Records Using Global Actions.....</b>	<b>15</b>
Step 1: Create a Global Action.....	15
Step 2: Customize the Global Publisher Layout.....	15
<b>Tutorial 5: Create Related Records with Object-Specific Actions.....</b>	<b>17</b>
Step 1: Define an Object-Specific Action.....	17
Step 2: Choose Fields and Predefine Values.....	17
Step 3: Customize an Object-Specific Layout.....	18
<b>Tutorial 6: Develop a Visualforce Page and Add it to the Navigation Menu.....</b>	<b>19</b>
Prerequisites: Set Up Your Development Environment.....	19
Step 1: Install the Enhanced Warehouse Data Model.....	19
Step 2: Access the Mobile Browser App.....	19
Lesson 1: Create the FindNearby Apex Class.....	20
Step 1: Create the FindNearby Apex Class.....	20
Step 2: Create the getNearby Method.....	20
Step 3: Add Default Location Logic.....	21
Step 4: Run a Query and Return Results.....	21
Summary: Check Completed Code.....	21
Lesson 2: Create the Visualforce Page.....	22
Step 1: Bind the extension and Standard Controller to a Visualforce Page.....	22
Step 2: Add Static Resources to the Page.....	23

Step 3: Place a Container div for Rendering the Map.....	23
Step 4: Add the initialize JavaScript function.....	23
Step 5: Add the createMap function.....	24
Step 6: Create Markers for Nearby Warehouses.....	25
Step 7: Check the Final Page.....	26
Lesson 3: Expose the Page in Salesforce1.....	28
Step 1: Create a Tab.....	28
Step 2: Add the Tab to Mobile Navigation.....	29
Step 3: Try Out the App.....	29

# About the Salesforce1 Mobile App Workbook



While you can use the Salesforce platform to build virtually any kind of app, most apps share certain characteristics, such as:

- A database to model the information in the app
- A user interface to expose data and functionality to those logged into your app
- Business logic and workflow to carry out particular tasks under certain conditions

In addition, apps developed on the Salesforce Platform automatically support:

- A mobile app that is easy to use, customize, and further develop
- A public website to allow access to data and functionality
- A native social environment that allows you to interact with people or data
- Built-in security for protecting data and defining access across your organization
- Multiple APIs to integrate with external systems
- The ability to install or create packaged apps

This workbook shows you how to use, configure, and develop the Salesforce1 mobile app in a series of tutorials. Initially you create a very simple app to track your learning progress, which is enough to show you the basics. If you following along to the end of this tutorial, you'll also install a Warehouse app that will help you learn more sophisticated examples with code.

# Tutorial 1: Initial Set Up

The first thing you need to do is set up your development and testing environment. Then you'll set up your browser and mobile device for testing.

## Step 1: Get a New DE Org

If you already have a Developer Edition org, you might not need a new one. However, if you've previously done any tutorials with the Warehouse app, the advanced section at the end of this workbook uses this app, and there might be a conflict when installing the package. While you might be able to navigate around potential conflicts, it's easier and faster to create a new org. In addition, Developer Edition orgs created before the Winter '14 release don't have some necessary functionality.

1. In your browser, go to <http://bit.ly/slgettingstarted>.
2. Fill in the fields about you and your company.
3. In the `Email Address` field, make sure to use a public address you can easily check from a Web browser right now.
4. Type a unique Username like `firstname.lastname@slworkshop.com`.
5. Read and then select the checkbox for the `Master Subscription Agreement`, and then click **Submit Registration**.
6. In a moment you'll receive an email with a login link. Click the link and change your password.

## Step 2: Create an App

To understand how to access custom apps in the Salesforce1 mobile app, you need to create an app. Use the App Quick Start wizard to create an app that will help you track your learning progress.

1. On the Force.com Setup page, click the green **Add App** button.
2. Fill in the form as follows:
  - For the App type `Learn Salesforce1`.
  - For the Label, type `Lesson`.
  - For the Plural Label, type `Lessons`.
3. Click **Create** and once the wizard finishes, click **Go To My App** and then **Start Tour**, to get a quick overview of your app's interface.
4. Click **New** to create a new Lesson. Name it `Use Mobile App` and then click **Save**.

## Step 3: Download the Salesforce1 Mobile App

For final testing, you'll also need to install the Salesforce1 mobile app on your device. If you've already downloaded the Salesforce1 mobile app, you can skip this step.



**Note:** If you're already using the Salesforce1 mobile app, you can skip this section. Note that you'll have to log out of the mobile app and log back in with the credentials of your new org.

1. Use your mobile device's browser to go to [www.salesforce.com/mobile](http://www.salesforce.com/mobile), select the appropriate platform, and download Salesforce1.
2. Open Salesforce1 from your mobile device.
3. Enter the login credentials from your new org and tap **Log in to Salesforce**.

4. If this is the first time using the Salesforce1 mobile app, the phone prompts you to email a verification code. Tap the button, check your email, and copy the verification code into the space provided.
5. Tap **Verify my code and log me in.**, and then tap **Allow** so that the app can access to your data..

## Tutorial 2: Use the Mobile App

When you create an app in Salesforce, you automatically create a mobile version of the app. Indeed, you could say that every Salesforce developer is a mobile developer!

In this tutorial you access the mobile version of the app you just downloaded. Because the way you navigate apps, tabs, and records is different for mobile, your first task is to go through a quick tour of the mobile app. Along the way you'll find out what the various parts of the mobile interface are called and how to add more functionality to them.

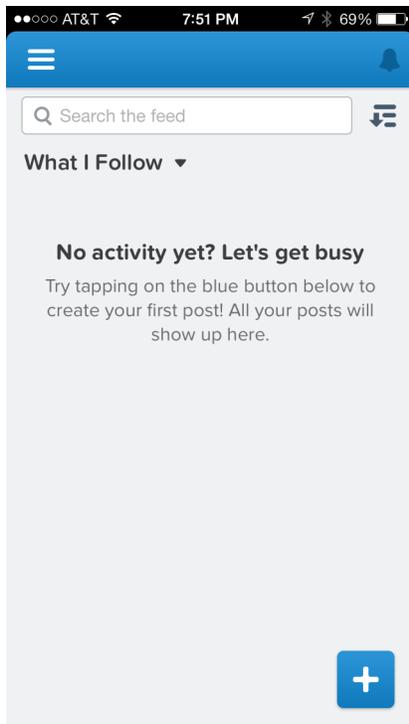


**Note:** For the following exercises, you need to use the downloaded mobile app on your phone. The mobile browser app doesn't support the phone, the Today app, and other required features.

### Step 1: Create Your First Post

When you start up the Salesforce1 mobile app for the first time, you are prompted to create your first post.

1. Tap .



2. Tap the Photo icon and choose **New Photo**.
3. Snap a selfie (or choose a photo from your library) and then add a title like "First Post!"
4. Tap **Done** and then **Share** it.

Your posts shows up on your feed, so other people that follow you can keep up with what you're doing. The photo itself is stored as a file and can be attached to other Salesforce records as well.

### Tell Me More....

The downloadable mobile app provides the best mobile experience. However, you can also access a fully supported version of Salesforce1 from any mobile browser. Developer Edition orgs are already enabled for the mobile browser, but if you want to enable this feature for your company's org, you need to configure that setting in the full (non-mobile) Salesforce site:

1. From **Setup** click **Mobile Administration** > **Salesforce1**.
2. Select **Enable the Salesforce mobile browser app**.

Now, when you navigate to [login.salesforce.com](https://login.salesforce.com) from your mobile browser, Salesforce will recognize that you're working from a mobile device and redirect you to the Salesforce1 mobile browser app.

## Step 2: Create a Task

Mobile apps are all about being productive in micro moments, so you can start by creating a task for yourself.

1. In the bottom right corner, tap  and the *publisher* opens. Tap **New Task**.
2. For the Subject enter `Sync my calendar in the Today app`.
3. Use the calendar control to set the Due Date to **Today** and then tap **Submit**.

When you created that task, you probably noticed some other actions you could take in the publisher. All of those actions are *global actions*. Global actions aren't associated with any other data, and can be thought of as quick things that you'll follow up with later. In Salesforce, you can create your own custom global actions and add them to the publisher.

## Step 3: Use the Today app

The Today app integrates calendar events from your mobile device with your Salesforce tasks, contacts, and accounts.

1. Tap , and then tap the **Today** app.
2. Tap **Get Started**.
3. Choose which calendars you want Today to access, then tap **Save**.



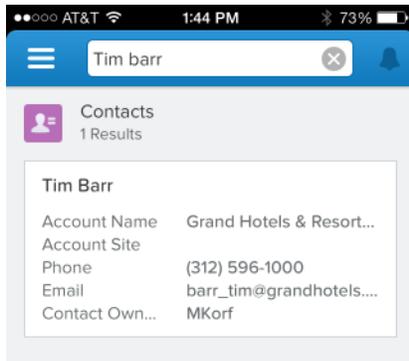
**Note:** You may need to give Salesforce1 access to your calendars in your device's privacy settings before continuing. Once you've given Salesforce1 access to your calendars, return to the Today app and tap **Get Started** again.

4. Tap **Tasks for Today** and notice the task you created for yourself in the previous step. Close the task by tapping the check box.
5. Tap the back arrow to go back to the Today app.

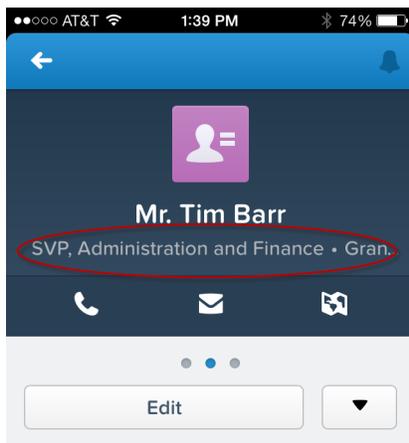
## Step 4: Navigate to a Record

Just as in the full Salesforce site, the record view is where you'll find most of the data you're looking for. In the mobile app, the record view is comprised of three pages. From left to right, these are the record feed page, the record detail page, and the related information page.

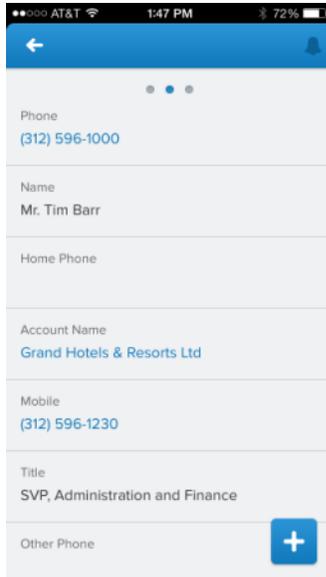
1. Tap the navigation icon , then tap **Contacts**.
2. Enter Tim Barr and then tap Search..
3. Notice how informative the search preview is? These are called *record preview cards*. Your search might have returned multiple preview cards, and so what's on those cards should convey important information at a glance. You can customize record preview cards by creating a *compact layout*.



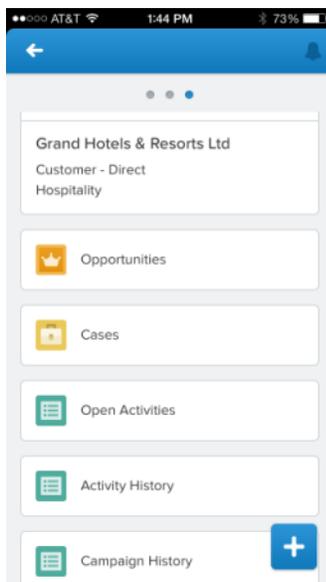
4. Tap the preview card to open the record. The section at the top of the screen is called the *record highlights area*. Notice the three fields under the record name, this is another place in which the compact layout is used. The first three fields of the compact layout determine what is in the record highlights area.



5. Pull down to perform a refresh.
6. Swipe up to dismiss the highlights area and view the fields displayed on a record detail page. These fields are determined by the page layout. As you can see, there are a lot of fields, and you can use the advanced page layout editor to modify all your page layouts to be mobile-friendly. You can also create mobile-specific layouts and then assign them to users who primarily use Salesforce from a mobile device.



7. Swipe left on a record detail page to get to the *record related information* page. (If you're using a mouse, there are three small buttons on top, click the one on the right.) Different objects will have different kinds of related information. Notice in the following image that this account has Opportunities and Cases which are related to this account. In addition, this account has Open Activities, Activity History, and Campaign History you can access with a touch. You can add more related information here by editing the page layout and adding *mobile cards*.



8. Swipe twice to the right and you get to the feed. (Again, if you're using a mouse, click the blue button on the left.) There's nothing to see here yet, so tap **Follow** in case something noteworthy happens with Tim.
9. On Tim Barr's detail page, tap **Edit** and add a new Home Phone for him. Tap **Save**. (Also notice you can **Delete** and **Clone** this record right from the detail page.)

## Step 5: Try a Record Action

Some types of standard objects have built-in *record actions*. Before you get started creating your own actions, it's useful to see what the built-in actions can do.:

1. On Tim Barr's detail page, notice the three *record actions* here. Tap the icon for Call , and notice the new Home Phone number you added.
2. Try some of the other record actions by tapping Email , and Map .
3. Now tap  and you'll see the publisher actions associated with this record. Swipe left and you'll see more actions. These are object-specific actions, and they all have to do with Tim Barr. You can create your own publisher actions in Salesforce. Once you add them to the page layout for an object, they'll show up in its publisher in Salesforce1.
4. Tap **Log a Call**, and then tap **Subject**. Choose **Call** as the type of interaction. .
5. In the Comments section, enter No answer and then tap **Submit**.



**Note:** Log a call actions are special kinds of publisher actions that record interactions with other people. After you create this action, notice that this record is associated directly with Tim Barr. If you don't see a Log a Call action, you're using a Developer Edition org that was created prior to Winter '14. To go any further in these tutorials, you need to get a new Developer Edition organization.

### Tell Me More....

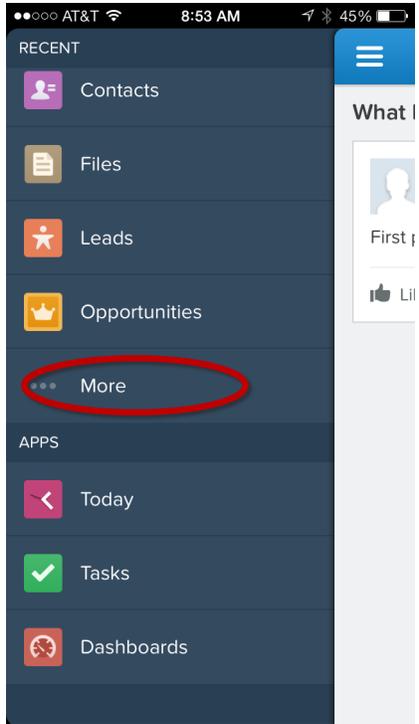
Some of these actions, like **Call** will automatically be updated as more data is added to the record. You saw this already when you created a new phone number. Anyone who accesses this record automatically gets the updates that you entered, and vice versa.

## Step 6: Add a Record to Your App

If you've used Salesforce before, you might be wondering where the Home tab is. Or where Salesforce apps, such as the Sales app or your custom app, appear in the Salesforce1 mobile app. The short answer is, they don't. Instead, the mobile app figures out which *records* you look at most often. Rather than using the Force.com app menu to customize the tabs a user sees regularly, the smart search items under the **Recent** section reorder themselves based on the user's history of recent objects.

To see how this works, navigate to a lesson and create a new record.

1. Tap  and scroll down and tap **More** (or **Show More** if you're on the mobile browser version).



2. Tap **Lessons**.
3. Tap **New** to create a lesson from the mobile device.
4. Name it `Customize mobile layout`, and then **Save**.

### Tell Me More....

Each tab is represented through a menu item in the **Recent** section of the Salesforce1 navigation menu. Searches in the full Salesforce site (not the mobile app) determine what shows up here. Since your app's Lessons tab is new and you probably haven't searched for it, it doesn't appear on the **Recent** section.

## Step 7: Pin Frequently Used Searches

When you first toured the mobile app, you saw how apps and tabs don't work the same as in the full Salesforce site. For example, the content in the Recent section of the Salesforce1 mobile app's navigation menu represents a set of recently searched-for objects.

It would be nice if the Lessons tab showed up at the top of the recent items list, instead of tapping **More** every time you want to find it. To influence the order of items in the Recent section you can *pin* the objects on the Search Results screen in the full site. When you pin an object, it will stick to the top of the Recent section in the Salesforce1 mobile app.

1. In the full site, type `Customize` in the Search box and then click **Search**.



2. In the search results, scroll down, hover to the right of **Lessons**, and then click the pin icon.

**Search Results**

Search Feeds Customize

**Records**

- Accounts (0)
- Activities (0)
- Assets (0)
- Attachments (0)
- Campaigns (0)
- Case Comments (0)
- Cases (0)
- Contacts (0)
- Contracts (0)
- Documents (0)
- Files (0)
- Groups (0)
- Ideas (0)
- Leads (0)
- Lessons (1)**
- Notes (0)
- Opportunities (0)

**Lessons (1)**

Action	Lesson Name
Edit	<a href="#">Customize mobile la</a>

**Solutions (3)**

To further filter these

Solution Title
<a href="#">Maintenance guidelines for generator unclear</a>
<a href="#">Generator assembly instructi unclear</a>
<a href="#">GenWatt Installation Services</a>

Pin Lessons to top

- Go back to your mobile app (you may need to refresh by pulling down) and notice that Lesson is now pinned to the top of the Recent section.

### Tell Me More....

Pinning a search term is an easy way to provide better productivity, but that's really just the beginning. In this next section you'll see how to use page layouts, compact layouts, and mobile cards to optimize the experience for mobile users.

## Tutorial 3: Optimize for the Mobile Display

A well-designed page layout can often be used by both desktop and mobile devices. However, some objects may still have so many fields that viewing the details can be difficult on a mobile screen. Moreover, mobile users often have different jobs and priorities than desktop users, and so it's useful to create a mobile-specific page layout. A mobile-optimized layout can be assigned to different user profiles, so that people who primarily use a phone are assigned the mobile layout, while desktop users would be assigned the standard layout.

In this tutorial you learn how to do the following:

- Modify an existing page layout so that it's optimized for a mobile device — If your users access your app from desktop and mobile devices, then you might want to optimize your page layouts so that they work with various form factors. However, if your users are entirely or mostly mobile, they might find a new mobile-specific layout is more productive.
- Create a compact layout specifically for mobile devices — Compact layouts determine the fields that show up in an object's record highlights area, and an object's record preview cards (mobile cards that display as record items in list views). Compact layouts are a great way to display a record's key fields at a glance.
- Add mobile cards to the related information page — Mobile cards can show lookup information or Visualforce pages.
- Enable notifications — Notifications are a great way to stay on top of what's important to you. If you're using the Salesforce1 downloadable app, you'll receive notifications when someone mentions you in a post, or when you receive an approval request.

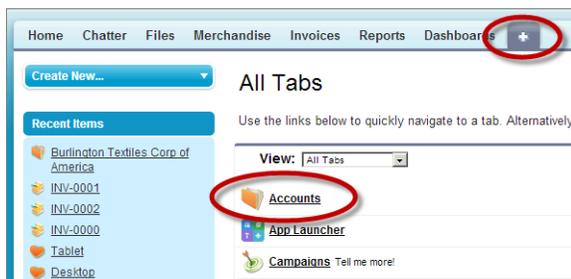


**Note:** There's another kind of mobile layout called a global publisher layout, which determines where global actions go. You'll learn about that later after you create a global action.

### Step 1: Create a Page Layout for a Mobile User Profile

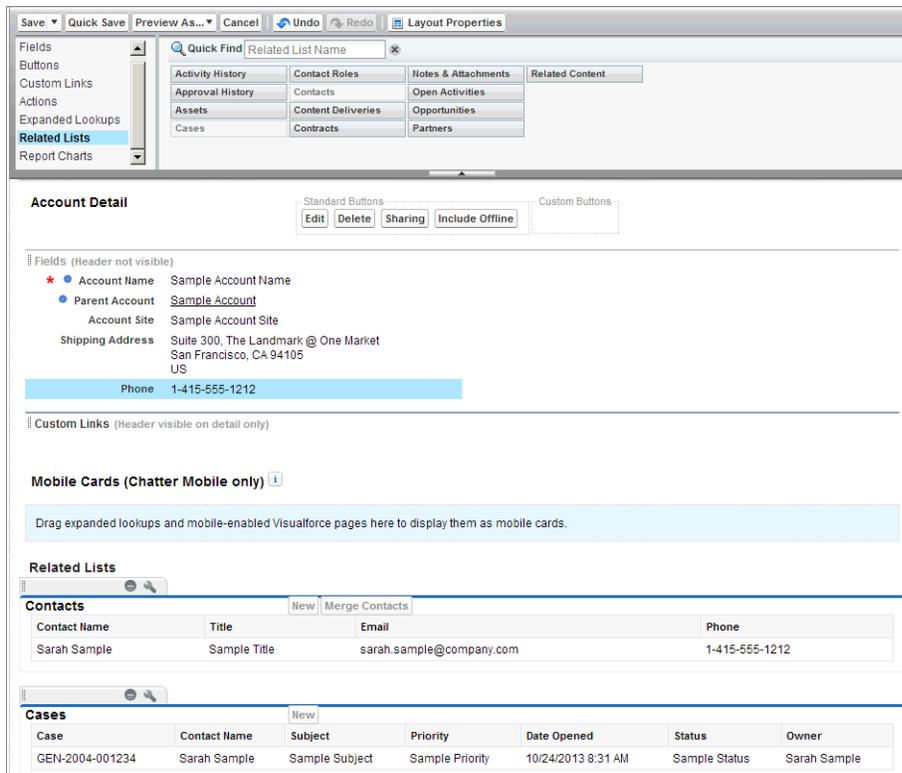
For this step, imagine the regular working day of a mobile technician. The technician is either on site or in a vehicle, and so a mobile phone is his primary means of accessing Salesforce. While working, the technician doesn't need every last detail of the businesses he visits, just the ones that are important to him.

1. On your mobile phone, tap , and then **Accounts** (you may need to tap **More** to find Account).
2. In the Search bar, type `Burl` and then tap **Search**.
3. Tap on the *record preview card* (the search result item) and take a look at the Burlington Textiles account detail page.
4. Scroll down and you'll see there are a lot of fields, really too many for a mobile user to use efficiently. Time to fix all that.
5. In the full Salesforce site, navigate to an existing account by clicking the (+) tab and then **Account**.



6. In the View drop-down list, select **All Accounts**.
7. Click the Burlington Textiles account. Notice that there's a lot of information on this tab, which is why it was a challenge to navigate on a small screen.

8. Click **Setup**, click **Customize > Accounts > Page Layouts** and then click **New**.
9. Name the page layout **Account Mobile Layout** and then **Save**.
10. Add a few fields that are important to mobile users. Drag the **Account Site**, **Shipping Address**, and **Phone** fields onto the **Account Detail** area of the page layout.
11. Click the **Related Lists** category and drag **Cases** and **Contacts** to the **Related Lists** section. Related lists show up on the record related information page in Salesforce1. When a mobile user using this page layout navigates to an account's related information page, they'll see preview cards containing brief information about the cases and contacts for that location.



12. Click **Save** and then **No** when asked if you want to override users' customized related lists.
13. Now you need to assign the mobile-optimized page layout to your user profile. Click **Page Layout Assignment** and then **Edit Assignment**.
14. Click **System Administrator**, which is your user profile.
15. In the Page Layout to Use drop-down list, select **Account Mobile Layout** and then **Save**.

Because you're logged in as the System Administrator, when you access the Account object, you'll do so through the mobile-optimized layout.

1. Try it now by going to the mobile browser app and tap (click) **Accounts** in the sidebar.
2. Since you just accessed the Burlington Textiles account from the full Salesforce site, you should see that in the Recent Accounts list. Tap that account.
3. Notice the fields you customized on the page layout, this is a lot easier to manage.
4. Swipe left to see the related information page and notice the Cases and Contacts you added.

### Tell Me More....

Typically, after creating a page layout for mobile users, you'd add it to a mobile-user's profile. To keep things simple (so that you don't have to log out and switch users to see the new layout), you simply added the page layout to your own System Administrator profile instead.

## Step 2: Display Key Fields Using Compact Layouts

In the previous tutorial you learned how page layouts can be used to optimize a layout for mobile users. However, page layouts aren't the only thing used to help customize how your data appears in a mobile environment. Salesforce1 uses *compact layouts* to display a record's key fields at a glance. You don't need to create compact layouts for Salesforce1, as the system will generate a default compact layout for all standard and custom objects. However, just as you saw with page layouts, a custom compact layout can help your mobile users be even more productive.

1. On your mobile phone, tap , and then **Accounts**.
2. In the Search bar, type `Burl` and then tap **Search**. You've seen this *record preview card* before, but what you might not have known is that the fields shown here are determined by the compact layout.
3. Tap on the record preview card and take a look at the fields on the Burlington Textiles account detail page. The name and the first three fields in the *record highlights area* are also determined by the compact layout.
4. Back in the full Salesforce site, click **Setup > Customize > Accounts**.
5. Click **Create Compact Layouts** and then **New**.
6. In the Label field, enter `Account Compact Layout` and then press the Tab key.
7. Try using some different fields by moving `Account Name`, `Customer Priority`, and `SLA` to the Selected list, and then **Save**.
8. Now you need to set the compact layout as the primary. Click **Compact Layout Assignment**.
9. Click **Edit Assignment** and select the compact layout you just created and then **Save**.
10. Now go to the mobile browser tab and tap an Account record. Refresh the screen by pulling down, and you should see the fields you defined in the record highlights section.

### Tell Me More....

- Compact layouts aren't just for mobile. When accessing Salesforce from a desktop browser, compact layouts determine which fields appear in a Chatter feed when you create a record using an action in the publisher.
- The record name and the first three fields you assign to your compact layout populate the *record highlights* section at the top of each record view. The number of fields that are shown in a compact layout depends on the screen size of the device you are using. You can add more fields to the compact layout, but typically only three will show up on a mobile phone.
- The fields you define on the compact layout also determine what users see in the *record preview cards* that are returned from search results.

## Step 3: Add Mobile Cards to the Record Related Information Page

You've already seen the related information page when you toured the mobile app for the first time. You navigate to the related information page by swiping left on the detail page for a record. You can add other kinds of related information using *mobile cards*. There are two kinds of mobile cards, *related lookup cards* and *Visualforce page cards*.

In this step, you add a related lookup card to the Account object. Account already has a lookup field that's automatically generated, Last Modified By, so in the interest of brevity, you can use that standard field.

1. From Setup, open the page layout for Account by navigating to **Customize > Account** and click **Page Layout**.

2. Click the **Edit** link next to Account Mobile Layout.
3. In the Page Layout Editor, click the **Expanded Lookups** category.
4. Drag **Last Modified By** to the Mobile Cards section and then **Save**.
5. To test it out, go back to your mobile device and look at an account.
6. Swipe left to get to the related information page and you'll see the mobile card you added.

### Tell Me More....

- Once you've enabled a Visualforce page for mobile, you can use the page layout editor to add the pages to the Mobile Cards section in the same way.
- Unlike compact layouts, mobile cards only appear in the Salesforce1 mobile interface.

## Step 4: Enable Notifications

You probably receive alerts on your phone from apps, even when you aren't using that app. These are push notifications. If you're using the Salesforce1 downloadable app, you can enable push notifications and then receive updates when someone mentions you in a post, or when you receive an approval request.

To enable notifications:

1. Switch over to your laptop and click **Setup**.
2. Click **Mobile Administration > Notifications > Settings**.
3. Select both **Enable in-app notifications** and **Enable Push Notifications**.
4. Click **Save**.

If you're in the full Salesforce site, you receive notifications when someone posts to your profile or mentions you in a post. All notifications, including push notifications, show up in the notification tray.

# Tutorial 4: Quickly Create Records Using Global Actions

There are two kinds of publisher actions, global actions and object-specific actions. Global actions let users create records that aren't related to any other record. Object-specific actions are created in the context of another record, and are automatically related to that record.

Another way to think of global actions are things that users want to do quickly, but not necessarily completely. For example, imagine one of your users works at a trade show and meets new people all day long. She needs a way to quickly add someone as a contact without navigating to a record or associating this person with any other information. That's what a global action is for, quick things that users can follow up with later. In fact, there's a built-in *global* action for creating a New Contact.

There's also a built-in *object-specific* action on the Account object. If you were to navigate to an account record and tap , you'd see there's also a New Contact action there. If you create the new contact in the context of this account, the contact is already associated with the account. The names of the actions are the same, but they behave differently when called from different places.

You can include global actions on global publisher layouts, as well as page layouts for any supported object. In this tutorial you add the global action to the global publisher layout.

## Step 1: Create a Global Action

In this step you create a global action that creates new lesson directly from the feed.

1. In Setup, go to **Create > Global Actions**.

Notice there are a number of actions to choose from; you've seen some of these already in the mobile app.

2. Click **New Action**.
3. For Action Type, select **Create a Record**.
4. For Target Object select **Lesson**.
5. For Label, enter Add Lesson.
6. Click **Save**.

After saving, the action layout editor opens. Typically at this point you'd customize the fields that show up here, but there aren't many fields on this object, so it's not necessary yet.

### Tell Me More....

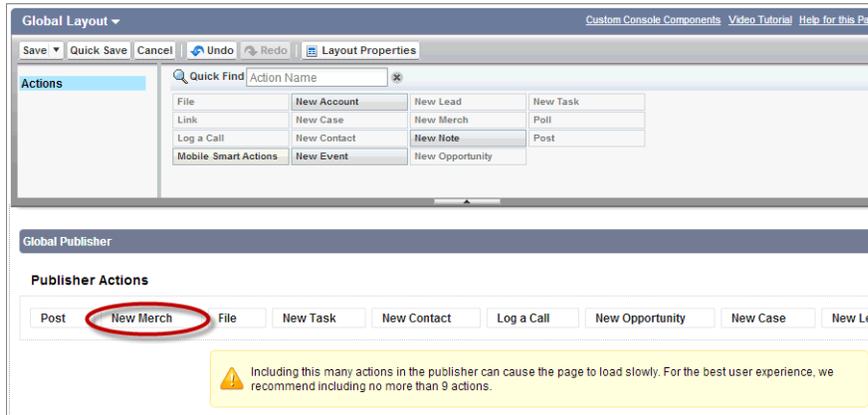
At the bottom of the action layout editor there's a section for predefined values. If you predefine a required field, you don't need to show that field on the page. Predefining fields is also a great way to customize the mobile experience, and you'll learn about that in just a bit.

## Step 2: Customize the Global Publisher Layout

Before the global action will show up either in the full Salesforce site, or in the Salesforce1 mobile app, you need to add it to the global publisher layout.

1. In Setup, go to **Customize > Chatter > Publisher Layouts**.
2. Next to the Global Layout, click **Edit**.

3. In the editor, notice there are a number of items in the Publisher Actions area, such as Post, File, New Task, etc. Drag the **Add Lesson** action into the left side of the Publisher Actions section, between Post and File.



4. There's a warning about the number of actions in the publisher. You can remove some of the actions by dragging them up to the palette. Click **Save**.
5. Now try it out by opening the mobile app. Refresh the app by pulling down, then tap , and then tap the **Feed**.
6. Tap  and you'll see the **Add Lesson** item in the publisher.

### Tell Me More....

- Global actions show up in the publisher on pages to which the global publisher layout applies: in Chatter and in any layout that hasn't been overridden by a more specific publisher layout.
- If you've used page layouts before, you know that page layouts can be assigned to user profiles. The global layout is assigned to all user profiles by default, so you don't need to assign the layout to a profile in order to see the global action you created. But it's useful to know that if you have different users that need different global layouts, it's easy to change right here by clicking **Publisher Layout Assignment**.

# Tutorial 5: Create Related Records with Object-Specific Actions

Object-specific actions let users create records that are automatically associated with related records. This example uses the Account and Case standard objects, which come in every Developer Edition org.

In this example, a mobile technician might want a way to create a new case while still on site with a customer. If you put a record create action on the Account object with Case as the target object, the technicians can browse to the customer account record on their mobile device, and log cases directly from there.

The overall steps for creating an object-specific action are the following:

1. Create the object-specific action.
2. Choose which fields users see. Predefine required field values where possible.
3. Add the action to one or more of that object's page layouts.

## Step 1: Define an Object-Specific Action

For this scenario, you create an invoice that's associated with an existing account.

1. In Setup, go to **Customize > Accounts > Buttons, Links and Actions** and click **New Action**.
2. For Action Type, select **Create a Record**.
3. For Target Object, select **Case**.
4. For Label, enter `Create a Case` and then **Save**.

The action layout editor opens, which is where you can customize the fields assigned to the action.

5. Drag the `Status` field off the layout and back up into the palette and then **Save**.
6. You get a warning message about a required field. Click **Yes**, because you'll fix that next.

### Tell Me More....

You just dragged a required field off the page layout. The platform gives you a warning message, as well it should, because users won't be able to create a case from the mobile action! The reason for removing that field will become clear in the next step, when you predefine that field's value.

## Step 2: Choose Fields and Predefine Values

Objects can have many fields, and so when a user creates a record for that object, it can result in a long list that results in a lot of scrolling. So it's important to choose which fields show up on the action layout. Additionally, you can predefine field values, and then remove them from the action layout.

For this example using a mobile technician, they are already on site logging the case. Rather than require them to choose a Status every time they create a case, you can predefine the field value. Then you can remove the required field from the action layout. Whenever a Create Case action is used, the status will automatically be set.

1. In Setup, click **Customize > Accounts > Buttons, Links, and Actions**.
2. Click the **Create a Case** action you just created.
3. In the Predefined Values related list, click **New**.
4. From the Field Name drop-down list, select **Status**.

5. Set its value to **Working** and then **Save**.

### Tell Me More....

Note that predefined values override default values. In the previous example, imagine that cases created on the full Salesforce site are typically new, and so whenever a case is created there, the default value is set to “Open”. But when a new case is created from a mobile device, it’s because there’s a mobile technician on site, and they are actually working on that case. New cases logged from the mobile device overrides the default value and predefines it as “Working”. As you can see, not only do predefined field values free up screen space, they can also be used to optimize for what people do when they are mobile.

## Step 3: Customize an Object-Specific Layout

Before the action will show up either in the full Salesforce site, or in the Salesforce1 mobile app, it needs to be added to a page layout.

1. In Setup, click **Customize** > **Accounts**, and then click **Page Layouts**.
2. Next to Account Mobile Layout, click **Edit**.

This is the layout you created earlier. Notice that the Publisher Actions section is empty, and there’s a message there telling you that any actions on this layout are being inherited from the global publisher layout. You don’t want that, you want to customize the actions on this layout to be pertinent to the work the mobile users need to do.

3. In the Publisher Actions section, click **override the global publisher layout**.
4. Click the **Actions** category in the palette, then drag **Create a Case** so that it’s the second item in the list.

Notice there’s also a **New Case** item in the palette. The **New Case** item is a default action assigned to the Account object, but it’s not editable. You don’t want this default action, because you created a custom Create a Case action.

5. Click **Save**. The new Create a Case action will now show up in the feed on the Account detail page in the full Salesforce site, and in the publisher for the Salesforce1 mobile app for all profiles with this layout.
6. Now test it on your mobile device by navigating to an account.
7. On the detail page for an account, tap  and tap **Create a Case**.

You don’t see the required `STATUS` field for the case, but it’s there, and so is the association to this particular account.

### Tell Me More....

When you create object-specific layouts, put the most important ones first. Keep in mind that the first six actions in the list show up on the first page of the publisher in the Salesforce1 mobile app.

# Tutorial 6: Develop a Visualforce Page and Add it to the Navigation Menu

The previous lessons used the built-in features of the platform and relied on point-and-click development. From this point on, you'll be using a custom app and writing code. To get you started quickly, there's a pre-built Warehouse app you can install.

In this tutorial, you give mobile technicians that work for the Acme Wireless organization a way to find nearby warehouses. For example, if the technician is out on a call and needs a part, they can use this page to look for warehouses within a 20-mile radius. For each warehouse, a map should display a pin along with the warehouse name, address, and phone number. You can use your knowledge of Visualforce to extend the Salesforce1 app and give your mobile users the functionality they need.

## Prerequisites: Set Up Your Development Environment

The following lessons all use a Warehouse app that is installed from a package. It's a very simple data model, but just enough to illustrate the basic concepts.

- The warehouse has a Merchandise object that represents computer hardware and peripherals: laptops, desktops, tablets, monitors, that kind of thing.
- An invoice is used to keep track of how merchandise moves out of the warehouse. Each line item on the invoice has a particular piece of merchandise, and the number of items ordered. The invoice rolls up all the prices and quantities for an invoice total.

## Step 1: Install the Enhanced Warehouse Data Model

To prepare your developer organization for the exercises in this book, you need to import the Warehouse data model. You might be familiar with the Warehouse app if you've gone through the previous Hands-on Workshop, or from the tutorials in the *Force.com Workbook*. The Warehouse app used here is an enhanced version that will help demonstrate what Salesforce1 mobile app can do.

1. In your browser go to [http://bit.ly/warehouse\\_schema11](http://bit.ly/warehouse_schema11)
2. If you were already logged in, you will be redirected to the Package Installation Details page. Otherwise, log in with your Developer Edition credentials.
3. Click **Continue**, **Next**, **Next**, and **Install**.
4. After the installation finishes, click the Force.com app menu and select **Warehouse**.
5. Click the **Data** tab and then click the **Create Data** button.

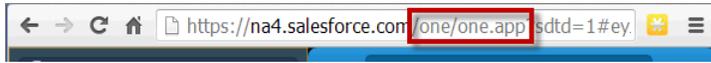
The package contains a pre-built Visualforce page, as well as some supporting resources. You'll learn about those right after your development and testing environments are set up.

## Step 2: Access the Mobile Browser App

When developing Visualforce pages for the Salesforce1 mobile app, you can't do the familiar `https://<instance>/apex/<page>` hack on the URL to view the page: you must view the pages in the mobile app. The best way to test your pages is with the installed app, because it provides the most realistic experience. However, since it's a pain to grab your phone every time you want to see a change, you can open a new browser tab and use the one .app mobile browser version.

1. In your browser, open a new tab.

2. Copy and paste your Salesforce instance into the address bar of the new tab, and add `/one/one.app` to the end. For example, if your Salesforce instance has an URL of `https://na4.salesforce.com`, use `https://na4.salesforce.com/one/one.app`.



You should now see the mobile browser version of Salesforce1. As you go through the exercises in this workbook, you can develop in one tab and then test in the other!



**Note:** The `one/one.app` version is great for development, but you should always test on the actual devices and browsers that you intend to support.

## Lesson 1: Create the FindNearby Apex Class

You can use Apex either as a standalone controller or as extension to existing controller logic. In this case, you use a Standard Controller for the main logic and extend with an Apex class. A Standard Controller is a feature of the platform that provides baseline functionality like create, read, update and delete without having to add additional code.

### Step 1: Create the FindNearby Apex Class

First you need to define the class itself and give it a constructor method. This method will be called when the class is initialized by the Standard Controller, which will pass in a reference to itself.

1. Go to **Setup > Develop > Apex Classes** and click **New**.
2. In the Editor enter the following code

```
global with sharing class FindNearby {  
    public FindNearby(ApexPages.StandardSetController controller) { }  
}
```

3. Click **Quick Save**.

### Step 2: Create the getNearby Method

You need to add some logic which will perform the distance query itself. To do this, you annotate this as a `RemoteAction` method, which will make the functionality easily exposed to JavaScript within Visualforce.

1. In the code editor, under the constructor method, before the bracket that closes the class add the following:

```
@RemoteAction  
// Find warehouses nearest a geolocation  
global static List<Warehouse__c> getNearby(String lat, String lon) {  
  
}
```

## 2. Click **Quick Save**.



**Note:** When you save the file, you get an error: "Error: Compile Error: Non-void method might not return a value or might have statement after a return statement. at line 6 column 68". You can ignore this message for now, it will go away after Step 4.

## Step 3: Add Default Location Logic

If for some reason the latitude or longitude aren't sent to the method, you want to provide a default location. Add a check for incoming lat and lon variables and give them the values if the query was for San Francisco: `FindNearbyWarehousesPage` uses the Google Maps JavaScript API v3 to plot the nearby warehouses on a map. The map is resized based on the records returned by the SOQL query and then each record is plotted as a marker on the map.

### 1. Within the `getNearby` method (about line 9), add the following:

```
// If geolocation isn't set, use San Francisco
if(lat == null || lon == null || lat.equals('') || lon.equals('')) {
    lat = '37.77493';
    lon = '-122.419416';
}
```

## 2. Click **Quick Save**.

## Step 4: Run a Query and Return Results

We have a stub for our method and given it a default set of values for lat and long, now we should call the database itself to see if there is anything nearby. We will dynamically put the query together based on our information and then use the geolocation feature of SOQL to be able to find results:

### 1. Under the default values if statement (about line 15), add the following:

```
// SOQL query to get the nearest warehouses
String queryString =
    'SELECT Id, Name, Location__Longitude__s, Location__Latitude__s, ' +
    'Street_Address__c, Phone__c, City__c ' +
    'FROM Warehouse__c ' +
    'WHERE DISTANCE(Location__c, GEOLOCATION('+lat+', '+lon+'), \'mi\') < 20 ' +
    'ORDER BY DISTANCE(Location__c, GEOLOCATION('+lat+', '+lon+'), \'mi\') ' +
    'LIMIT 10';

// Run and return the query results
return(database.Query(queryString));
```

## 2. Click **Quick Save**.

## Summary: Check Completed Code

Your completed class should look like the following:

```
global with sharing class FindNearby {

    public FindNearby(ApexPages.StandardSetController controller) { }
```

```

@RemoteAction
// Find warehouses nearest a geolocation
global static List<Warehouse__c> getNearby(String lat, String lon) {

    // If geolocation isn't set, use San Francisco
    if(lat == null || lon == null || lat.equals('') || lon.equals('')) {
        lat = '37.77493';
        lon = '-122.419416';
    }

    // SQL query to get the nearest warehouses
    String queryString =
        'SELECT Id, Name, Location__Longitude__s, Location__Latitude__s, ' +
        'Street_Address__c, Phone__c, City__c ' +
        'FROM Warehouse__c ' +
        'WHERE DISTANCE(Location__c, GEOLOCATION('+lat+', '+lon+'), \'mi\') < 20 ' +
        'ORDER BY DISTANCE(Location__c, GEOLOCATION('+lat+', '+lon+'), \'mi\') ' +
        'LIMIT 10';

    // Run and return the query results
    return(database.Query(queryString));
}
}

```

## Lesson 2: Create the Visualforce Page

You now have an Apex extension that will return Warehouses that are close to an existing latitude and longitude. Now you need an interface for the user to call that query and display results. There are many ways you could build this UI, but to make a mobile-friendly and dynamic page you are going to use the Google Maps API. The JavaScript required to access the API and render maps has already been included in the Enhanced Warehouse as a static resource.

### Step 1: Bind the extension and Standard Controller to a Visualforce Page

The first thing you need to do is create a blank page and then associate it with the server-side controller logic. As noted in the previous step, this will be a Standard Controller and an extension to perform the geolocation search.

1. From Setup, click **Develop > Pages**.
2. Click **New**.
3. For the Label and Name enter `FindNearbyWarehousesPage`.
4. Select the checkbox for **Available for Salesforce mobile apps**.
5. In the code editor, enter:

```

<apex:page sidebar="false" showheader="false" standardController="Warehouse__c"
recordSetVar="warehouses" extensions="FindNearby">

</apex:page>

```

6. Click **Quick Save**.

## Step 2: Add Static Resources to the Page

You defined the core page element, but before you start writing any JavaScript you will need to have a reference to libraries you will use. These are stored as static resources in the system and can be associated with the page using the `includeScript` component. This component will make sure that the JavaScript is included in the rendered HTML's header properly. You are also going to add a small amount of CSS to the page in order to show the map in the correct dimensions.

1. Inside the page component (around line 3), enter the following code:
2. In the code editor, enter:

```
<!-- Include in Google's Maps API via JavaScript static resource -->
<apex:includeScript value="{!$Resource.googleMapsAPI}" />

<!-- Set this API key to fix JavaScript errors in production -->

<!--http://salesforcesolutions.blogspot.com/2013/01/integration-of-salesforcecom-and-google.html-->

<!--<script type="text/javascript"
src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&sensor=false">
</script-->

<!-- Setup the map to take up the whole window -->
<style>
html, body { height: 100%; }
.page-map, .ui-content, #map-canvas { width: 100%; height:100%; padding: 0; }
#map-canvas { height: min-height: 100%; }
</style>
```

3. Click **Quick Save**.

## Step 3: Place a Container `div` for Rendering the Map

You are about ready to write the JavaScript to show the map, but the map will need an HTML div to render the graphics. Add that towards the end of the page:

1. Before the end page tag (around line 20), enter the following code:

```
<!-- All content is rendered by the Google Maps code -->
<!-- This minimal HTML just provide a target for GMaps to write to -->
<body style="font-family: Arial; border: 0 none;"
<div id="map-canvas"</div>
</body>
```

2. Click **Quick Save**.

## Step 4: Add the initialize JavaScript function

Now our page is ready for some JavaScript to make it work. We will start with a function that we're going to use when the page loads. This function will be responsible for calling the Apex method that we created earlier and get the list of warehouses to display.

1. Under the style tag (around line 19), add the following code:

```
<script>
  function initialize() {
    var lat, lon;

    // Get the position of the user via device geolocation
    if (navigator.geolocation) {
      navigator.geolocation.getCurrentPosition(function(position) {
        lat = position.coords.latitude;
        lon = position.coords.longitude;

        // Use Visualforce JavaScript Remoting to query for nearby warehouses
        Visualforce.remoting.Manager.invokeAction('{!$RemoteAction.FindNearby.getNearby}',
        lat, lon,
          function(result, event){
            if (event.status) {
              console.log(result);
              createMap(lat, lon, result);
            } else if (event.type === 'exception') {
              //exception case code
            } else {

            }
          },
          {escape: true}
        );
      });
    } else {
      // Set default values for map if the device doesn't have geo
      /** San Francisco */
      lat = 37.77493;
      lon = -122.419416;

      var result = [];
      createMap(lat, lon, result);
    }
  }
</script>
```

2. Click **Quick Save**.

## Step 5: Add the createMap function

Now we have some values of nearby Warehouses, the interface should be able to generate the map from Google. You can see in the code from the previous step that there are a few references to a createMap function. Since we don't have that right, the page won't work. Let us go ahead and add that.

1. Before the end script tag (around line 55), add the following code:

```
function createMap(lat, lon, warehouses){
  // Get the map div, and center the map at the proper geolocation
  var currentPosition = new google.maps.LatLng(lat,lon);
  var mapDiv = document.getElementById('map-canvas');
  var map = new google.maps.Map(mapDiv, {
    center: currentPosition,
    zoom: 13,
    mapTypeId: google.maps.MapTypeId.ROADMAP
  });
}
```

```

// Set a marker for the current location
var positionMarker = new google.maps.Marker({
  map: map,
  position: currentPosition,
  icon: 'http://maps.google.com/mapfiles/ms/micons/green.png'
});

// Keep track of the map boundary that holds all markers
var mapBoundary = new google.maps.LatLngBounds();
mapBoundary.extend(currentPosition);

// Set markers on the map from the @RemoteAction results
var warehouse;
for(var i=0; i<warehouses.length;i++){
  warehouse = warehouses[i];
  console.log(warehouses[i]);
  setupMarker();
}

// Resize map to neatly fit all of the markers
map.fitBounds(mapBoundary);
}

```

2. Click **Quick Save**.

## Step 6: Create Markers for Nearby Warehouses

The page is nearly complete. Our JavaScript is calling into Apex, getting a list of nearby warehouses, and then using Google to create a map of our current location. Now we just need to map the results to actual markers we can place on the map. Once again, we see a reference towards the end of the last code snippet which refers to a `setupMarker` function being called while iterating through our found warehouses. Here is the code for that function.

1. Under the `fitBounds` function call and before the end bracket (around line 88), add the following:

```

function setupMarker(){
  var warehouseNavUrl;

  // Determine if we are in Salesforce and set navigation link appropriately
  try{
    if(sforce.one){
      warehouseNavUrl =
        'javascript:sforce.one.navigateToSObject('\'' + warehouse.Id
+ '\')';
    }
  } catch(err) {
    console.log(err);
    warehouseNavUrl = '\'' + warehouse.Id;
  }

  var warehouseDetails =
    '<a href="' + warehouseNavUrl + '"'>' +
    warehouse.Name + '</a><br/>' +
    warehouse.Street_Address__c + '<br/>' +
    warehouse.City__c + '<br/>' +
    warehouse.Phone__c;

  // Create the callout that will pop up on the marker
  var infowindow = new google.maps.InfoWindow({

```

```

        content: warehouseDetails
    });

    // Place the marker on the map
    var marker = new google.maps.Marker({
        map: map,
        position: new google.maps.LatLng(
            warehouse.Location__Latitude__s,
            warehouse.Location__Longitude__s)
    });
    mapBoundary.extend(marker.getPosition());

    // Add the action to open up the panel when it's marker is clicked

    google.maps.event.addListener(marker, 'click', function(){
        infowindow.open(map, marker);
    });
}

```

- Now add the following code below that method so that the JavaScript will run when the page loads:

```

// Fire the initialize function when the window loads
google.maps.event.addDomListener(window, 'load', initialize);

```

- Click **Quick Save**.

## Step 7: Check the Final Page

You should be able to test the page now by going to your instance URL in your browser (for example, <https://na15.salesforce.com/>) and appending `/apex/FindNearbyWarehousesPage`. The final page is a lot of JavaScript and a small bit of HTML. Here is the entire page if you are not seeing a Google Map in the final version:

```

<apex:page sidebar="false" showheader="false" standardController="Warehouse__c"
recordSetVar="warehouses" extensions="FindNearby">

    <!-- Include in Google's Maps API via JavaScript static resource -->
    <apex:includeScript value="{!$Resource.googleMapsAPI}" />

    <!-- Set this API key to fix JavaScript errors in production -->

<!--http://salesforcesolutions.blogspot.com/2013/01/integration-of-salesforcecom-and-google.html-->

    <!--<script type="text/javascript"
        src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&sensor=false">
    </script-->

    <!-- Setup the map to take up the whole window -->
    <style>
        html, body { height: 100%; }
        .page-map, .ui-content, #map-canvas { width: 100%; height:100%; padding: 0; }
        #map-canvas { height: min-height: 100%; }
    </style>

    <script>
        function initialize() {
            var lat, lon;

            // If we can, get the position of the user via device geolocation
            if (navigator.geolocation) {
                navigator.geolocation.getCurrentPosition(function(position) {
                    lat = position.coords.latitude;
                    lon = position.coords.longitude;

```

```

        // Use Visualforce JavaScript Remoting to query for nearby warehouses

Visualforce.remoting.Manager.invokeAction('{!$RemoteAction.FindNearby.getNearby}', lat,
lon,
        function(result, event){
            if (event.status) {
                console.log(result);
                createMap(lat, lon, result);
            } else if (event.type === 'exception') {
                //exception case code
            } else {

            }
        },
        {escape: true}
    );
});
} else {
    // Set default values for map if the device doesn't have geolocation
capabilities
    /** San Francisco */
    lat = 37.77493;
    lon = -122.419416;

    var result = [];
    createMap(lat, lon, result);
}

}

function createMap(lat, lon, warehouses){
    // Get the map div, and center the map at the proper geolocation
    var currentPosition = new google.maps.LatLng(lat,lon);
    var mapDiv = document.getElementById('map-canvas');
    var map = new google.maps.Map(mapDiv, {
        center: currentPosition,
        zoom: 13,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    });

    // Set a marker for the current location
    var positionMarker = new google.maps.Marker({
        map: map,
        position: currentPosition,
        icon: 'http://maps.google.com/mapfiles/ms/micons/green.png'
    });

    // Keep track of the map boundary that holds all markers
    var mapBoundary = new google.maps.LatLngBounds();
    mapBoundary.extend(currentPosition);

    // Set markers on the map from the @RemoteAction results
    var warehouse;
    for(var i=0; i<warehouses.length;i++){
        warehouse = warehouses[i];
        console.log(warehouses[i]);
        setupMarker();
    }

    // Resize map to neatly fit all of the markers
    map.fitBounds(mapBoundary);

    function setupMarker(){
        var warehouseNavUrl;

        // Determine if we are in Salesforce1 and set navigation link appropriately

```

```

        try{
            if(sforce.one){
                warehouseNavUrl =
                    'javascript:sforce.one.navigateToSObject(\'\' + warehouse.Id +
                    '\')';
            }
        } catch(err) {
            console.log(err);
            warehouseNavUrl = '\\\' + warehouse.Id;
        }

        var warehouseDetails =
            '<a href=\'\' + warehouseNavUrl + \'\'>' +
            warehouse.Name + '</a><br/>' +
            warehouse.Street_Address__c + '<br/>' +
            warehouse.City__c + '<br/>' +
            warehouse.Phone__c;

        // Create the callout that will pop up on the marker
        var infowindow = new google.maps.InfoWindow({
            content: warehouseDetails
        });

        // Place the marker on the map
        var marker = new google.maps.Marker({
            map: map,
            position: new google.maps.LatLng(
                warehouse.Location__Latitude__s,
                warehouse.Location__Longitude__s
            )
        });
        mapBoundary.extend(marker.getPosition());

        // Add the action to open up the panel when it's marker is clicked
        google.maps.event.addListener(marker, 'click', function(){
            infowindow.open(map, marker);
        });
    }
}

// Fire the initialize function when the window loads
google.maps.event.addDomListener(window, 'load', initialize);

</script>

<!-- All content is rendered by the Google Maps code -->
<!-- This minimal HTML justs provide a target for GMaps to write to -->
<body style="font-family: Arial; border: 0 none;">
    <div id="map-canvas"></div>
</body>
</apex:page>

```

## Lesson 3: Expose the Page in Salesforce1

Now that the page is complete, you can add it to the mobile app. In order to do that, you first need to create a tab and then you can add the tab to the mobile navigation menu.

### Step 1: Create a Tab

Start by creating a tab.

1. From Setup, click **Create > Tabs**.

2. In the Visualforce Tabs section, click **New**.
3. In the Visualforce Page drop-down list, select **FindNearbyWarehousesPage**.
4. In the Tab Label field, enter `Find Nearby Warehouses`.

The label field is what users see both on the full site and the mobile app. With that in mind, keep your labels no longer than this.

5. Click into the Tab Style field, and select the **Globe** style.

The icon for this style appears as the icon for the page in the Salesforce1 mobile app's navigation menu.

6. Click **Next**, and **Next** again.
7. Deselect the **Include Tab** checkbox so that the tab isn't included in any of the apps in the full site. You only want this tab to appear when users are viewing it on their mobile device.
8. Click **Save**.

Now that you've created the Visualforce page and the tab, you're ready to add the new tab to the navigation menu.

## Step 2: Add the Tab to Mobile Navigation

In this step you add the tab as a navigation menu item in the Salesforce1 mobile app. The menu item will instantly become available to mobile app users that have access to it.

1. From Setup, click **Mobile Administration** > **Mobile Navigation**.
2. Move **Find Nearby Warehouses** to the Selected list and then **Save**.

## Step 3: Try Out the App

Now you can search nearby warehouses on your device.

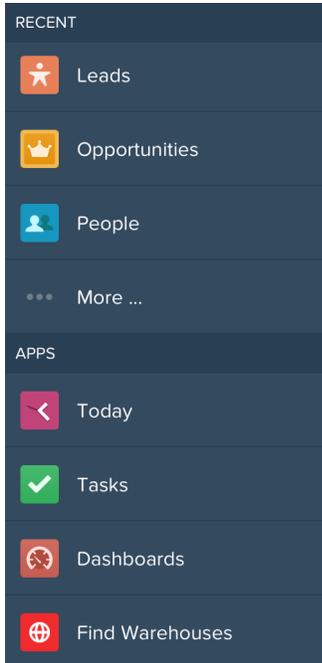
1. Open the Salesforce1 app on your mobile device. Refresh the app by pulling down.
2. Tap  to access the navigation menu. You should see Find Nearby Warehouses under the Apps section.



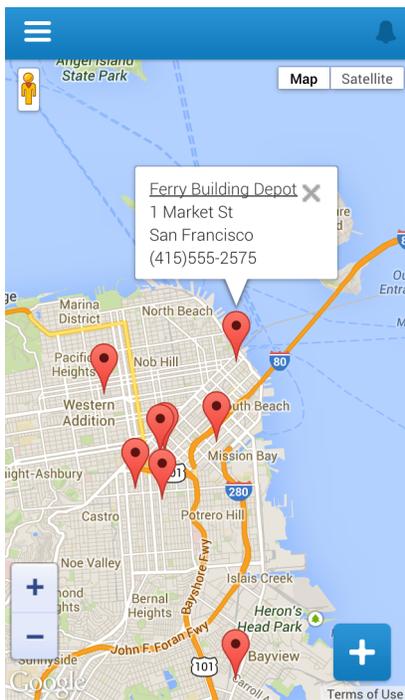
**Note:**

- If you're using the `one/one.app` browser version, you may need to refresh the browser to see the page in the navigation menu.
- If you're using the installed mobile app, you may need to log out and log in again to see the change.

3. Tap **Find Nearby Warehouses**.



4. Click **OK** when you see a prompt that asks to use your current location. A map that contains all the nearby warehouse locations within 20 miles appears.



**Note:** If you don't receive a prompt, this may be related to your device settings. If that's the case, the geographical area should default to San Francisco.

The warehouses in the package sample data are all located in the San Francisco area. If you're testing this from another location, be sure to add a warehouse located within 20 miles. That's it! You can see how easy it is to make standard pages and tabs available to your mobile users. For more information about development guidelines for Visualforce pages, see Visualforce Guidelines and Best Practices.

[https://developer.salesforce.com/docs/atlas.en-us.salesforce1.meta/salesforce1/vf\\_dev\\_best\\_practices.htm](https://developer.salesforce.com/docs/atlas.en-us.salesforce1.meta/salesforce1/vf_dev_best_practices.htm)