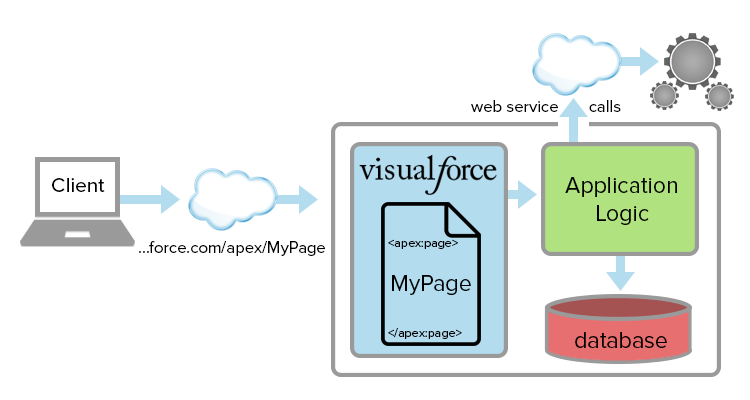
**Visualforce Pages**

Visualforce is a web development framework that enables developers to build sophisticated, custom user interfaces for mobile and desktop apps that can be hosted on the Force.com platform. You can use Visualforce to build apps that align with the styling of Lightning Experience, as well as your own completely custom interface. It is similar to HTML but it's primary use is to access, display and update the organization’s data.

Each tag in visual force language corresponds to some user interface component like section of a page, a list view or a field of an object. Interestingly, it can be easily mixed up wtih HTML markup, CSS style and Java libraries etc.



## An Example Visualforce Page

Here’s a short example of a working Visualforce page.

|  |  |  |
| --- | --- | --- |
|  | <apex:page standardController="Contact" > | |
|  | <apex:form > |

|  |  |
| --- | --- |
|  |  |
|  | <apex:pageBlock title="Edit Contact"> | |

|  |  |
| --- | --- |
|  |  |
|  | <apex:pageBlockSection columns="1"> | |

|  |  |  |
| --- | --- | --- |
|  | <apex:inputField value="{!Contact.FirstName}"/> | |
|  | <apex:inputField value="{!Contact.LastName}"/> |

|  |  |
| --- | --- |
|  | <apex:inputField value="{!Contact.Email}"/> |
|  | <apex:inputField value="{!Contact.Birthdate}"/> | |

|  |  |  |
| --- | --- | --- |
|  | </apex:pageBlockSection> | |
|  |  |

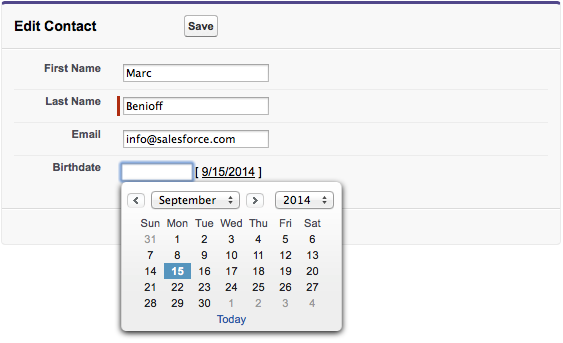
|  |  |
| --- | --- |
|  | <apex:pageBlockButtons > |
|  | <apex:commandButton action="{!save}" value="Save"/> | |

|  |  |  |
| --- | --- | --- |
|  | </apex:pageBlockButtons> | |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | </apex:pageBlock> | |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | </apex:form> | |
|  | </apex:page> |

This page displays a contact data entry form.



**Tools for Visualforce Development**

Before you begin developing Visualforce pages and components, familiarize yourself with the different places to create them:

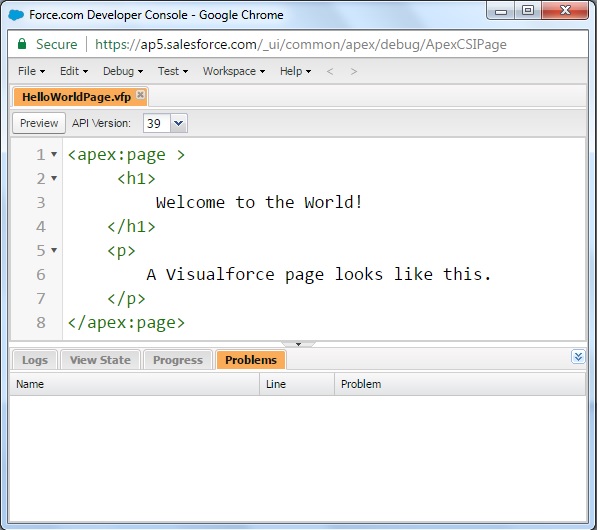
* The best way to build Visualforce is by enabling Visualforce development mode. Visualforce development mode is only available for users with the “Customize Application” permission. Development mode provides you with:
  + A special development footer on every Visualforce page that includes the page’s view state, any associated controller, a link to the component reference documentation, and a page markup editor that offers highlighting, find-replace functionality, and auto-suggest for component tag and attribute names.
  + The ability to define new Visualforce pages just by entering a unique URL.
  + Error messages that include more detailed stack traces than what standard users receive.

To enable Visualforce development mode:

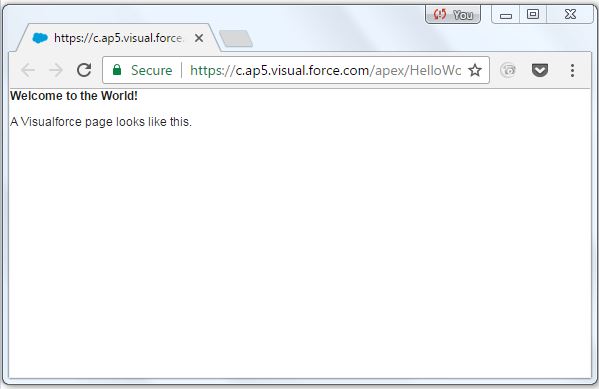
* + From your personal settings, enter Advanced User Details in the Quick Find box, then select **Advanced User Details**. No results? Enter Personal Information in the Quick Find box, then select **Personal Information**.
  + Click **Edit**.
  + Select the Development Mode checkbox.
  + Optionally, select the Show View State in Development Mode checkbox to enable the View State tab on the development footer. This tab is useful for monitoring the performance of your Visualforce pages.
  + Click **Save**.
* You can also develop Visualforce pages through the Salesforce user interface from Setup by entering Visualforce Pages in the Quick Find box, then selecting **Visualforce Pages**. For Visualforce components, from Setup, enterComponents in the Quick Find box, then select **Visualforce Components**.

## Creating a Visualforce Page

Go to the link **developer console -> File -> New -> Visualforce page**. The new window opens askign for a page name. Lets call it HelloworldPage. Next we write the code as shown in the below diagram.

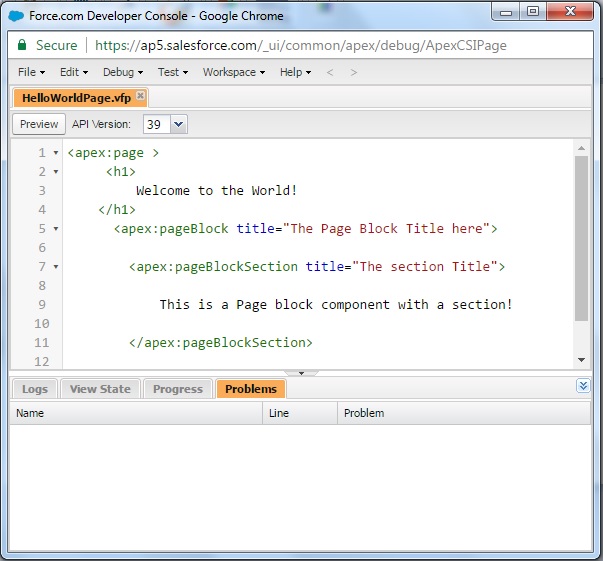


Click Save. Then click on Preview. This opens a new webpage showing the result as below.

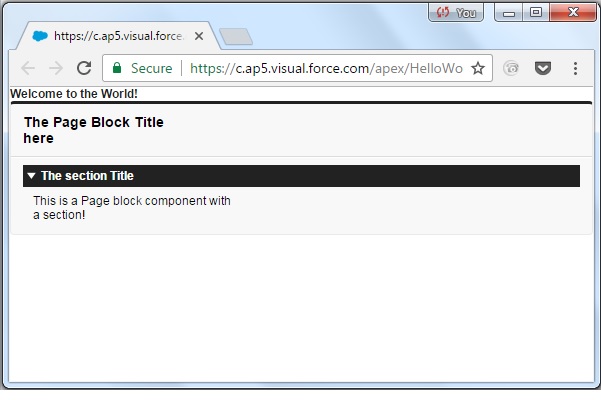


## Adding components

Let's add some user interface components to the above program created. We add a block and a section in that block by using the following code.

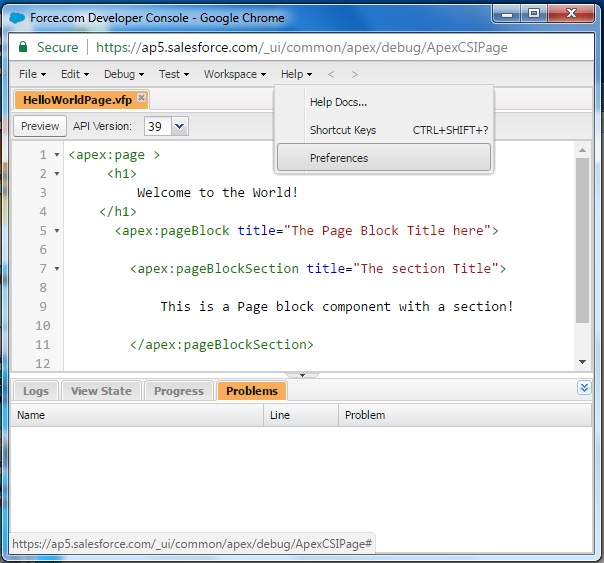


On previeweing the page we get the following output.



## Setting Prefernces

We can set the various settings for easy navigation by going to Help->Prefernces.



# Another way to create visualpage

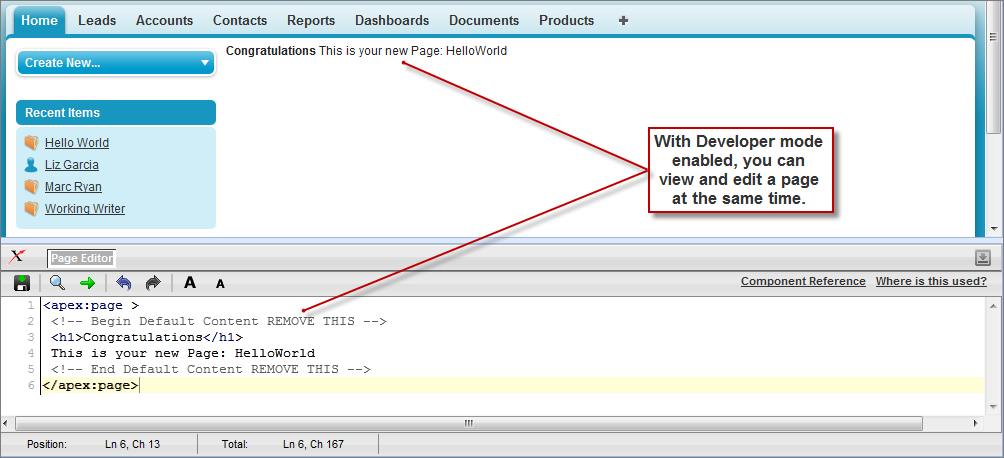
# Creating Your First Page

With development mode enabled, you can create your first Visualforce page by entering a URL for the page in your browser's address bar as follows:

|  |  |
| --- | --- |
| 1 | https://Salesforce\_instance/apex/myNewPageName |

For example, if you want to create a page called “HelloWorld” and your Salesforce organization uses na3.salesforce.com, enter http://na3.salesforce.com/apex/HelloWorld.

Because the page does not yet exist, you are directed to an intermediary page from which you can create your new page. Click **Create Page <myNewPageName>** to create it automatically.

**A New Visualforce Page**

You now have a Visualforce page that includes default text. To edit your new page, click the **Page Editor** bar that appears at the bottom of the browser. It expands to show you the following Visualforce markup:

|  |  |
| --- | --- |
| 1 | <apex:page> |
| 2 | <!-- Begin Default Content REMOVE THIS --> | |

|  |  |
| --- | --- |
| 3 | <h1>Congratulations</h1> |
| 4 | This is your new Apex Page: HelloWorld | |

|  |  |  |
| --- | --- | --- |
| 5 | <!-- End Default Content REMOVE THIS --> | |
| 6 | </apex:page> |

This default markup includes the only required tag for any page— the <apex:page> tag that begins and ends any page markup. Embedded within the start and close <apex:page> tags is plain text, some of which is formatted with a standard HTML tag, <h1>.

As long as you keep the required <apex:page> tag you can add as much plain text or valid HTML to this page as you want. For example, after entering the following code and clicking **Save** in the Page Editor, the page displays the text “Hello World!” in bold:

|  |  |
| --- | --- |
| 1 | <apex:page> |
| 2 | <b>Hello World!</b> | |

|  |  |
| --- | --- |
| 3 | </apex:page> |

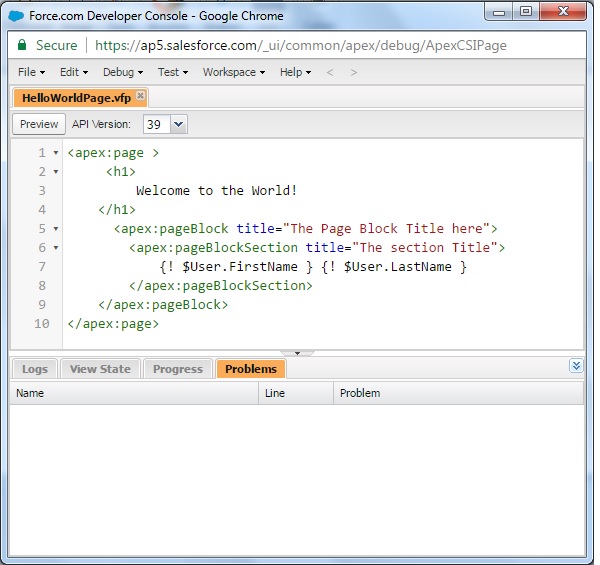
# Variables & Formulas

The syntax of a visualforce expression is as below.

{! expression }

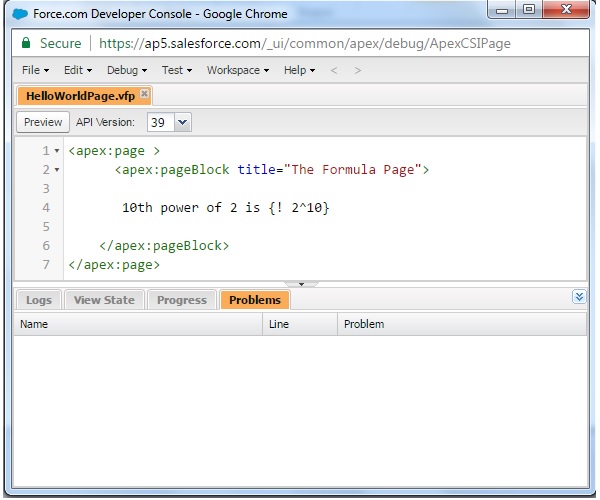
## Example

Lets use the global variable $user. We can write the following code to get the user name First name and login name.



On previeweing the output, we get the following details.

Similarly we can use many other variables and manipulate them using formula as shown in the code below.



On previeweing the output, we get the following details.

# vf_variables_expressions4.JPG

# Displaying Field Values with Visualforce

Visualforce pages use the same expression language as formulas—that is, anything inside {! } is evaluated as an expression that can access values from records that are currently in context. For example, you can display the current user's first name by adding the {!$User.FirstName} expression to a page:

|  |  |
| --- | --- |
| 1 | <apex:page> |
| 2 | Hello {!$User.FirstName}! | |

|  |  |
| --- | --- |
| 3 | </apex:page> |

$User is a global variable that always represents the current user record. All global variables are referenced with a $ symbol. For a list of global variables that you can use in Visualforce, see Global Variables.

To access fields from a record that is not globally available, like a specific account, contact, or custom object record, you need to associate your page with a controller. Controllers provide pages with the data and business logic that make your application run, including the logic that specifies how to access a particular object's records. While you can define a custom controller for any page with Apex, Salesforce includes standard controllers for every standard and custom object.

For example, to use the standard controller for accounts, add the standardController attribute to the <apex:page> tag, and assign it the name of the account object:

|  |  |  |
| --- | --- | --- |
| 1 | <apex:page standardController="Account"> | |
| 2 | Hello {!$User.FirstName}! |

|  |  |
| --- | --- |
| 3 | </apex:page> |

After you save your page, the Accounts tab is highlighted for the page, and the look-and-feel for the components on the page match the Accounts tab. Additionally, you can now access fields on the account record currently in context by using {!account.<fieldName>} expression syntax.

For example, to display an account's name on a page, use {!account.name} in the page markup:

|  |  |  |
| --- | --- | --- |
| 1 | <apex:page standardController="Account"> | |
| 2 | Hello {!$User.FirstName}! |

|  |  |  |
| --- | --- | --- |
| 3 | <p>You are viewing the {!account.name} account.</p> | |
| 4 | </apex:page> |

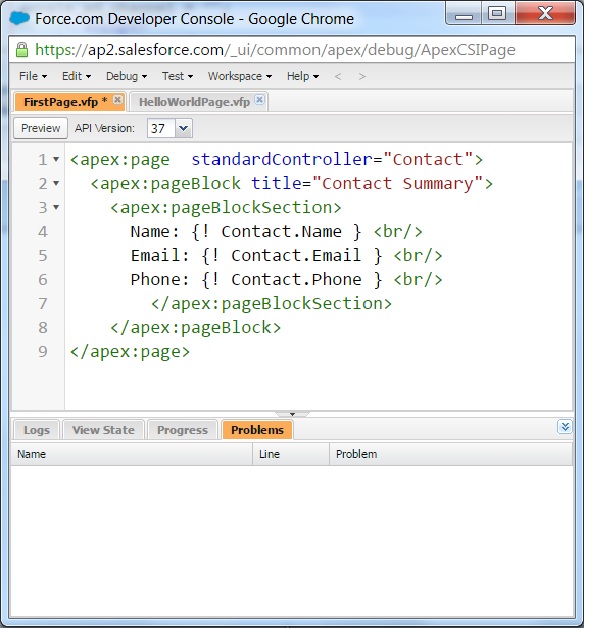
# Standard Controllers

Visualforce consist of many built-in controllers which can be used to access and display data. It works on the MVC (model-view-controller) approach. The controllers intrecat with the database and pull the data fromthe database to view the data through a webpage created by apex page.

To display a specifric record or group of records we need the record ID. When intergrated with other visualforce pages the ID can flow to the controller page automatically. But in a stand alone page we need to specofy the record ID manually to see the controller working.

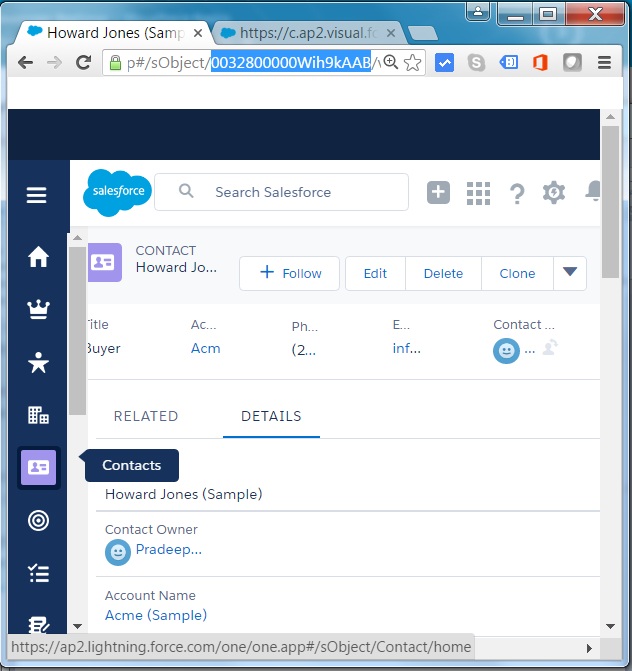
## Example

Lets create a visualforce page to get the summary of a record in the Contact Object. To do this we use the component called **standardController** and put it in a apex block. The below diagram shows the code to achieve this.



Here we dispaly some select fields from the Object. They are Name, Email and phone. If we go to the preview window, we find that the page only displays the labels but no data. That is because we have not associated the result from the controller with any specific record.

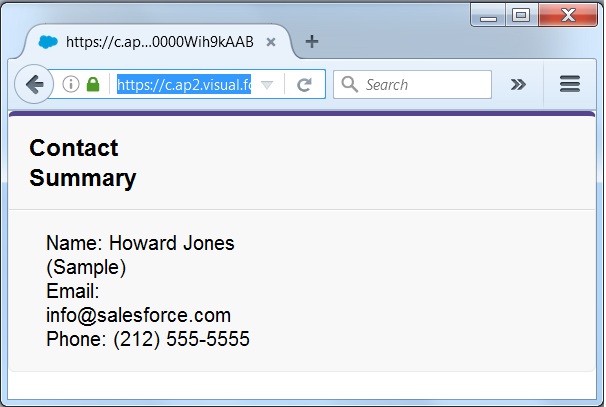
So next we identify a record form the **Contact Object** to be attached to the result form the controller. Open the contacts object and click on any of contact name. It will open the following window from which we capture the ID of the record. The ID is higlighted in the URL. In your environment it will be a similar string of characters.



Finally we add this ID of the record to the YRL of the preview window of the visualforce standrad controller page we created. In the current example the ID of the record is added as shown below.

https://c.ap2.visual.force.com/apex/FirstPage?core.apexpages.request.devconsole=1&id=0032800000Wih9kAAB

On visiting the above URL from the Organization's salesforce accoutn we get the details of the record as shoen below.

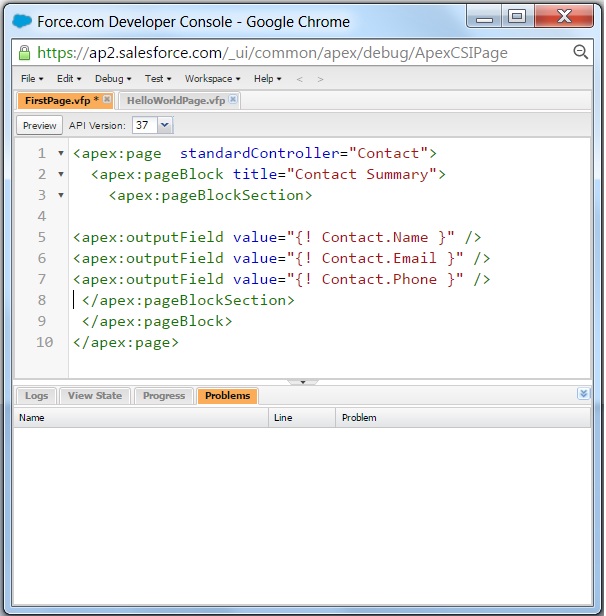


# Records, Fields & Tables

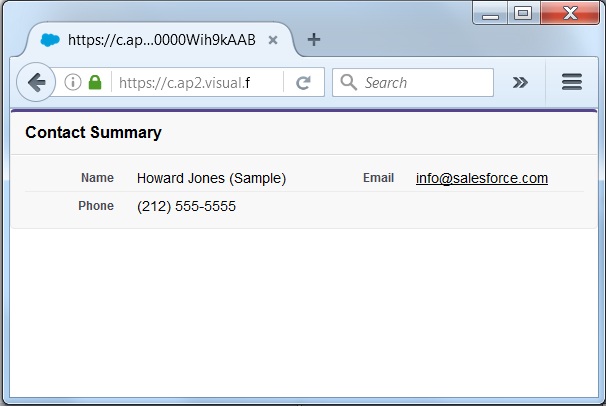
We have already seen how to get the values of a record using the standrad controller. But we can aslo display the field values in a more formatted way by using additional controllers. In this chapter we see how to display the fields of an object and a the data of a details table in a Master-detail relation.

## Display Fields

When we want to display the fields of a record in a formatted manner with column headers, rather than just the labels and values, we can use the **OutputField** option. The below code shows the apex program to display the firled data from Contact. As you can see we do not need the labels to be in place to indicate the field values.

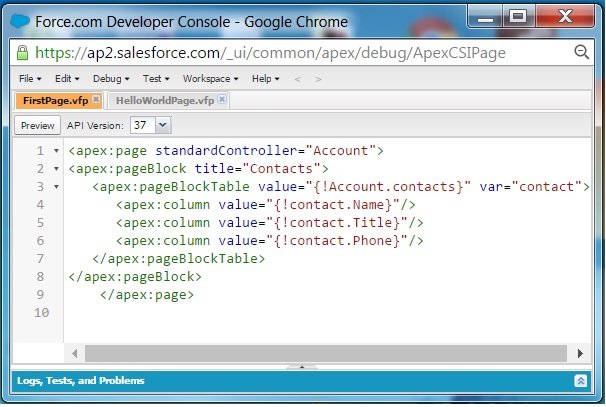


To preview the result of the above coe we use the ID of the record as described in the previous chapter. The result shows the following output showing the field values of the records along wiht column headers.

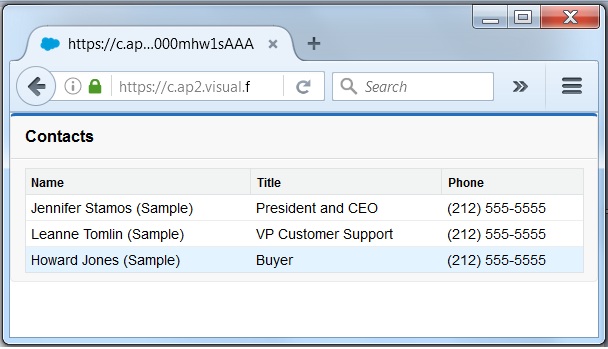


## Display Tables

W can display all the records of a details table by taking a value from a master table. For example we can diaplay all the contacts associated with an account. In such case we use the **iteration component** which in pur case is the contacts table linked to account. Below is the code to display all the contacts associated with the Account table.



On previeweing the putput form the above code we get the below result.



# Using Forms

Salesforce is a data driven platfrom. So vieweing and editing data in it's apps is a fundamental requirement. Like the traditional HTML forms we can create similar forms in saleforce for editing the data in salesforce. It is achieved by using the standard controller with the component named

### Creating Input Form

The Input forms are cretaed using a visualforce page. We open a new visualforce page by using the path **Developer Console -> File -> New -> Visualforce Page**. We write the code as shown below which uses **Contact** as the standard controller and the purpose is to edit the values in the 3 fields - Name, Email and Phone.

But we can improve the above form further by aligning the fields to one column and putting all the input fields and labels in to a block. The below code shows how we are a section and block to thte form and also puttign the fields in one column.

On running the above code for the visualforce page the output we get is as below.

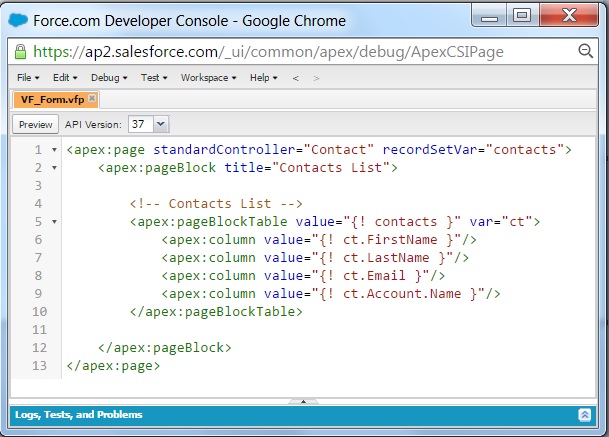
# List Controllers

Many times we need to view a given set of records from an salesforce object applying some filter criteria. This is achieved by using list controllers which allow you to create Visualforce pages that can display or act on a set of records. The standard list controllers can be used in the following set of objects.

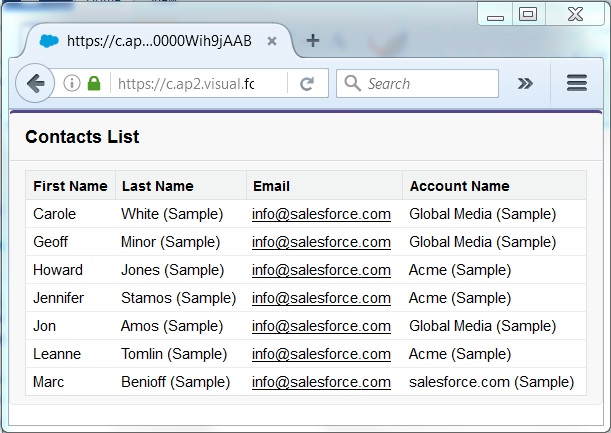
* Account
* Asset
* Campaign
* Case
* Contact
* Contract
* Idea
* Lead
* Opportunity
* Order
* Solution
* User
* Custom objects

## Example

We take the example of contact object. We fetch the records from this object and display it using list controllers. To achieve this we create a apex page with the following code. The code creates a pageblock with column values matching the column names of the contact object.



On running the baove code we get the put put as shown below.



# Static Resources

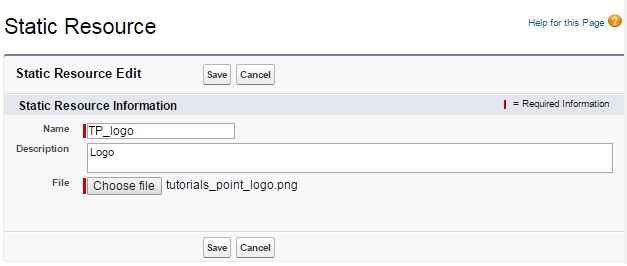
The user interface in a visualforc ebase can display dynamic content whose value keeps chaning based on user responses. But there are times when we need some content which should not change with change in th evalues of other components in th page. For example a image file may be required to remain constant. Such unchanging content in a page are know as static resourcess. The list of static resources used in Salesforec are below.

* Images
* Javascript Files
* Flash files
* CSS files

The steps to create a static resource are as below.

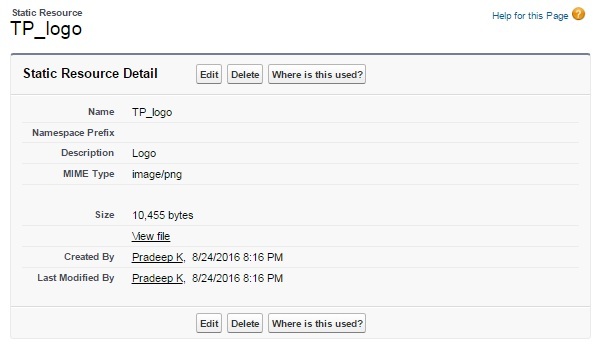
### Create a Static Resource container

Go to **Develop -> Static resource**and mention the values for name, description and file location for the static resource.



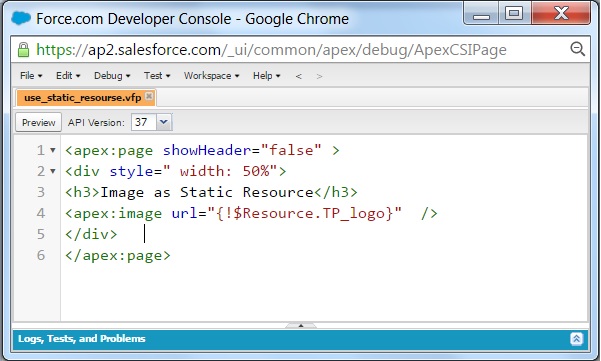
### Upload a Static Resource

Click save in the above screen to get the file attached to the user interface controller.



### Reference the Static Resource

Next we create the apex code below to reference the static object we created above.



### Output

Running the above apex code gives us the below result which shows the static resource.

**Replace the existing code with the following and click Save:**

|  |  |  |
| --- | --- | --- |
| <apex:page standardController="Account" showHeader="true" | | |
| 02 | tabStyle="account" > |

|  |  |
| --- | --- |
| 03 | <style> |
| 04 | .activeTab {background-color: #236FBD; color:white; | |

|  |  |
| --- | --- |
| 05 | background-image:none} |
| 06 | .inactiveTab { background-color: lightgrey; color:black; | |

|  |  |  |
| --- | --- | --- |
| 07 | background-image:none} | |
| 08 | </style> |

|  |  |  |
| --- | --- | --- |
| 09 | <apex:tabPanel switchType="client" selectedTab="tabdetails" | |
| 10 | id="AccountTabPanel" tabClass='activeTab' |

|  |  |
| --- | --- |
| 11 | inactiveTabClass='inactiveTab'> |
| 12 | <apex:tab label="Details" name="AccDetails" id="tabdetails"> | |

|  |  |  |
| --- | --- | --- |
| 13 | <apex:detail relatedList="false" title="true"/> | |
| 14 | </apex:tab> |

|  |  |  |
| --- | --- | --- |
| 15 | <apex:tab label="Contacts" name="Contacts" id="tabContact"> | |
| 16 | <apex:relatedList subject="{!account}" list="contacts" /> |

|  |  |
| --- | --- |
| 17 | </apex:tab> |
| 18 | <apex:tab label="Opportunities" name="Opportunities" | |

|  |  |
| --- | --- |
| 19 | id="tabOpp"> |
| 20 | <apex:relatedList subject="{!account}" | |

|  |  |  |
| --- | --- | --- |
| 21 | list="opportunities" /> | |
| 22 | </apex:tab> |

|  |  |  |
| --- | --- | --- |
| 23 | <apex:tab label="Open Activities" name="OpenActivities" | |
| 24 | id="tabOpenAct"> |

|  |  |  |
| --- | --- | --- |
| 25 | <apex:relatedList subject="{!account}" | |
| 26 | list="OpenActivities" /> |

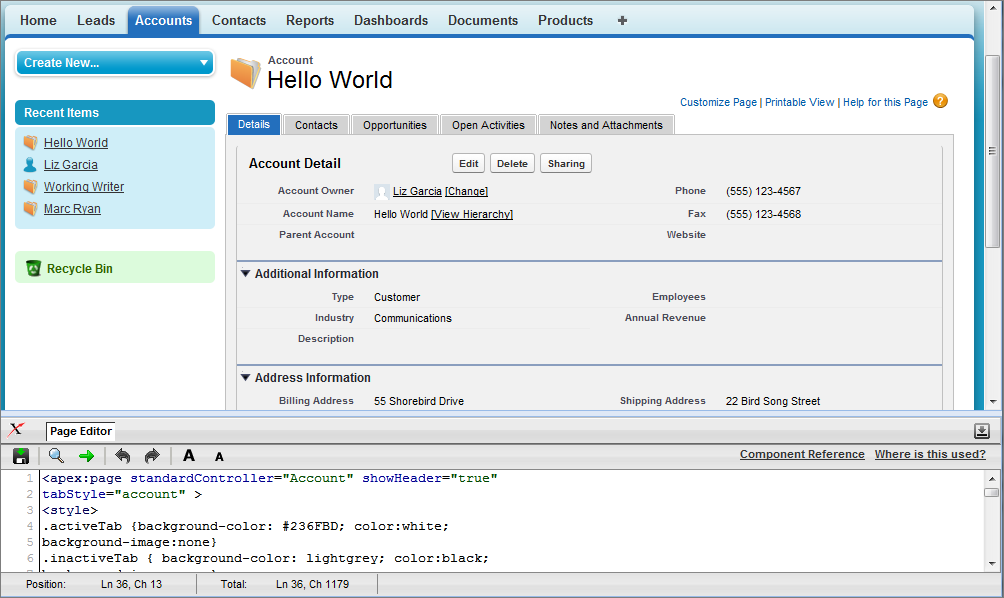
|  |  |
| --- | --- |
| 27 | </apex:tab> |
| 28 | <apex:tab label="Notes and Attachments" | |

|  |  |  |
| --- | --- | --- |
| 29 | name="NotesAndAttachments" id="tabNoteAtt"> | |
| 30 | <apex:relatedList subject="{!account}" |

|  |  |  |
| --- | --- | --- |
| 31 | list="CombinedAttachments" /> | |
| 32 | </apex:tab> |

|  |  |  |
| --- | --- | --- |
| 33 | </apex:tabPanel> | |
| 34 | </apex:page> |

1. Notice that there is no data in the Account page. You need to specify the ID of a particular account in the URL, as you've done with previous pages, for example, https://*Salesforce\_instance*/apex/tabbedAccount?id=001D000000IRt53. After you add in an account ID, your page should display as follows:

****

Things to note about the page markup:

* <style> is actually part of CSS markup, not Visualforce markup. It defines the styles for two types of tabs: activeTab and inactiveTab.
* <apex:tabPanel> is used to generate the tabs. Notice how it uses the following attributes:
  + tabClass attribute: specifies the style class used to display a tab when it is active.
  + inactiveTabClass attribute: specifies the style class used to display a tab when it is inactive.
* Within the definition of the tab panel, is the definition of each child tab component, <apex:tab>. The first tab uses the <apex:detail> tag to return that portion of the detail view for the page:

|  |  |  |
| --- | --- | --- |
| 1 | <apex:tab label="Details" name="AccDetails" id="tabdetails"> | |
| 2 | <apex:detail relatedList="false" title="true"/> |

|  |  |
| --- | --- |
| 3 | </apex:tab> |

* While the rest of the tabs use the <apex:relatedList> to specify the different parts of the account page. The following is the tab for contacts. It uses an existing list of contacts.

|  |  |  |
| --- | --- | --- |
| 1 | <apex:tab label="Contacts" name="Contacts" id="tabContact"> | |
| 2 | <apex:relatedList subject="{!account}" list="contacts" /> |

|  |  |
| --- | --- |
| 3 | </apex:tab> |

Now that you've created a page to display an account with tabs, you can use this page to override the detail view for all accounts.

1. From the object management settings for accounts, go to Buttons, Links, and Actions.
2. Click **Edit** next to View.
3. For Override With select Visualforce Page.
4. From the Visualforce Page drop-down list, select tabbedAccount.
5. Click **Save**.

**Creating and Using Custom Components**

Salesforce provides a library of standard, pre-built components, such as <apex:relatedList> and <apex:dataTable>, that can be used to develop Visualforce pages. In addition, you can build your own custom components to augment this library. This chapter provides an overview of custom components and how to create them:

* What are Custom Components?
* Custom Component Markup
* Using Custom Components in a Visualforce Page
* Custom Component Attributes
* Custom Component Controllers
* Defining Custom Components

**What are Custom Components?**

Similar to the way you can encapsulate a piece of code in a method and then reuse that method several times in a program, you can encapsulate a common design pattern in a custom component and then reuse that component several times in one or more Visualforce pages.

For example, suppose you want to create a photo album using Visualforce pages. Each photo in the album has its own border color, and a text caption that displays beneath it. Rather than repeating the Visualforce markup required for displaying every photo in the album, you can define a custom component named singlePhoto that has attributes for image, border color, and caption, and then uses those attributes to display the image on the page. Once defined, every Visualforce page in your organization can leverage the singlePhoto custom component in the same way as a page can leverage standard components such as <apex:dataTable> or <apex:relatedList>.

Unlike page templates, which also enable developers to reuse markup, custom components provide more power and flexibility because:

* Custom components allow developers to define attributes that can be passed in to each component. The value of an attribute can then change the way the markup is displayed on the final page, and the controller-based logic that executes for that instance of the component. This behavior differs from that of templates, which do not have a way of passing information from the page that uses a template to the template's definition itself.
* Custom component descriptions are displayed in the application's component reference dialog alongside standard component descriptions. Template descriptions, on the other hand, can only be referenced through the Setup area of Salesforce because they are defined as pages.

# apex:form

A section of a Visualforce page that allows users to enter input and then submit it with an <apex:commandButton> or <apex:commandLink>. The body of the form determines the data that is displayed and the way it's processed. It's a best practice to use only one <apex:form> tag in a page or custom component.

As of API version 18.0, this tag can't be a child component of <apex:repeat>.

This component supports [HTML pass-through attributes](https://developer.salesforce.com/docs/atlas.en-us.210.0.pages.meta/pages/pages_html_features_pass_through_attributes.htm) using the "html-" prefix. Pass-through attributes are attached to the generated <form> tag.

## Example

|  |  |  |
| --- | --- | --- |
| 01 | <!-- For this example to render properly, you must associate the Visualforce page | |
| 02 | with a valid case record in the URL. |

|  |  |  |
| --- | --- | --- |
| 03 | For example, if 001D000000IRt53 is the case ID, the resulting URL should be: | |
| 04 | https://Salesforce\_instance/apex/myPage?id=001D000000IRt53 |

|  |  |  |
| --- | --- | --- |
| 05 | See the Visualforce Developer's Guide Quick Start Tutorial for more information. --> | |
| 06 |  |

|  |  |
| --- | --- |
| 07 |  |
| 08 | <apex:page standardController="Case" recordSetVar="cases" tabstyle="case"> | |

|  |  |  |
| --- | --- | --- |
| 09 | <apex:form id="changeStatusForm"> | |
| 10 | <apex:pageBlock > |

|  |  |
| --- | --- |
| 11 | <apex:pageMessages /> |
| 12 | <apex:pageBlockButtons> | |

|  |  |  |
| --- | --- | --- |
| 13 | <apex:commandButton value="Save" action="{!save}"/> | |
| 14 | </apex:pageBlockButtons> |

|  |  |  |
| --- | --- | --- |
| 15 | <apex:pageBlockTable value="{!cases}" var="c"> | |
| 16 | <apex:column value="{!c.casenumber}"/> |

|  |  |  |
| --- | --- | --- |
| 17 | <apex:column value="{!c.account.name}"/> | |
| 18 | <apex:column value="{!c.contact.name}"/> |

|  |  |
| --- | --- |
| 19 | <apex:column value="{!c.subject}"/> |
| 20 | <apex:column headerValue="Status"> | |

|  |  |  |
| --- | --- | --- |
| 21 | <apex:inputField value="{!c.Status}"/> | |
| 22 | </apex:column> |

|  |  |  |
| --- | --- | --- |
| 23 | </apex:pageBlockTable> | |
| 24 | </apex:pageBlock> |

|  |  |  |
| --- | --- | --- |
| 25 | </apex:form> | |
| 26 | </apex:page> |

The example above renders the following HTML:

|  |  |
| --- | --- |
| 01 | <!-- allows you to change the status of your cases --> |
| 02 | <form id="j\_id0:changeStatusForm" name="j\_id0:changeStatusForm" method="post" | |

|  |  |  |
| --- | --- | --- |
| 03 | action="/apex/sandbox" enctype="application/x-www-form-urlencoded"> | |
| 04 | <!-- opening div tags --> |

|  |  |  |
| --- | --- | --- |
| 05 | <table border="0" cellpadding="0" cellspacing="0"> | |
| 06 | <tr> |

|  |  |
| --- | --- |
| 07 | <td class="pbTitle"> </td> |
| 08 | <td id="j\_id0:changeStatusForm:j\_id1:j\_id29" class="pbButton"> | |

|  |  |
| --- | --- |
| 09 | <input type="submit" |
| 10 | name="j\_id0:changeStatusForm:j\_id1:j\_id29:j\_id30" | |

|  |  |  |
| --- | --- | --- |
| 11 | value="Save" class="btn"/> | |
| 12 | </td> |

|  |  |
| --- | --- |
| 13 | </tr> |
| 14 | </table> | |

|  |  |
| --- | --- |
| 15 |  |
| 16 | <div class="pbBody"> | |

|  |  |  |
| --- | --- | --- |
| 17 | <table class="list" border="0" cellpadding="0" cellspacing="0"> | |
| 18 | <colgroup span="5"/> |

|  |  |
| --- | --- |
| 19 | <thead> |
| 20 | <tr class="headerRow "> | |

|  |  |
| --- | --- |
| 21 | <th class="headerRow  " scope="col">Case Number</th> |
| 22 | <th class="headerRow " scope="col">Account Name</th> | |

|  |  |
| --- | --- |
| 23 | <th class="headerRow  " scope="col">Name</th> |
| 24 | <th class="headerRow  " scope="col">Subject</th> | |

|  |  |  |
| --- | --- | --- |
| 25 | <th class="headerRow  " scope="col">Status</th> | |
| 26 | </tr> |

|  |  |  |
| --- | --- | --- |
| 27 | </thead> | |
| 28 |  |

|  |  |
| --- | --- |
| 29 | <tbody> |
| 30 | <tr class="dataRow even  first "> | |

|  |  |
| --- | --- |
| 31 | <td class="dataCell"><span>00001000</span></td> |
| 32 | <td class="dataCell"><span>Edge Communications</span></td> | |

|  |  |
| --- | --- |
| 33 | <td class="dataCell"><span>Rose Gonzalez</span></td> |
| 34 | <td class="dataCell"><span>Starting generator after electrical failure</span></td> | |

|  |  |  |
| --- | --- | --- |
| 35 | <td class="dataCell"> | |
| 36 | <select> |

|  |  |
| --- | --- |
| 37 | <option value="">--None--</option> |
| 38 | <option value="New">New</option> | |

|  |  |  |
| --- | --- | --- |
| 39 | <option value="Working" selected="selected">Working</option> | |
| 40 | <option value="Escalated">Escalated</option> |

|  |  |  |
| --- | --- | --- |
| 41 | <option value="Closed">Closed</option> | |
| 42 | </select> |

|  |  |  |
| --- | --- | --- |
| 43 | </td> | |
| 44 | </tr> |

|  |  |
| --- | --- |
| 45 |  |
| 46 | <tr class="dataRow odd last "> | |

|  |  |
| --- | --- |
| 47 | <td class="dataCell"><span>00001027</span></td> |
| 48 | <td class="dataCell"><span>Joyce Bookings</span></td> | |

|  |  |
| --- | --- |
| 49 | <td class="dataCell"><span>Andy Young</span></td> |
| 50 | <td class="dataCell"><span>Checking paper jam</span></td> | |

|  |  |  |
| --- | --- | --- |
| 51 | <td class="dataCell"> | |
| 52 | <select> |

|  |  |
| --- | --- |
| 53 | <option value="">--None--</option> |
| 54 | <option value="New">New</option> | |

|  |  |  |
| --- | --- | --- |
| 55 | <option value="Working" selected="selected">Working</option> | |
| 56 | <option value="Escalated">Escalated</option> |

|  |  |  |
| --- | --- | --- |
| 57 | <option value="Closed">Closed</option> | |
| 58 | </select> |

|  |  |  |
| --- | --- | --- |
| 59 | </td> | |
| 60 | </tr> |

|  |  |  |
| --- | --- | --- |
| 61 | </tbody> | |
| 62 | </table> |

|  |  |
| --- | --- |
| 63 | </div> |
| 64 | <!-- closing div tags --> | |

|  |  |
| --- | --- |
| 65 | </form> |

## Attributes

| **Attribute Name** | **Attribute Type** | **Description** | **Required?** | **API Version** | **Access** |
| --- | --- | --- | --- | --- | --- |
| accept | String | A comma-separated list of content types that a server processing this form can handle. Possible values for this attribute include "text/html", "image/png", "image/gif", "video/mpeg", "text/css", and "audio/basic". For more information, including a complete list of possible values, see [the W3C specifications](http://www.w3.org/TR/html4/types.html#type-content-type). |  | 10.0 | global |
| acceptcharset | String | A comma-separated list of character encodings that a server processing this form can handle. If not specified, this value defaults to "UNKNOWN". |  | 10.0 | global |
| dir | String | The direction in which the generated HTML component should be read. Possible values include "RTL" (right to left) or "LTR" (left to right). |  | 10.0 | global |
| enctype | String | The content type used to submit the form to the server. If not specified, this value defaults to "application/x-www-form-urlencoded". |  | 10.0 | global |
| forceSSL | Boolean | The form will be submitted using SSL, regardless of whether the page itself was served with SSL. The default is false. If the value is false, the form will be submitted using the same protocol as the page. If forceSSL is set to true, when the form is submitted, the page returned will use SSL. |  | 14.0 |  |
| id | String | An identifier that allows the form component to be referenced by other components in the page. |  | 10.0 | global |
| lang | String | The base language for the generated HTML output, for example, "en" or "en-US". For more information on this attribute, see [the W3C specifications](http://www.w3.org/TR/REC-html40/struct/dirlang.html). |  | 10.0 | global |
| onclick | String | The JavaScript invoked if the onclick event occurs--that is, if the user clicks the form. |  | 10.0 | global |
| ondblclick | String | The JavaScript invoked if the ondblclick event occurs--that is, if the user clicks the form twice. |  | 10.0 | global |
| onkeydown | String | The JavaScript invoked if the onkeydown event occurs--that is, if the user presses a keyboard key. |  | 10.0 | global |
| onkeypress | String | The JavaScript invoked if the onkeypress event occurs--that is, if the user presses or holds down a keyboard key. |  | 10.0 | global |
| onkeyup | String | The JavaScript invoked if the onkeyup event occurs--that is, if the user releases a keyboard key. |  | 10.0 | global |
| onmousedown | String | The JavaScript invoked if the onmousedown event occurs--that is, if the user clicks a mouse button. |  | 10.0 | global |
| onmousemove | String | The JavaScript invoked if the onmousemove event occurs--that is, if the user moves the mouse pointer. |  | 10.0 | global |
| onmouseout | String | The JavaScript invoked if the onmouseout event occurs--that is, if the user moves the mouse pointer away from the form. |  | 10.0 | global |
| onmouseover | String | The JavaScript invoked if the onmouseover event occurs--that is, if the user moves the mouse pointer over the form. |  | 10.0 | global |
| onmouseup | String | The JavaScript invoked if the onmouseup event occurs--that is, if the user releases the mouse button. |  | 10.0 | global |
| onreset | String | The JavaScript invoked if the onreset event occurs--that is, if the user clicks the reset button on the form. |  | 10.0 | global |
| onsubmit | String | The JavaScript invoked if the onsubmit event occurs--that is, if the user clicks the submit button on the form. |  | 10.0 | global |
| prependId | Boolean | A Boolean value that specifies whether or not this form should prepend its ID to the IDs of its child components during the clientid generation process. If not specified, the value defaults to true. |  | 10.0 | global |
| rendered | Boolean | A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true. |  | 10.0 | global |
| style | String | The style used to display the form component, used primarily for adding inline CSS styles. |  | 10.0 | global |
| styleClass | String | The style class used to display the form component, used primarily to designate which CSS styles are applied when using an external CSS stylesheet. |  | 10.0 | global |
| target | String | The name of the frame that displays the response after the form is submitted. Possible values for this attribute include "\_blank", "\_parent", "\_self", and "\_top". You can also specify your own target names by assigning a value to the name attribute of a desired destination. |  | 10.0 | global |
| title | String | The text to display as a tooltip when the user's mouse pointer hovers over this component. |  | 10.0 | global |

# apex:gaugeSeries

A data series that shows progress along a specific metric. At a minimum you must specify the fields in the data collection to use as label and value pair for the gauge level to be shown. The readability of a gauge chart benefits when you specify meaningful values for the minimum and maximum along the associated <apex:axis>, which must be of type "gauge".

**Note:** This component must be enclosed within an <apex:chart> component. You should put only one <apex:gaugeSeries>in a chart.

## Example

|  |  |
| --- | --- |
| 1 | <!-- Page: --> |
| 2 | <apex:chart height="250" width="450" animate="true" legend="true" data="{!data}"> | |

|  |  |  |
| --- | --- | --- |
| 3 | <apex:axis type="gauge" position="left" margin="-10" | |
| 4 | minimum="0" maximum="100" steps="10"/> |

|  |  |  |
| --- | --- | --- |
| 5 | <apex:gaugeSeries dataField="data1" highlight="true" tips="true" donut="25" | |
| 6 | colorSet="#F49D10, #ddd"> |

|  |  |  |
| --- | --- | --- |
| 7 | <apex:chartLabel display="over"/> | |
| 8 | </apex:gaugeSeries> |

|  |  |
| --- | --- |
| 9 | </apex:chart> |

## Attributes

| **Attribute Name** | **Attribute Type** | **Description** | **Required?** | **API Version** | **Access** |
| --- | --- | --- | --- | --- | --- |
| colorSet | String | A set of color values used as the gauge level fill colors. Colors are specified as HTML-style (hexadecimal) colors, and should be comma separated. For example, #00F,#0F0. |  | 26.0 |  |
| dataField | String | The field in the records provided in the chart data from which to retrieve the data value for the gauge level. Only the first record is used. | Yes | 26.0 |  |
| donut | Integer | An integer representing the radius of the hole to place in the center of the gauge chart, as a percentage of the radius of the gauge. The default of 0 creates a gauge chart with no hole, that is, a half-circle. |  | 26.0 |  |
| highlight | Boolean | A Boolean value that specifies whether each gauge level should be highlighted when the mouse pointer passes over it. If not specified, this value defaults to true. |  | 26.0 |  |
| id | String | An identifier that allows the chart component to be referenced by other components on the page. |  | 26.0 | global |
| labelField | String | The field in the records provided in the chart data from which to retrieve the label for the gauge level. Only the first record is used. If not specified, this value defaults to "name". |  | 23.0 |  |
| needle | Boolean | A Boolean value that specifies whether to show the gauge needle or not. Defaults to false, don't show the needle. |  | 26.0 |  |
| rendered | Boolean | A Boolean value that specifies whether the chart series is rendered in the chart. If not specified, this value defaults to true. |  | 26.0 |  |
| rendererFn | String | A string that specifies the name of a JavaScript function that augments or overrides how gauge elements are rendered. Implement to provide additional styling or to augment data. |  | 26.0 |  |
| tips | Boolean | A Boolean value that specifies whether to display a tooltip for the gauge level when the mouse pointer passes over it. The format of the tip is <labelField>: <dataField>. If not specified, this value defaults to true. |  | 26.0 |  |

# apex:iframe

A component that creates an inline frame within a Visualforce page. A frame allows you to keep some information visible while other information is scrolled or replaced.

This component supports [HTML pass-through attributes](https://developer.salesforce.com/docs/atlas.en-us.210.0.pages.meta/pages/pages_html_features_pass_through_attributes.htm) using the "html-" prefix. Pass-through attributes are attached to the generated <iframe> tag.

## Example

|  |  |
| --- | --- |
| 1 | <apex:iframe src="[http://www.salesforce.com](http://www.salesforce.com/)" scrolling="true" id="theIframe"/> |

The example above renders the following HTML:

|  |  |
| --- | --- |
| 1 | <iframe height="600px" id="theIframe" name="theIframe" src="[http://www.salesforce.com](http://www.salesforce.com/)" width="100%"></iframe> |

## Attributes

| **Attribute Name** | **Attribute Type** | **Description** | **Required?** | **API Version** | **Access** |
| --- | --- | --- | --- | --- | --- |
| frameborder | Boolean | A Boolean value that specifies whether a border should surround the inline frame. If not specified, this value defaults to false. |  | 10.0 | global |
| height | String | The height of the inline frame, expressed either as a percentage of the total available vertical space (for example height="50%"), or as the number of pixels (for example, height="300px"). If not specified, this value defaults to 600px. |  | 10.0 | global |
| id | String | An identifier that allows the inline frame component to be referenced by other components in the page. |  | 10.0 | global |
| rendered | Boolean | A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true. |  | 10.0 | global |
| scrolling | Boolean | A Boolean value that specifies whether the inline frame can be scrolled. If not specified, this value defaults to true. |  | 10.0 | global |
| src | String | The URL that specifies the initial contents of the inline frame. This URL can either be an external website, or another page in the application. |  | 10.0 | global |
| title | String | The text to display as a tooltip when the user's mouse pointer hovers over this component. |  | 10.0 | global |
| width | String | The width of the inline frame, expressed either as a percentage of the total available horizontal space (for example width="80%"), or as the number of pixels (for example, width="600px"). |  | 10.0 | global |

# apex:include

A component that inserts a second Visualforce page into the current page. The entire page subtree is injected into the Visualforce DOM at the point of reference and the scope of the included page is maintained.

If content should be stripped from the included page, use the <apex:composition> component instead.

## Example

|  |  |
| --- | --- |
| 01 | <!-- Page: --> |
| 02 | <apex:page id="thePage"> | |

|  |  |  |
| --- | --- | --- |
| 03 | <apex:outputText value="(page) This is the page."/><br/> | |
| 04 | <apex:include pageName="include"/> |

|  |  |  |
| --- | --- | --- |
| 05 | </apex:page> | |
| 06 |  |

|  |  |
| --- | --- |
| 07 | <!-- Page: include --> |
| 08 | <apex:page id="theIncludedPage"> | |

|  |  |  |
| --- | --- | --- |
| 09 | <apex:outputText value="(include) This is text from another page."/> | |
| 10 | </apex:page> |

The example above renders the following HTML:

|  |  |
| --- | --- |
| 1 | (page) This is the page.<br/> |
| 2 | <span id="thePage:include">(include) This is text from another page.</span> | |

|  |  |
| --- | --- |
| 3 |  |

## Attributes

| **Attribute Name** | **Attribute Type** | **Description** | **Required?** | **API Version** | **Access** |
| --- | --- | --- | --- | --- | --- |
| id | String | An identifier that allows the inserted page to be referenced by other components in the page. |  | 10.0 | global |
| pageName | ApexPages.PageReference | The Visualforce page whose content should be inserted into the current page. For this value, specify the name of the Visualforce page or use merge-field syntax to reference a page or PageReference. | Yes | 10.0 | global |
| rendered | Boolean | A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true. |  | 10.0 | global |