Visualforce Pages

Visualforce pages are webpages that belong to Salesforce. These webpages are created using a unique tag-based Mark-up language. It is similar to HTML but it's primary use is to access, display and update the organization’s data. The page is accessed by using a URL similar to that of a traditional webserver page.

## Creating a Visualforce Page

Go to the link **developer console → File → New → Visualforce page**. The new window opens asking for a page name. Let us now call it **HelloworldPage**.

<apex:page >

<h1>

First Page

</h1>

<p>

Test page for learning

</p>

<apex:pageBlock title="First Block">

<apex:pageBlockSection title="Page block section">

This is page block section

</apex:pageBlockSection>

</apex:pageBlock>

</apex:page>

## Setting Preferences

We can set the various settings for easy navigation by going to **Help → Preferences**.

# Variables & Formulas

## Example

Let us use the global variable **$user**. We can write the following code to get the username, First name and login name.

# Expression:

{! expression }

# {! $User.FirstName} {! $User.LastName}

# Standard Controllers

Visualforce consists 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 interact with the database and pull the data from the database to view the data through a webpage created by apex page.

## Example

Let us 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 an apex block. The diagram given below shows the code to achieve this.

<apex:page standardController="Contact" >

<apex:pageBlock title="Contact Details">

<apex:pageBlockSection>

Name : {! Contact.Name} <br/>

Email : {! Contact.Email} <br/>

Phone : {! Contact.Phone} <br/>

</apex:pageBlockSection>

</apex:pageBlock>

</apex:page>

# Records, Fields and Tables

## 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 code given below shows the apex program to display the filled data from Contact. As you can see, we do not need the labels to be in place to indicate the field values.

# <apex:pageBlock>

# <apex:pageBlockSection>

# <apex:outputField value="{! Contact.Name}"></apex:outputField>

# <apex:outputField value="{! Contact.Email}"></apex:outputField>

# <apex:outputField value="{! Contact.Phone}"></apex:outputField>

# </apex:pageBlockSection>

# </apex:pageBlock>

## Display Tables

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

<apex:page standardController="Account" >

{! contact.Name}

<apex:pageBlock title="Account Details">

<apex:pageBlockTable value="{! Account.contacts}" var="contact">

<apex:column value="{! contact.Name}"></apex:column>

<apex:column value="{! contact.Title}"></apex:column>

<apex:column value="{! contact.Phone}"></apex:column>

</apex:pageBlockTable>

</apex:pageBlock>

</apex:page>

# Using Forms

# <apex:page standardController="Account" >

# <h1>

# Edit Account

# </h1>

# <apex:form>

# <apex:inputField value="{! Account.Name}"/>

# 

# <apex:inputField value="{! Account.Phone}"/>

# <apex:commandButton action="{! save }" value="Save"/>

# 

# </apex:form>

# </apex:page>

# List Controllers

# <apex:page standardController="Account" recordSetVar="accounts" >

# <apex:pageBlock title="List">

# <apex:pageBlockTable value="{! accounts}" var="ac">

# <apex:column value="{! ac.Name }"/>

# <apex:column value="{! ac.Phone }"/>

# </apex:pageBlockTable>

# </apex:pageBlock>

# </apex:page>

# Static Resources

The user interface in a Visualforce base can display dynamic content the value of which keeps changing based on user responses. But there are times when we need some content which should not change with the change in values of other components in the page. For example, an image file may be required to remain constant. Such content which cannot be changed in a page is known as a static resource.

Following are a few static resources in Salesforce −

* 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.



<apex:page showHeader="true">

<div style="width:50%;">

<h1>

Images

</h1>

<apex:image url="{!$Resource.Static}"/>

</div>

</apex:page>