salesforce 1 Platform

Salesforce1 Developer Cheat Sheet

Overview

Salesforce1 is a mobile app development platform for everyone. Designed for scale with open APIs for extensibility and integration, and powerful developer tools, there's no limit to what you can build on the platform. Salesforce1's flexible development models enable every user to create custom apps backed by mobile back-end services and a unique, yet familiar, mobile user experience. ISVs can develop apps that take advantage of advanced packaging and version management controls, complete enterprise marketplace capabilities with the AppExchange, and feed-first discovery of their apps within the Salesforce1 Platform.

Installation, Administration, and Declarative Development

For installation steps, mobile app administration, and development using using point-and-click tools, see the Salesforce1 Mobile Admin Cheat Sheet or the Salesforce1 App Admin Guide.

Visualforce Pages in Mobile

You can access Visualforce pages from the following areas of the mobile app; each area is optimized for a specific purpose:

- **Navigation Menu** — For pages that aren't related to other objects or that require a full screen, add the page to the navigation menu, or as a global action in the publisher.
- **Record Page** — Pages that appear within the context of a record should go on the object's page layout.
- **Related Items** — If you want a Visualforce page to appear on the related information page of an object record, add the page as a mobile card. Mobile cards are mobile only; they don't appear in the full Salesforce site.
- **Global or Object-Specific Publisher Actions** — If you want a Visualforce page to surface in the publisher, you can create it as a global or object-specific action. Custom actions aren't specific to mobile users, and appear in both the Salesforce1 app and in the full Salesforce site. In the Salesforce1 app, custom actions appear in the publisher. In the full Salesforce site, custom actions appear in the publisher menu.

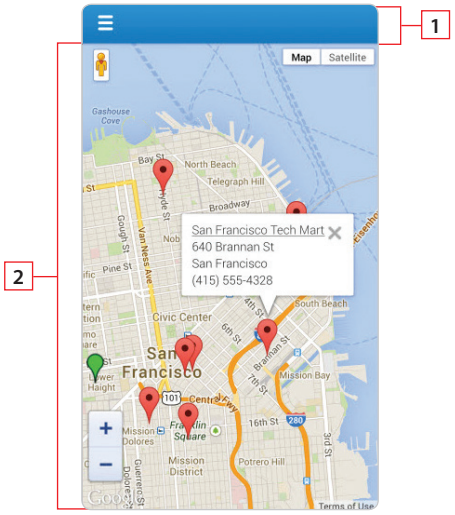
These options are explained in more detail in the following sections

Visualforce Pages in the Publisher

When displayed in the Salesforce1 app, the standard Salesforce header and sidebar are automatically removed. However, Visualforce pages used as custom actions in the publisher are shared with the full Salesforce site, and pages added to the Salesforce1 navigation may or may not be shared. Pages shared with the full site shouldn't have these items explicitly removed, unless that's the standard practice for all Visualforce pages in your organization.

1. The Salesforce1 header, which provides access to the main Salesforce1 menu, is 42 pixels tall. The contents of the header can't be changed.
2. The rest of the device screen is dedicated to the Visualforce page.

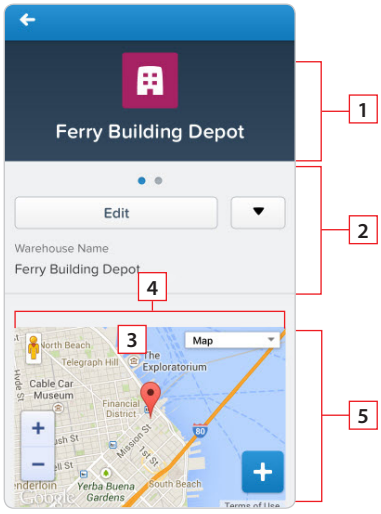
Note: You must be sure to select the **Available for Salesforce mobile apps** checkbox. This designates that the page is mobile-ready and can be used in Salesforce1.



Visualforce Pages on Records

Visualforce pages added to an object's page layout display on the record details page. Unlike mobile cards, you can control the Visualforce element's placement on the Salesforce1 record details page, putting fields and other record details above and below it, by changing its placement on the object's page layout. Visualforce pages added this way follow the same rules for ordering that fields and other elements do.

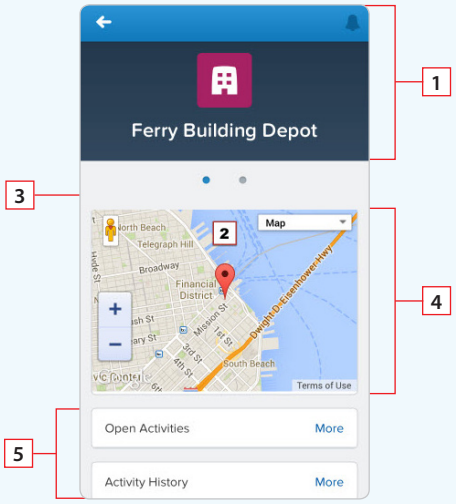
1. The record highlights header displays when a record is loaded, but can be scrolled up and off the screen by the user. When on screen, it's 158 pixels tall on all devices, and takes the full width of the screen. You can't control the display of the record highlights.
2. Record actions and details, automatically generated by Salesforce1.
3. A Visualforce page added to the object's page layout.
4. Set the width to 100%; the element sizes automatically, minus some padding on either side.
5. Control the height of the Visualforce page's area by setting the height of the item in pixels in the page layout editor. The Visualforce element uses exactly that height, even if the content is shorter. In that case, the extra area is blank. If the page's content is taller, the content is clipped. As a best practice, don't set inline Visualforce pages to be taller than the smallest device screen you intend to support.



Visualforce Pages on Record Related Items

Mobile cards appear on the Record Information page of a record. It's a best practice to add only one or two cards, since users have to scroll past them to get to the other related items. If you need a full screen to display your page, consider moving it to a custom action on the object instead.

1. The record highlights header displays when a record is loaded, but can be scrolled up and off the screen by the user. When on screen, it's 158 pixels tall on all devices, and takes the full width of the screen. You can't control the display of the record header.
2. Mobile cards display above all of the related items on the record.
3. Set the width to 100%; the element sizes automatically, minus some padding on either side. The content of mobile cards can't be scrolled, so make sure it fits in the space you provide it.
4. Control the height of the mobile card by setting the height in pixels in the page layout editor. The mobile card area uses exactly that height, even if the mobile card's content is shorter. In that case, the extra area is blank. If the card's content is taller, the content is clipped. As a best practice, don't create mobile cards taller than the smallest device screen you intend to support. Be sure to set the height of screen elements relevant to your environment. So if you have an element that's always 300 pixels high, set the height to 300 pixels.
5. The record's related items are displayed after all mobile cards.



Creating a Visualforce Custom Action

Creating a Visualforce custom action is the same process as global actions or object-specific actions described on the Salesforce1 Admin Cheat Sheet. To use a Visualforce page in a global or object-specific action, select Custom Visualforce for `Action Type`.

Note, when making an object-specific Visualforce publisher action, you need to set the Standard Controller of your page to the SObject you want to create the action on.

Styling a Mobile Visualforce Page for Salesforce1

Use Salesforce Mobile Design Templates as a starting point for your UI/UX design to dramatically shorten the time it takes to develop a great looking web/hybrid app on the Salesforce Platform. Combine these open-source modular design templates with Salesforce1 Platform Mobile Services and your mobile app to view, edit and update customer data, view backend reports, find nearby records, and more.

The quickest way to get started with the Mobile Design Templates is via their interactive home page. As you scroll down the page, you'll see each template broken down into its constituent HTML5/CSS3 components.

`http://bit.ly/sfdcmobiletemplates`

Managing Navigation with the `sforce.one` Object

The Salesforce1 Platform includes a strict event mechanism for navigation. This is exposed in Visualforce as a JavaScript object called `sforce.one`. It's available in any Visualforce page that appears in Salesforce1.

`sforce.one.navigateToSObject (recordId,view)`	Navigates to the record home of a particular SObject specified by `recordId`. `view` is optional and can specify the view within record home to select - `"chatter", "related", "detail"`
`sforce.one. navigateToURL(url, isredirect)`	Navigates to the specified URL. Relative urls will retain navigation history, absolute urls will open a new browser window. `isredirect` indicates a history entry that should not be created
`sforce.one. navigateToFeed(subjectId, type)`	Navigates to the specific feed `subjectId`. `type` is the expected feed type, for example `"NEWS"`
`sforce.one.navigateToFeedItemDetail(feedItemID)`	Triggers a navigation to the specific feed item
`sforce.one.navigateToRelatedList(relatedListId, parentRecordId)`	Triggers a navigation to the specific related list
`sforce.one. navigateToList(listViewId, listViewName, scope)`	Triggers a navigation to the specific list view
`sforce.one. createRecord(entityName, recordTypeId)`	Triggers the record create workflow for the specified `entityName` and optional `recordTypeId`
`sforce.one. editRecord(recordId)`	Triggers the record edit workflow for the specified `recordId`
`sforce.one.back(refresh)...`	Goes back to the previous state. Pass `true` to indicate the component should refresh if possible.

Define an HTML5 Visualforce Page

Use the `docType="html-5.0"` attribute in the `<apex:page>` tag to define the structure of the rendered page. If html-5.0 is specified, you can utilize HTML5 browser features like tags, JavaScript APIs (like drag and drop, local storage, and geolocation), and the useful Visualforce features as well. This will also relax the default HTML tidying in Visualforce for HTML5 applications.

When building a custom style mobile page, set `standardStylesheets="false"` on the `<apex:page>` tag to eliminate any conflicts with any the standard Salesforce stylesheets are added to the generated page header if the showHeader attribute is set to false. Use the `applyHtmlTag` and applyBodyTag attributes of the `<apex:page>` tag to suppress the automatic generation of `<html>` and `<body>` tags, in favor of static markup you add to the page yourself.

Offline Caching Using the HTML5 Manifest Attribute

Use the manifest attribute of the `<apex:page>` tag to set an HTML5 cache manifest for offline caching of a page's critical resources. You can use Visualforce to provide a page's cache manifest.

The value of the `manifest` attribute is passed through to the generated HTML. For example:

```
<apex:page showHeader="false" sidebar="false"
standardStylesheets="false"
docType="html-5.0" manifest="/apex/CacheManifest">
        <header>
            <h1>Congratulations!</h1>
        </header>
        <article>
            <p>This page looks almost like HTML5!</p>
        </article>
</apex:page>
```

Setting Custom HTML Attributes on Visualforce Components

You can add arbitrary attributes to many Visualforce components that will be "passed through" to the rendered HTML. This is useful, for example, when using Visualforce with JavaScript frameworks, such as jQuery Mobile, AngularJS, and Knockout, which use `data-*` or other attributes as hooks to activate framework functions. It can also be used to improve usability with HTML5 features such as `placeholder` "ghost" text, `pattern` client-side validation, and `title` help text attributes.

To add a pass-through attribute to, for example, an `<apex:outputPanel>` component, prefix the attribute with `"html-"` and set the attribute value as normal.

```
<apex:page showHeader="false" standardStylesheets="false"
doctype="html-5.0">
    <apex:outputPanel layout="block" html-data-role="panel"
    html-data-id="menu">
        <apex:insert name="menu"/>
    </apex:outputPanel>
    <apex:outputPanel layout="block" html-data-role="panel"
    html-data-id="main">
        <apex:insert name="main"/>
    </apex:outputPanel>
</apex:page>
```

Check the *Visualforce Developer's Guide* to see all of the Visualforce components that support pass-through attributes.

User Input and Interaction

Use `<apex:input>`, the type attribute, and pass-through HTML attributes to create mobile-friendly forms and user interfaces.

While you can use `<apex:inputField>` to create an HTML input element for a value that corresponds to a field on a Salesforce object, this is not the most efficient component to use over a mobile wireless connection. However, the benefit of `<apex:inputField>` is the built-in client-side and server-side validation.

Set the `type` Attribute for Input Widgets

Set the type attribute on input components to display UI widgets that help users enter data. Each input adapts for the type of data expected: text fields show a keyboard, date fields show a date picker, etc. For example:

```
<apex:form >
    <apex:outputLabel value="Phone" for="phone"/>
    <apex:input id="phone" value="{!fPhone}" type="tel"/><br/>

    <apex:outputLabel value="Email" for="email"/>
    <apex:input id="email" value="{!fText}" type="email"/><br/>

    <apex:outputLabel value="That Number" for="num"/>
    <apex:input id="num" value="{!fNumber}" type="number"/><br/>

    <apex:outputLabel value="The Big Day" for="date"/>
    <apex:input id="date" value="{!fDate}" type="date"/><br/>
</apex:form>
```

Input type

You can also set `type` to auto, and the data `type` of the associated controller property or method is used. You can also explicitly set the following input types:

- date
- datetime
- datetime-local
- month
- week
- time
- email
- number
- range
- search
- tel
- text
- url

Client-Side Validation

Set HTML5 pass-through attributes to avoid sending a request to the server and waiting for a response. To enable client-side validation, set an `html-pattern` attribute on the `<apex:input>` tag to match expected form values.

If the input matches the regular expression validation pattern, the input is considered valid. If it doesn't match, an error message displays and the form isn't submitted to the server. The following example requires an email address from a specific domain. It uses the placeholder attribute to display sample ghost text.

```
<apex:input id="email" value="{!fText}" type="email"
    html-placeholder="you@example.com"
    html-pattern="^[a-zA-Z0-9._-]+@example.com$"
    html-title="Please enter an example.com email address"/>
```

You can also provide auto-suggestion by using `<apex:input.list>` to generate an HTML5 `<datalist>`. For example, the following will provide options based on what the user types in the box: `<apex:input.list="Aaa,Aabb,Aaacc">`

Navigation

Salesforce1 provides a framework for handling various navigation controls and events, that isn't available to Visualforce pages when they run on the full Salesforce site. For pages shared between the Salesforce1 app and the full site, use the `sforce` object when it's available, and standard Visualforce navigation when it's not. The following example runs after a JavaScript remoting request successfully returns from the `@RemoteAction` method that creates a quick order. This code is from a Visualforce page that's used as a custom action, which adds it to the publisher in the Salesforce1 app and the publisher menu in the full site. The intent of the code is to navigate to the detail page for the account for whom the order was placed, and it needs to work in both places:

```
// Go back to the Account detail page
if( (typeof sforce != 'undefined') && (sforce != null) ) {
    // Salesforce1 navigation
    sforce.one.navigateToSObject(aId);
}
else {
    // Set the window's URL using a Visualforce expression
    window.location.href = '{!URLFOR($Action.Account.View, account.
Id)}';
```

Publisher Events

When you expose a Visualforce page or canvas app in the publisher, you can use well-defined events to enable communication between the page and the publisher as well as utilize the **Submit** button to submit the form, close the publisher, and post to the feed.

For Visualforce Include:

```
 <script type='text/javascript' src='/canvas/sdk/js/29.0/
publisher.js'></script>
```

For Force.com Canvas Include:

```
 <script type='text/javascript' src='/canvas/sdk/js/29.0/
canvas-all.js'></script>
```

The publisher event methods can be used both by the Visualforce and Canvas. Not all publisher event methods are available to Canvas and Visualforce, and the availability of these events are outlined below.

Note: When viewing a Visualforce publisher action outside of Salesforce1, the Submit button is no longer visible automatically. Use the navigation example above to distinguish whether or not you are viewing the action from Salesforce1 and create a submit button to fire the publisher.close event on submit.

Visualforce	Canvas	Field	Description
X	X	`publisher.clearPanelState`	Fired by the publisher when the canvas app/page is deactivated or hidden. This can happen when the user selects a different application in the publisher or after the Share/Submit button has been clicked.
X		`publisher.close`	Fired by the page to tell the publisher to close. Pair this action with a publisher.post to submit and close using the Submit button on the publisher.

Publisher Events continued.

Visualforce	Canvas	Field	Description
	X	`publisher.failure`	Fired by the publisher when an error condition is encountered such as when invalid data has been submitted. For example: • The text in the feed is too long • The page you're attempting to publish to the feed doesn't exist • The canvas app URL is invalid The canvas app should listen for this event and alert the user that an error occurred and the post didn't get created.
	X	`publisher.getPayload`	Fired by the publisher when the Share button is clicked. The payload contains information such as the text entered into the What are you working on? field and who the feed item is being shared with.
X		`publisher.post`	Fired by the publisher to indicate to the page that the Submit button has been pressed.
X		`publisher.refresh`	Fired by the page to refresh the feed.
X	X	`publisher.setupPanel`	Fired by the publisher when the feed is initially loaded.
	X	`publisher.setPayload`	Fired by the canvas app to indicate to the publisher that the content being sent to the publisher should be shared in the feed item. This event is in response to `publisher.getPayload` and contains information about the feed item you're trying to create. You can create three feed item types: • TextPost • LinkPost • CanvasPost
X	X	`publisher.setValidForSubmit`	Fired by the canvas app/page to indicate to the publisher that the canvas app/page is ready to submit a payload. After this event fires, the Share/Submit button becomes active. This code snippet enables the Share button: `$$.client.publish(sr.client, {name : 'publisher.setValidForSubmit', payload : true});`
X	X	`publisher.showPanel`	Fired by the publisher when the user selects a canvas app/page in the publisher. This event indicates that the canvas app is being displayed.
	X	`publisher.success`	Fired by the publisher after the Share button is tapped and data is successfully submitted.

Subscription on Publisher Events Visualforce Example:

Use subscription methods to monitor events on your page and respond accordingly. You can place these directly on the page and use them by including the publisher.js library mentioned above.

```
//Fired by the publisher when the user selects a Visualforce action
in the publisher. This event indicates that the Visualforce is
being displayed. Sfdc.canvas.publisher.subscribe({name: "publisher.
showPanel", onData:function(e) {
        //Enables the Share button on the publisher
        Sfdc.canvas.publisher.publish({name: "publisher.
        setValidForSubmit", payload:"true"});
}});

//Fired by the publisher to indicate to the page that the submit
button has been pressed.
Sfdc.canvas.publisher.subscribe({name: "publisher.post",
onData:function(e) {
        //Closes the publisher and refreshes the feed
        Sfdc.canvas.publisher.publish({name: "publisher.close",
        payload:{ refresh:"true"}});
}});
```

Set Canvas App Location and Create the Action

To add a canvas app to the publisher, you must set the location and create the action when you create the canvas app.

Note: Support for Force.com Canvas apps in the publisher, the Chatter feed, and Chatter Mobile is currently available through a pilot program and is available in all new Development Edition organizations. For information on enabling it for your organization, contact salesforce.com.

1. In Salesforce, from Setup, click **Create > Apps**.
2. In the Connected Apps related list, click **New**. Fill out the fields for your canvas app.
3. In the Locations field, select Publisher. You must select this location for your canvas app to appear in the publisher.
4. In the **Canvas App Settings** section, select the **Create Actions Automatically** field. This creates a quick action for the canvas app.

For the canvas app to appear as a publisher action, you must add the action to the global layout.

Create a Canvas Action

If you didn't select the Create Actions Automatically field when you created the canvas app, then you'll need to create the action manually.

1. From Setup, click **Create > Global Actions**
2. Click **New Action.**
3. In the **Action Type** field, select **Custom Canvas.**
4. In the **Canvas App** field, select the canvas app that you want to appear as an action. Only canvas apps that have a location of Publisher will appear in this field.
5. In the **Height** field, enter the height of the canvas app in pixels. This is the initial height of the canvas app when it appears in the publisher. You can use the Force.com Canvas SDK `resize()` method to change the height up to a maximum of 500 pixels.
6. In the **Label** field, enter a value. This value appears as the publisher action title in the user interface.
7. In the **Name** field, enter a unique value with no spaces.
8. Optionally, in the **Icon** field, you can upload an icon by clicking **Change Icon**. You must upload the icon as a static resource before you can change it here.
9. Click **Save**.

Sequence of Publisher Events in Force.com Canvas

Here's the order of publisher events from the perspective of the canvas app:

1. The canvas app listens for `publisher.setupPanel`.
2. The canvas app listens for `publisher.showPanel`.
3. The user interacts with the canvas app, for example, clicks a button or enters some text. The canvas app does any validation required and then fires `publisher.setValidForSubmit`. As a result, the publisher then enables the **Share** button.
4. The canvas app listens for `publisher.getPayload`.
5. The canvas app fires `publisher.setPayload`.
6. The canvas app listens for `publisher.success`.
7. The canvas app listens for `publisher.failure`.
8. The canvas app listens for `publisher.clearPanelState`.

Publisher Canvas App Access

When you display a canvas app inside of a feed item, the context information you receive from the signed request or from a `getContext()` call contains information specific to the feed:

- **Location**—If the canvas app is in the feed, then the `Environment.displayLocation` field contains the value `ChatterFeed`.
- **Parameters**—When you create a feed item that contains a canvas app, you can specify a JSON string as the parameters value. When the canvas app receives the context, the parameters in the feed item will be contained in the `Environment.Parameters` object.
- **Size**—The Environment.Dimensions object contains information about the size of the canvas app.
  - ◊ The canvas app height defaults to 100 pixels.
  - ◊ The canvas app width defaults to 420 pixels, which is the same as the maximum width of a canvas app in the feed.
  - ◊ The maximum height of a canvas app in the feed is 400 pixels.
  - ◊ The maximum width of a canvas app in the feed is 420 pixels.

Publisher Canvas App Access Considerations continued.

◊ This code snippet shows the default size values of a canvas app in the feed:

```
"dimensions":
{
"width": "420px",
"height": "100px",
"maxHeight": "400px",
"maxWidth": "420px"
}
```

◊ The feed is a fixed width of 420 pixels. For example, if you resize your canvas app to be 200 pixels, the feed width remains 420 pixels.

◊ You can use the `resize()` method in the Force.com Canvas SDK to change the values of your canvas app up to the `maxHeight` and `maxWidth`.

Working with Canvas Apps in the Publisher and Chatter Feed

When submitting your request using the **Share** button you can choose what type of feed item you would like to create. We currently support three types:

1. Text Post - This is the standard text type of post. The `TextPost` payload must contain:
   a. `p.feedItemType = "TextPost"`
   b. `p.auxText` - The text that you want to post. We concatenate this text with whatever the user enters into the "What are you working on" field. There must be some text provided (either by the user in the text box, or by your app) or there will be an error.
2. Link Post - This is the standard link type of post. The `LinkPost` payload must contain:
   a. `p.feedItemType = "LinkPost"`
   b. `p.auxText` - This is the text that appears over the link box in the feed item
   c. `p.url` - This is the URL the link will direct to
   d. `p.urlName` - This is the friendly name that appears over the URL in the link box
3. Canvas Post - This is a new feed item type that will create a feed item that can load a `Canvas App` directly in the feed. The `CanvasPost` must contain:
   a. `p.feedItemType = "CanvasPost"`
   b. `p.auxText` - This is the text that appears over the canvas box in the feed item
   c. `p.namespace` - The namespace of your `Canvas App` (if set)
   d. `p.developerName` - The API name of your `Canvas App`
   e. `p.thumbnailUrl` - An HTTPS URL to an icon you would like to display next to the app. If none is provided, the default canvas puzzle icon will be used
   f. `p.parameters` - A JSON string of custom parameters you would like to send to the app. This is an optional value that will be sent in the Signed Request as the "parameters" object
   g. `p.title` - The linked title that will display in the canvas box in the feed item. When clicked, this initiates the canvas app load
   h. `p.description` - A 255 character description that will appear under the link title

Creating Canvas Feed Items

There are two ways to get a canvas app into the feed. The above example demonstrates creating the app from the Publisher. You can also create the canvas feed item directly from the Connect API. The field requirements are the same, but the format would be slightly different. For instance, to create the preceding canvas feed item through the Connect API you would: POST to the feed resource you want, like `https://<instance>.salesforce.com/services/data/v29.0/chatter/feeds/news/<userId>/feed-items`

The message would look like:

```
{
    "body" : {
      "messageSegments" : [ {
        "type" : "Text",
        "text" : "Please Approve my trip: Release Planning at HQ"
      } ]
    },
    "attachment" : {
      "description" : "This is a travel itinerary for Itinerary -
Release Planning at HQ. Click the link to open the Canvas App.",
      "parameters" : "{&quot;itinerary&quot;:&quot;123&quot;}",
      "title" : "Itinerary - Release Planning at HQ",
      "namespacePrefix" : "",
      "developerName" : "Itinerary_App",
      "height" : "100px",
      "thumbnailUrl" : "https://icons.iconarchive.com/icons/
aha-soft/perfect-transport/48/   Airplane-icon.png",
      "attachmentType" : "Canvas"
    }
}
```